UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Coercion-resistance in electronic voting: design and analysis

THÈSE

présentée et soutenue publiquement le 23 juin 2023

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Quentin Yang

Composition du jury

Rapporteurs :	Adeline Roux-Langlois	Chargée de recherche, CNRS, Caen, France
	Damien Vergnaud	Professeur, Université de Sorbonne, Paris, France
Examinateurs :	Henri Gilbert	Responsable de laboratoire, ANSSI, Paris, France
	Marc Joye	Directeur scientifique, Zama, France
	Vanessa Teague	Associate Professor, ANU, Canberra, Australia
Encadrants :	Véronique Cortier	Directrice de recherche, CNRS, Nancy, France
	Pierrick Gaudry	Directeur de recherche, CNRS, Nancy, France
Président du jury :	Simon Perdrix	Directeur de recherche, Inria, Nancy, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Remerciements

Je remercie Isabelle pour ses desserts.

Un grand merci à toute l'équipe de la cantine : c'est vous qui faites du laboratoire un endroit si spécial.

Merci aussi à mes co-bureaux, qui ont su m'accompagner dans la folie. Certains se disent insupportables, pourtant j'ai savouré leur présence.

Merci à toi Véronique : tu es devenue l'agaçante petite voix dans ma tête qui dit que ce n'est pas bien défini, et ça n'a pas de prix.

Merci à toi Pierrick : tes conseils bienveillants et ta culture ont été des balises sur mon chemin, et tu as su souligner mes erreurs tout en les pardonnant... même si, à la fin, il est temps que ça se termine.

Contents

Introduction 1

Part I Preliminaries 10

Chapter 1 Security in electronic voting 11

1.1 The fundamental notions of electronic voting . . . 11

1.1.1 The generic structure of electronic voting protocols . . . 11

1.1.2 Formalization and notations . . . 14

1.1.3 The Helios voting protocol . . . 16

1.1.4 Classical attacks in electronic voting . . . 18

1.2 Addressing verifiability . . . 20

1.2.1 Step by step verification . . . 20

1.2.2 End-to-end verifiability . . . 20

1.3 Four notions of privacy . . . 23

1.3.1 Vote swapping and Benaloh privacy . . . 23

1.3.2 Ballot privacy and ideal tally . . . 25

1.3.3 A quantitative definition of privacy . . . 28

1.3.4 Our approach: comparing a real and an ideal process . . . 29

1.3.5 Encountering a new definition: the survival manual . . . 31

Chapter 2 Cryptography in electronic voting 33

2.1 Computational assumptions in electronic voting . . . 33

2.1.1 Algebraic notations for cryptography . . . 33

2.1.2 The decisional Diffie-Hellman assumption . . . 33

2.1.3 The random oracle model . . . 34

2.2 Encrypting a ballot to preserve privacy . . . 37

2.2.1 Public key encryption . . . 37

2.2.2 The ElGamal encryption scheme . . . 39

2.2.3 Threshold cryptography . . . 41

2.2.4 The Paillier encryption scheme . . . 44
2.3 Zero Knowledge Proofs in electronic voting . . . 46
2.3.1 Introduction to Zero Knwoledge Proofs . . . 46
2.3.2 Generalization . . . 48
2.3.3 Proof of partial knowledge . . . 49
2.3.4 Non-interactive proofs . . . 50
2.3.5 Short proofs and what they can really do . . . 52
2.4 The most commonly used ZKP . . . 54
2.4.1 A basic example: proving the validity of a ballot . . . 54
2.4.2 Proof of correct decryption . . . 55
2.4.3 Mixnets and their applications . . . 55
2.4.4 Plaintext Equivalence Tests . . . 59
2.4.5 Designated Verifier Zero Knowledge Proofs . . . 60
2.4.6 Cryptographic signatures derived from PoK . . . 60

Chapter 3 Security proofs in electronic voting **62**
3.1 Cryptographic reductions and game hops . . . 62
3.1.1 Game hops . . . 62
3.1.2 The hybrid lemma . . . 64
3.2 Known results in the random oracle model . . . 67
3.2.1 Extracting a witness from a proof of knowledge . . . 67
3.2.2 Good practices for non-interactive proofs . . . 68
3.3 Universally composable security . . . 73
3.3.1 Presentation of the framework . . . 73
3.3.2 The composition theorem . . . 75
3.3.3 Programmable random oracle model . . . 77
3.3.4 An illustrative example: synchronous broadcast . . . 77

Part II Secure Tally-Hiding **82**

Chapter 4 Multi-party computation for electronic voting **83**
4.1 Three popular approaches for multi-party computation . . . 84
4.1.1 Garbled circuits . . . 84
4.1.2 Linear secret sharing schemes . . . 85
4.1.3 Fully homomorphic encryption . . . 86
4.2 The arithmetic blackbox for Paillier encrypted integers . . . 87
4.2.1 MPC from threshold homomorphic encryption . . . 87

4.2.2 Known MPC protocols in the ABB framework . . . 88
4.2.3 Range proofs for Paillier-encrypted integers . . . 88
4.2.4 Comparing two Paillier encrypted integers . . . 90
4.3 The conditional gate protocol in the ElGamal setting . . . 93
4.3.1 Presentation of the protocol . . . 93
4.3.2 Universal verifiability . . . 96
4.3.3 Comparison with the multiplication protocol . . . 96
4.4 Security of the conditional gate in the SUC framework . . . 98
4.4.1 Proof strategy for the conditional gate . . . 98
4.4.2 The rerandomization . . . 100
4.4.3 The threshold decryption . . . 102
4.4.4 The round of communications . . . 104
4.4.5 The conditional gate protocol is SUC-secure . . . 108

Chapter 5 A toolbox for verifiable tally-hiding **109**
5.1 The basic primitives of the MPC toolbox . . . 109
5.1.1 Logical operations on encrypted data . . . 109
5.1.2 Application to elementary arithmetic . . . 112
5.1.3 Comparisons and tie breaking . . . 112
5.2 Advanced algorithms . . . 114
5.2.1 Multiplication and division . . . 114
5.2.2 Solving ordering related problems . . . 115
5.2.3 Aggregation of several encrypted binary values . . . 118
5.2.4 Different communication/computation trade-offs . . . 119
5.3 Comparison with other approaches . . . 122
5.3.1 Comparison with Ordinos . . . 123
5.3.2 Public tally hiding . . . 124

Chapter 6 Application of the toolbox to electronic voting **126**
6.1 Homomorphic tally for the Condorcet methods . . . 127
6.1.1 Existing approaches for Condorcet methods . . . 127
6.1.2 A new proof of well-formedness for homomorphic ranked voting . . . 128
6.2 A tally-hiding protocol for Condorcet-Schulze . . . 133
6.2.1 The Schulze method . . . 133
6.2.2 Ballots as lists of integers . . . 134
6.2.3 Obtaining the adjacency matrix from the encrypted ballot . . . 135
6.2.4 Computing the result from the encrypted adjacency matrix . . . 135

6.2.5 Condorcet-Schulze, the bottom-line . . . 135
6.2.6 Comparison with Ordinos . . . 136
6.2.7 Implementation . . . 138
6.2.8 A possible adaptation for the ranked pairs variant . . . 139
6.3 A solution for single transferable vote . . . 140
6.3.1 Existing solutions for STV in electronic voting . . . 141
6.3.2 Choosing one version of STV . . . 142
6.3.3 Ballots as lists of candidates . . . 143
6.3.4 A tally-hiding protocol for academic STV . . . 143
6.3.5 Complexity analysis . . . 146
6.4 Majority Judgment . . . 148
6.4.1 Existing approaches for computing the Majority Judgment . . . 149
6.4.2 A new algorithm for cleartext Majority Judgment . . . 150
6.4.3 Adaptation to the Paillier setting . . . 150
6.4.4 An adaptation to the ElGamal setting . . . 156
6.4.5 Comparison with [CPST18] . . . 159
6.5 Single choice voting . . . 159
6.5.1 Basic single choice voting . . . 161
6.5.2 List voting: computing the D'Hondt method in MPC . . . 161
6.6 Security of the toolbox in the context of electronic voting . . . 163
6.6.1 Universal verifiability . . . 165
6.6.2 Privacy . . . 165
6.7 Lessons learned . . . 169

Part III Coercion resistance **170**

Chapter 7 Is the JCJ voting system really coercion-resistant? **171**
7.1 The JCJ family . . . 171
7.1.1 Presentation of the JCJ protocol . . . 171
7.1.2 Some variants of the JCJ voting system . . . 172
7.2 Unveiling a shortcoming in JCJ . . . 173
7.2.1 Leakage in case of revoting . . . 173
7.3 The impact on coercion-resistance . . . 174
7.3.1 Quantifying coercion-resistance . . . 174
7.3.2 The technical incident scenario . . . 175
7.3.3 A discredit in the press . . . 176
7.4 Defining coercion-resistance . . . 176

7.4.1 The original definition of JCJ . . . 178
7.4.2 Our definition of coercion-resistance . . . 181
7.5 A description of the leakage in JCJ . . . 183
7.5.1 Generalization . . . 186
7.6 Discussion . . . 187

Chapter 8 CHide: a cleansing-hiding variant of JCJ 189
8.1 Description of the protocol . . . 189
8.1.1 Efficiency considerations . . . 191
8.2 Security proofs for CHide . . . 191
8.2.1 Proof of coercion-resistance . . . 192
8.2.2 Proof of verifiability . . . 196
8.3 Conclusion . . . 202

Chapter 9 Traceable encryption for verifiable receipt-free electronic voting 203
9.1 Our definition of receipt-freeness . . . 205
9.1.1 Existing definitions . . . 205
9.1.2 Modeling vote buying . . . 206
9.2 Introduction to traceable encryptions . . . 209
9.2.1 Definition . . . 209
9.2.2 Security notions for verifiable receipt-free voting . . . 210
9.3 Building blocks . . . 211
9.3.1 Bilinear pairings . . . 211
9.3.2 Linearly Homomorphic Structure-Preserving Signatures . . . 212
9.3.3 The Groth-Sahai proof system . . . 214
9.4 Construction of a traceable encryption scheme . . . 216
9.5 Security proofs for our traceable encryption scheme . . . 218
9.5.1 Verifiability . . . 218
9.5.2 Traceability . . . 221
9.5.3 TCCA security . . . 224
9.6 Application to verifiable receipt-free electronic voting . . . 228
9.6.1 A voting scheme based on a traceable encryption . . . 228
9.6.2 Implementation . . . 230
9.6.3 Receipt-freeness . . . 231
9.7 Adapting the scheme to provide cast-as-intended verification . . . 233
9.7.1 Adapting our scheme for the Benaloh challenge . . . 234
9.7.2 On the fly cast-as-intended verification . . . 236

9.8 Conclusion 237

Conclusion **238**

Appendices **241**

Appendix A ZK-TCPA security of the ElGamal threshold encryption scheme **241**

Appendix B The hybrid argument **244**

Appendix C Proof of correctness for the Majority Judgment algorithm **247**

Appendix D Computing the coercion levels **254**

D.1 The coercion level in the ideal game 254

D.2 Modeling the real game 256

D.3 Quantifying the coercion level in some specific cases 257

D.4 The impact of the parameters 259

Appendix E Proof of privacy for CHide **262**

Bibliography **265**

Résumé **281**

Introduction

Voting is the basis of all democracies: it is the keystone that legitimates the actions of a government, and is described by Lyndon B. Johnson, the 58th president of the USA, as the "most powerful instrument ever devised by man for breaking down injustice". Recently, electronic voting arose as a way to improve the existing voting systems. First, electronic voting may allow people to vote through the Internet, which is more accessible than a designated polling station determined by the electoral list. This could represent an interesting alternative to postal or proxy voting, especially for the expatriates, the disabled and the students. Second, electronic voting may be a necessary alternative in case of a long-term lock-down, as seen during the COVID pandemic. Finally, the use of computers can facilitate the act of counting votes and computing the result, and help to limit the usage of paper ballots, which have a negative impact on the environment. Because of these advantages, Internet voting has been used for politically-binding elections in several countries, such as Australia, Canada, France, Norway and Switzerland. Other forms of electronic voting, based on voting machines, were used, for instance, in Bangladesh, Brazil, Namibia, New Zealand, Pakistan, South Korea and the USA. The most notable example for the deployment of Internet voting is Estonia, where voters can vote through the Internet since 2005, and where the proportion of the Internet voters grew from a small percentage to 51% in 2023 [ESWV22, Vot23]. On the other hand, some countries, such as Germany, Italy and the UK, have banned voting machines, considering that they are susceptible to fraud or that a voter must fully understand all the steps that their ballots go through, even if they do not have any technical background.

One of the main reasons to be reluctant about electronic voting is the risk of a hack: by exploiting a vulnerability, an attacker may be able to break down the service, recover the ballots chosen by the voters or rig the result of the election, which would jeopardize the sovereignty of the country and question the legitimacy of the elected representatives. For this reason, it is extremely important that the voting system guarantee privacy and verifiability. Intuitively, *privacy* is achieved if no one can learn the choice of a given voter. As for *verifiability*, it is often decomposed into several properties, namely individual verifiability, universal verifiability and eligibility. Informally, eligibility states that only eligible voters may be able to have their ballot counted, and that at most one ballot is counted per voter. Individual verifiability states that a voter is able to verify that the cast ballot is indeed added to the ballot box, and that it contains the chosen voting option (or candidate's name). Finally, universal verifiability states that anyone can verify that the result of the election is consistent with the ballot box. To achieve those properties, it is usual to make several *trust assumptions*. For instance, in paper voting, it is assumed that the ballot box is secure, so that no one can maliciously remove or add ballots, nor consult the content of a specific ballot. Also, it is generally difficult to enforce several properties simultaneously. For instance, declaring that Kim Jong-un is the winner would perfectly respect privacy, as the voters would not give any information about their choice. By contrast, vote by show of hands could be a possible solution for verifiability, but does not guarantee privacy.

In academic electronic voting, various voting schemes exist to address both privacy and verifiability. For privacy, the main strategy is to *encrypt* the ballots. This way, if the server is compromised, the attacker – also referred to as the *adversary* – can only recover encrypted data, which do not tell which voter chose which candidate (or, more generally, which voting option). For universal verifiability, the main cryptographic primitive is the *zero knowledge proof* (ZKP), which allows the talliers to prove that the tally protocol, that is used to compute the result from the encrypted ballots, was correctly executed by the talliers, who hold the shares of the secret decryption key. A ZKP, as the name implies, does not reveal anything about the secrets used to produce the proof; this way, no information (other than the result of the tally protocol) is leaked about the content of the ballot box, and privacy is preserved.

Encryptions and ZKP are extremely common in electronic voting; however, schemes such as sElect [KMST16] do not rely on ZKP for verifiability, and schemes based on self-tallying ballot boxes (*e.g.*, [KY02]) do not rely on encryption for privacy. Some examples of academic voting systems that combine encryptions and ZKP are Adder [KKW06], Helios [Adi08, dMPQ09] and Belenios [CGG19], which are very similar in their design. However, there are numerous other academic proposals; for instance, schemes such as D-Demos [CZZ+16] and BeleniosVS [CFL19] can ensure privacy, even against an adversarial voting device; and schemes such as [CPP13] provide *everlasting privacy* (see [HMMP23] for a survey on the subjet). Other schemes, such as Prêt-à-Voter [Rya05, RS06], ThreeBallot [Smi07] and Scantegrity [CEC+08], focus on paper-based electronic voting. When paper ballots are used, the result can be counted from the physical ballots or from the electronic ballots; examples of schemes that count the electronic ballots are STAR-Vote [BBE+13] and Bingo Voting [BMR07]. The interest of having both electronic and physical paper ballots is that a technique known as *risk limiting audit* [LS12] can be used to reduce the risk of an error or a manipulation when counting the ballots.

Although privacy and universal verifiability are two fundamental security requirements in electronic voting, they are not considered sufficient for high stake elections. A first difficulty is related to the individual verifiability: since the ballots published in the ballot box are encrypted, it is difficult for the voters to gain confidence as to whether their ballot actually contains the desired voting option. This can be a problem, for instance, if the voting device is compromised by a malware. To address this possibility, several strategies exist to provide *cast-as-intended* verification. A popular approach is based on return codes: each voting option is associated to a secret code, which is unique for each voter. At some point after the voting phase, the voter receives a return code which is computed from the encrypted ballot, and compares it to the expected one. If the return code corresponds to the chosen voting option, then the voter is convinced that the encrypted ballot was produced and cast correctly. This is the approach, for instance, of [HRT10] and CHVote [HKLD17]. Return codes were notably used in Norway [Gjø11, PG12] and in Switzerland [GGP15], where Internet voting was allowed from 2003 to 2019 and will be resumed in June 2023.

Apart from return codes, another popular strategy is the Benaloh challenge [Ben06], which is used in Helios. A recent alternative to the Benaloh challenge is proposed in Themis [BCC+22]. In Estonia, however, cast-as-intended verification is made thanks to a third party verification device, that the voter can query with a cryptographic receipt to check whether the correct voting option was encrypted within the ballot [HW14]. Other approaches exist to provide cast-as-intended verification; for instance, Selene [RRI16] is based on tracking data. For a comparison and a categorization of several academic proposals, see [MZR+21].

Receipt-freeness and coercion-resistance

Besides the cast-as-intended verification, a second difficulty, related to privacy, is the risk of *vote-buying*. Indeed, in a classical voting system such as Helios, the voter produces an encrypted ballot that contains their choice. Yet, by using an ad-hoc voting algorithm, the voter can produce a ballot for which they know the randomness used for the encryption. This randomness can be used as a *receipt* to convince a vote buyer that the ballot encrypts a specific choice. To capture this threat, the notion of *receipt-freeness* was proposed [BT94].

There are various notions of receipt-freeness in the literature but, intuitively, a voting system is receipt-free if the voter cannot convince a third party that they voted in a specific way, even if they are willing to give away their privacy or if they follow a specific instruction given by the third party. To achieve receipt-freeness while preserving verifiability, there are two main approaches in the literature. First, the *deniable revoting* paradigm consists of allowing the voters to revote. When a voter revotes, the previous vote is canceled, but an external observer is unable to tell whether a given ballot has been canceled or not. This way, even if the voter proves that they voted in a specific way, the vote buyer would have no guarantee that the said vote was not cancelled by a subsequent revote. Examples of academic proposals based on the revoting paradigm are, for instance, [LHK16] and VoteAgain [LQT20].

Another approach is based on the *rerandomization* paradigm, where the voters cannot directly submit their ballot to the public ballot box. Instead, the ballot is privately sent to a *rerandomization server*, which is trusted for the purpose of receipt-freeness. The server rerandomizes the ballot, so that it becomes indistinguishable from a random ballot. This way, even if the ballot was created maliciously by the voter (or was given by the vote buyer), it is no longer possible to prove that the rerandomized ballot contains a specific voting option. Nevertheless, individual verifiability is still achieved, which means that the voter has a guarantee that the content of the ballot has not been modified. To achieve this without letting the voter use the guarantee as a receipt, [Hir10] proposes to use designated verifier zero knowledge proofs (DVZKP) [JJ00], which can only convince the voter. However, using DVZKP requires a setup which raises some practical issues. Later on, the need for DVZKP was dropped thanks to bilinear pairings, which introduces the possibility to rerandomize ciphertexts and signatures altogether [BFPV11]. The idea was further developed in BeleniosRF [CCFG16], which also provides a modern definition of receipt-freeness.

A threat related to vote buying is that of *coercion*, where an attacker, the coercer, asks a voter to vote in a specific way, using a threat or a reward. Compared to receipt-freeness, *coercion-resistance* assumes a stronger adversary, which can be active during the voting phase and asks the voter to give away their voting material so as to cast the ballot instead of the voter. When an electronic voting solution is used with no counter-measure against coercion, the coercer can coerce a larger number of voters, or gain confidence – thanks to the verifiability mechanism – as to whether the coerced voters obeyed or not. In Estonia, the main way to mitigate coercion is to allow the voters to revote, so that they can first comply with the coercer and then revote for the desired voting option, when given a moment of privacy. Although revoting is an intuitive counter-measure against coercion, this assumes that the voter is able to revote *after* the coercer, which is not necessarily justified as the coercer can wait for the last moment to cast their ballot. In addition, if the coercer asks for the voter's credentials, then the voter cannot know when the coercer is going to use them.

The main academic solution to address coercion is the JCJ protocol, proposed in [JCJ05], which also formalizes the notion of coercion-resistance. The idea is that a voter is able to give a fake, invalid voting credential to the coercer. The latter can cast a ballot with the given credential,

- Privacy: no one can learn how a given voter voted
- Individual verifiability: the voter is guaranteed that the cast ballot is added to the ballot box, and contains the desired voting option
- Universal verifiability: everyone can verify that the result of the tally is correct with respect to the ballot box
- Receipt-freeness: a malicious voter is unable to prove that they voted in a specific way
- Coercion-resistance: no one can force the voter to vote in a specific way

Figure 1: Some desirable security properties in electronic voting

and the ballot will be added to the ballot box independently of the validity of the credential. However, ballots that use an invalid credential, and ballots that use duplicated credentials, are removed after the voting phase. Hence, the main security property of the JCJ protocol is that the coercer is unable to distinguish a real credential from a fake one, or to tell whether a given ballot has been removed or not. In the literature, many subsequent schemes were based on the fake credential paradigm, and can be considered as iterations over the JCJ protocol. The most notorious example is Civitas [CCM08], which proposed an explicit registration phase. Other contributions, for instance, were focused on improving credential handling [CH11, NV12]. Finally, many proposals aimed at improving the scalability of JCJ: see, for instance, the schemes by Spycher *et al.* [SKHS11, SHKS11, SKHS12] and the schemes by Araújo *et al.* [AFT08, ARR+10, AT13, ABBT16].

Preventing Italian attacks

One major threat which is not addressed in coercion-resistance nor receipt-freeness is that of *Italian attacks*, which are based on the information available from the result of the tally. Indeed, one of the main strategies to compute the result of the election from the encrypted ballots is to rely on a *mixnet* [Cha81], which reveals the list of all the voting options chosen by the voters, but in a random order. In general, this gives more information that just the result of the election (*i.e.* the name(s) of the winner(s), the number of counted ballots and the number of ballots cast), and this information can be used by a coercer to decide if a coerced voter obeyed or not. For instance, in preferential voting, a choice can be any permutation of the candidates, so that there may be much more possible choices than there are voters: in Australia, the 2019 New South Wale legislative election, which used a preferential voting system known as single transferable vote (STV), featured several hundreds of candidates [NSW19]. In such a situation, it is possible for the coercer to instruct the voter to first rank the coercer's preferred candidate, then to use a very specific and unlikely permutation of the candidates, for instance which alternates several opposite parties. If the voter does not obey, there will most likely be no other voter who would cast such a ballot, so that the coercer can deduce, by observing the result of the tally where all the chosen permutations are revealed, whether the voter obeyed or not. This way, the coercer can coerce a large number of voters, and know exactly which of them obeyed or disobeyed.

There are not so many counter-measures against Italian attacks in the literature; the most promising approach is that of *tally-hiding*. In a *partially* tally-hiding scheme, the tally protocol

still leaks some side-information, but not necessarily all the voting options chosen by the voters. In [RCPT19], an MPC protocol for instant run-off voting (IRV, a specific case of STV where there is a single winner) is proposed. This protocol allows to compute an IRV tally without revealing all the permutations chosen by the voters; however it reveals some information about the progress of the protocol. For the Condorcet methods, which are several counting methods for preferential voting which respect a criterion introduced by Condorcet [Con85], the main strategy is to represent the choice of a voter as a matrix, so that the ballots can be added together. This is, for instance, the approach proposed in [HPT19]. Another example that uses a form of partial tally-hiding is the Shuffle-sum scheme [BMN+09], which aims at mitigating the risk of an Italian attack in STV by concealing the most crucial information. However, the attacker may still exploit the other side-information that are available in the tally. As explained in [BMN+09, Section III. B.], it is possible that this information may still be too much: in particular, they propose a few realistic scenarios where the leakage of their own scheme could be exploited to detect the absence of a specific permutation, and hence to perform an Italian attack. To address this, they propose an alternative solution where the information exploited by their attacks are only available to the talliers, and where less information are leaked to the public. Nevertheless, this imposes an additional trust assumption, where the talliers are not supposed to collude with the coercer. In addition, the resulting scheme is only partially tally-hiding, which means that a full analysis of whether the information leaked during the tally can or cannot be exploited by the adversary is necessary.

In Kryvos [HKK+22], a solution based on *public* tally-hiding is proposed. The idea is that the public only has access to the result of the election, while the talliers learn more information. The main problem with this approach is that it does not protect the voters against a coercion from a tallier.

Alternatively, it is possible to design a *fully* tally-hiding protocol. This was done, for instance, in [CPST18]; however, this solution was proposed for the Majority Judgment, for which it is possible to use a homomorphic tally, so that the risk of an Italian attack is low. Concurrently to this thesis, the independent work of Ordinos [KLM+20] was proposed to achieve full tally-hiding. Ordinos was extended in [HHK+21] to cover various counting functions, including some variants of the Condorcet method. The solution proposed in Ordinos is expensive on the voter side, and does not allow the voters to rank several candidates at equality, which is restrictive in the context of Condorcet voting. In both proposals, the solution proposed relied in multi-parti computation (MPC) based on the Paillier encryption scheme, which, compared to the more popular ElGamal encryption scheme, has the property of being *additively* homomorphic. However, this extra property comes with several drawbacks: first, in the Paillier setting, the key length is much longer, so that computing an encryption is more expensive than in the ElGamal setting. Compared to a Helios-like solution, the cost of encrypting a ballot on the voter side can be several order of magnitude more expensive in the Paillier setting, which raises some practical issues. In addition, as the Paillier encryption scheme is not as widely used as the ElGamal encryption scheme, there are no well-studied and widely deployed library available, which is all the more detrimental in electronic voting as we need two libraries: one on the server side and one on the voter side.

Provable security

The use of well-studied cryptographic primitives – such as encryption and ZKP – is not enough to guarantee the security of a protocol. Ideally, the latter should be analyzed at several levels of abstraction. First, the designers of the protocol should provide a cryptographic (or formal)

security proof; second, the protocol specification should undergo an audit to make sure that there is no vulnerability; finally, the implementation should also be audited, for instance through a public bug bounty. For these proofs and audits to be meaningful, the academic community recommends that the specifications of the protocol should be public, so that the whole community can analyze it. In Switzerland, the public audit of the Swiss Post voting system, whose specifications are available at [Swi], allowed the detection of some vulnerabilities before release, as described in [HLPT20] and [CDG22]. By contrast, it was revealed that the voting system used in Australia had some security issues [HT15], as well as the one used in France [DH22]. This could have been prevented if the academics from the electronic voting community had the opportunity to audit those systems *before* their deployment. Those failures show that it is not easy to design a secure electronic voting system, let alone assess its security. For this reason, it is usual to provide a computational or a formal proof that the desired properties are verified.

In a computational proof, the adversary is modeled as a *Turing machine*, which has a limited (polynomial) computational power but can perform arbitrary computations. The main strategy is to exhibit a polynomial reduction to a known computational problem, such as integer factorization or the discrete logarithm problem. In other words, a cryptographic proof is a mathematic proof that if a security property is breached, then there exists an explicit polynomial-time Turing machine (*i.e.* an efficient algorithm) that solves a computational problem which is considered hard. For a well-studied problem such as the discrete logarithm, this means that the security property is verified.

In a formal proof, a mathematic, symbolic model is designed to represent the protocol and the desired security property. In such a model, the cryptographic primitives used in the protocol, as well as the possible actions that the adversary can perform are idealized, for instance using rewriting rules or equational theories. Once the model has been created, the proof itself consists of an evidence that the security property can or cannot be breached. Typically, a formal proof is obtained thanks to a fully automatic or interactive tool based on deduction techniques. Compared to a computational proof which relies on a well-studied computational assumption, a formal proof assumes that the cryptography is perfect and cannot be breached. However, a formal proof usually covers more attack scenarios.

Before providing a proof, one key step is to model the desired security properties and to give a formal definition for them. Yet, the definitions of the security notions in electronic voting are not stabilized. For instance, one of the first definitions of privacy was given by Benaloh [Ben87], and has been used or extended in various future works (*e.g.*, [KZZ15, CL18]). However, this definition comes with several limitations, so that other definitions were proposed. In particular, the notion of *ballot privacy* [BCP+11, BPW12] converged to the BPRIV definition, given in [BCG+15b]. This definition was extended in [CLW20], to model the presence of a malicious ballot box.

For receipt-freeness, two modern definitions can be found in [CCFG16] and [KZZ15]. Compared to the definition of Kiayias *et al.*, the definition of Chaidos *et al.* does not take into account the individual verifiability mechanism. However, the definition of Kiayias *et al.* does not consider a malicious voter, which is restrictive. In [DPP22b], a modified version of the definition of Chaidos *et al.* has been proposed, where the registration protocol is no longer relevant. This was done in an attempt to achieve receipt-freeness (almost) independently of the remaining of the protocol, which allows a more modular security analysis and makes the voting scheme easier to adapt.

For coercion-resistance, the main academic definition is that of [JCJ05], which is still a reference in the research about coercion-resistance. Intuitively, this definition compares a real game with an ideal game: in the real game, the adversary observes the real JCJ protocol; in the

ideal game, the adversary is only given the "result"; in both games, the goal of the adversary is to guess whether the coerced voter obeyed or evaded coercion. This comparison is made because the adversary, by observing the result, can gain some information about the choice of the coerced voter: this is the idea behind an Italian attack. In [HS19], it is remarked that the JCJ definition is flawed and cannot be achieved by a voting scheme which has a public ballot box. The main reason why is that the ideal game does not yield any information about the ballot box. Consequently, by observing the size of the ballot box in the real game, and by comparing it to the result of the election (which indicates the number of counted ballots) the adversary gains some information about whether the ballot cast with the coerced voter's credential was counted or not. To fix this flaw, [HS19] proposes to modify the ideal game, and to add the information about the size of the ballot box. However, the fixed definition is still incomplete, as it does not consider revotes.

Indeed, in the context of coercion-resistance, it is natural to allow revoting, since it can constitute a first counter-measure against an implicit influence from a family member or an employer. Suppose, for instance, that a granddaughter explains to her grandfather how to vote online. For this purpose, she asks him to identify himself to the voting platform and to proceed step by step, while she remains over his shoulder to clarify each step. In this scenario, the grandfather may feel compelled to choose the democratic party while he would have preferred to choose the republican one. When revoting is allowed, the grandfather is able to select just any candidate (or even the candidate "suggested" by the granddaughter). Afterwards, when the granddaughter is no longer here, he can revote with his personal choice. Another example is when an employee is encouraged to vote *at work*, using a device which may be monitored by the employer. To avoid conflict with the employer, the employee may want to first vote for the conservative party when at work, then revote for the labor party at home. Hence, it is important that a definition of coercion-resistance should take revoting into account.

Our contributions

1. We propose a toolbox for MPC based on the ElGamal encryption scheme, that can be used to achieve full tally-hiding.

Our toolbox, presented in Chapter 5, is based on the conditional gate primitive from [ST04], which can be used in the ElGamal setting. This gives an interesting alternative to the Paillier setting, which is greatly beneficial on the voter-side and achieves similar computational complexities on the server side. This toolbox provides many MPC protocols to realize various usual operations on encrypted data, such as arithmetic operations and comparisons, but also more complex operations such as sorting. To make this possible, we propose several computation/communication trade-offs, which can be deployed to mitigate the greater communication complexity in the ElGamal setting. In Chapter 6, we apply our toolbox to design a tally-hiding protocol for Condorcet-Schulze, STV, Majority Judgment and the D'Hondt method. In the case of the Condorcet methods, we discovered a privacy breach in the solution of [HS19], which occurs when a voter gives the same rank to two candidates. In Section 6.1.2, we propose a new way for the voter to submit a ballot for Condorcet voting, which allows blank voting and is compatible with a homomorphic tally. For the Majority Judgment, we remarked a shortcoming in the solution proposed in [CPST18], which uses a heuristic known as the *majority gauge*. Indeed, the majority gauge is not guaranteed to output a result. Finally, we also discovered a problem with the solution proposed in [KLM+20], which was designed to reveal the names of the s candidates that received the most votes, where s is some parameter (for instance, the number of seats).

Indeed, in case of a tie, it is possible that their solution actually outputs more than s winners. We propose a non-intrusive way to include any tie-break mechanism into the solution proposed by Ordinos; this preserves the full tally-hiding property and does not deteriorate the efficiency.

We prove the security of our toolbox in the security framework of [CCL15], which is a simpler variant of the universally composable framework of [Can01] (for short, we refer to this framework as the SUC framework). To achieve this, we modified the conditional gate protocol and proved the SUC-security of the modified protocol in Section 4.4. As the SUC framework provides a *composition theorem*, the SUC-security of the main primitive can be used to prove the desired security properties, such as privacy and verifiability, which is done in Section 6.6.

2. We unveil a leakage in the JCJ scheme which can compromise its coercion-resistance when revoting is allowed.

When revoting is allowed, we discovered that the extra information revealed during the tally phase of the JCJ protocol can be exploited by the coercer to infer the behavior of the coerced voter, using Bayesian probabilities. More precisely, we identified the exact nature of the leakage in JCJ: Compared to the pure result of the election which would contain some information about the total number of ballots removed, the JCJ scheme leaks the number of ballots that have k duplicates for all $k \geq 1$, as well as the number of ballots that use an invalid credential. To assess the impact of this leakage, we used the formal framework of [KTV10a] which gives a quantitative definition of coercion-resistance. In this framework, it is possible to compare the *coercion level* of the real protocol to that of the ideal protocol, which would not suffer from any leakage. Using this framework, we propose several realistic scenarios where the difference between the ideal and the real coercion levels is not negligible.

One of the reasons why the flaw of the JCJ protocol was not noticed so far may be because the JCJ definition of coercion-resistance does not allow revoting. In addition, it is known that, in a JCJ-like scheme, an unpredictable number of ballots should be removed during the tally phase: otherwise, the coercer would notice when the ballot cast with the voter's credential is removed. For this reason, it is necessary to model the presence of ballots that use an invalid credential, but that are not a ballot from the coercer: they are referred to as *dummy* ballots. In the JCJ definition, those ballots are supposed to come from the honest voters, who have to sacrifice their own vote for this. This is not realistic and does not allow modeling a situation where additional dummies are cast by an external party, which is not an eligible voter. For these reasons, we designed a new definition of coercion-resistance, which better captures the presence of dummy ballots as well as the possibility to revote.

As the JCJ protocol does not verify our definition of coercion-resistance, we propose CHide, a variant of the JCJ protocol that uses the tally-hiding toolbox in order to prevent the leakage of the JCJ scheme. This is done in Chapter 8, and shows that our definition of coercion-resistance can be achieved by a practical protocol. To make the protocol practical for realistic parameters, we designed a new cleansing phase that relies on sorting, and is more scalable than the quadratic cleansing phase of JCJ. We prove that privacy, verifiability and coercion-resistance are achieved that CHide under the same trust assumptions as JCJ.

3. We investigate the notion of receipt-freeness, and propose a solution which can be a practical first step towards coercion-resistance.

In collaboration with Henri Devillez, Olivier Pereira and Thomas Peters, we propose a new definition of receipt-freeness, which does not make any assumption about the registration phase

or the eligibility mechanism. Compared to the definition of [KZZ15], our definition takes into account the fact that the voter may use *any* algorithm to produce their ballot, including an algorithm which may be provided by the vote buyer. Compared to the definition of [CCFG16], our definition allows the adversary to give *any* instruction to the voter, and not just a given ciphertext. In addition, it also captures the fact that the voter may be given a receipt during the voting phase, due to the individual verifiability mechanism. Overall, our definition of receipt-freeness is closer to the intuition of vote-buying, and achieving this definition can be a first step towards coercion-resistance. In addition, based on the previous work of [DPP22b], we propose a modular strategy that allows to build a receipt-free protocol, by providing an easy to verify set of conditions about the encryption scheme, the tally protocol and the voting phase. This makes achieving receipt-freeness more modular, and more independent of the specificities of the protocol. We also provide a new encryption scheme that satisfies the properties required by our strategy, so that it can be instantiated. Compared to the scheme proposed in [DPP22b], this new scheme supports 0/1 proofs (*i.e.* it is possible to prove that the ballot encrypts a message of a specific form), which is extremely interesting in the context of electronic voting. In addition, the setup protocol is public coin, which means that we need fewer trust assumptions. Compared to the scheme proposed in [CCFG16], our encryption scheme is not limited to encrypting small bitstrings.

Structure of the thesis

This thesis is divided into three parts. Part I gives some preliminaries about electronic voting, the security definitions and the security proofs. In Part II, Chapter 4 gives some background in MPC, and the first contribution of this thesis can be found in Section 4.4, where we prove the SUC-security of the modified conditional gate protocol. Afterwards, we present the various protocols of our MPC toolbox in Chapter 5, and explain how to apply them to compute the tally for various counting methods in Chapter 6. Finally, Part III contains our finding about coercion-resistance: Chapter 7 unveils the vulnerability of the JCJ scheme and proposes a new definition of coercion-resistance, Chapter 8 introduces CHide and the corresponding security proofs, and Chapter 9 contains our contributions about receipt freeness.

Related publications

- We presented a short paper at E-Vote-ID 2020 [CGY20], where we revealed a vulnerability of the Belenios protocol. This contribution is briefly described in Section 3.2.2.
- Our toolbox for generic tally-hiding in the ElGamal setting was presented at Esorics 2022 [CGY22a]. The full version of this paper, which includes details about the MPC protocols that we designed for Condorcet-Schulze, STV, Majority Judgment and D'Hondt, as well as the corresponding security proofs, can be found in [CGY21].
- Our finding about the weakness of the JCJ protocol was published on eprint [CGY22b], where we propose our new definition of coercion-resistance as well as the CHide protocol.

Part I

Preliminaries

Chapter 1

Security in electronic voting

This chapter introduces some vocabulary around electronic voting and defines the main security properties. It provides notations and definitions that are useful in the remaining of this thesis.

1.1 The fundamental notions of electronic voting

We first present the most common phases of electronic voting, as well as their participants.

1.1.1 The generic structure of electronic voting protocols

An electronic voting protocol involves many participants, each of whom has a specific role. A participant may be trusted for a given purpose, but not for another: this defines the *trust assumptions* of the protocol. Finally, the participants interact during different phases, which describe the overall structure of the protocol. Depending on the protocol, different participants, trust assumptions and phases may be used.

The main participants in electronic voting. We first present the most commonly considered participants, as well as some usual trust assumptions; see Table 1 for a summary. Depending on the protocol, additional participants and stronger assumptions may be used, which can be perceived as a drawback. On the other hand, if less participants or weaker assumptions are necessary for the same security level, this can be perceived as a feature. Most often than not, however, each protocol has its own compromise.

The **election authorities**, *i.e.* the organizers, choose the public parameters of the election: the questions asked, the possible answers, the eligible voters and the opening and closing dates of the election are up to the organizers. In addition, the latter also chooses the counting method and the actual protocol; this can include, for instance, the cryptographic group and the security parameter. In general, the election authorities are trusted for the above purposes, but not for any other. This is because they only intervene at the very beginning of the process, before the protocol even started. Indeed, the proposed cryptographic protocol ideally undergoes a careful public audit where anyone can look at the specifications and the implementation to make sure that there is no vulnerability or backdoor. Similarly, the non-cryptographic information published by the organizer can be publicly verified, so that it is not easy to add factitious voters or rig the election by choosing a specific counting method or formulating the questions in some biased way.

The **public board** displays some public information, such as the current state of the ballot box or some verification transcripts. It plays a critical role in both individual and universal

verifiability. Nevertheless, it is typically trusted as a public, append-only and shared dataset. It means that anyone can consult its contents, that the same view is given to everyone and that no data can be removed once they have been added to the board. To enforce this behavior, a consensus protocol can be used, as done in [CS14, HSB21]. Another solution would be to rely on the blockchain technology, as proposed in [MSH17, HHHH18, CYLR18]

The **server** may have various roles such as storing data, performing checks, providing communication channels, writing on the public board or providing the source code of the protocol (*e.g.*, a JavaScript). It is usually considered malicious when providing communication channels or writing on the board: for instance, it may temper with the communication with the ballot box or try to add illegitimate ballots. However, it is often assumed to provide the correct public information (*e.g.*, public encryption key and JavaScript). This can be enforced by auditors that will detect any misbehavior of the server.

The distinction between the public board and the server is not always clear. This is because the public board may be hosted by the server, in which case the latter can freely add data on it, including ballots. In addition, it is often necessary to use the server as an intermediate to add a ballot in the public board. Separating the server from the public board allows assuming the server malicious while the public board is still considered honest. Observe that even if the board is honest, there is no guarantee about the validity or the authenticity of its data: we only assume that they are the same for everyone and that they are added in an append-only fashion. Hence, if no mechanism prevents it, a malicious server could add ineligible ballots in the board: this is called *ballot stuffing*.

The **voters** send their encrypted ballot to the server. For this purpose, they use a **voting device** that performs the necessary cryptographic operations. The voters are not trusted: some of them, the *corrupted* voters, may be under the complete control of the adversary. However, we consider that the non-corrupted voters are *honest* and follow the protocol rigorously. Interestingly, it is possible to dissociate the voter from the voting device. Indeed, the latter may be compromised by a malware that encrypts another voting option than the chosen one. To address such a situation, a specific strategy called *cast-as-intended* verification can be deployed, that allows the voter to verify the behavior of the device. When separating the voter from the device, the latter is not trusted for individual verifiability but often trusted for privacy. Indeed, in the convenient scenario where the voter types or selects the chosen voting option, the device unavoidably learns the choice of the voter. For a possible counter-measure, see for instance [CFL19]. In this thesis, we do not separate the voter from the voting device, and hence assume that the voting device is honest.

The **registrars** give their voting material (typically, a secret voting credential) to the voters. Intuitively, this credential is necessary to either produce a valid ballot, or to have it accepted by the server. Therefore, the role of the registrars is to guarantee the eligibility. In general, they are trusted as a whole, but some may be corrupted.

The **talliers** compute the result of the election from the encrypted ballots. For this purpose, they need the secret decryption key. Since the latter allows decrypting the ballots individually, hence compromising privacy, it is common that a specific strategy is deployed to *distribute* the trust between the talliers, so that no individual tallier can learn the secret key. In general, a cryptographic technique named *secret sharing* (see Section 2.2.3) is used to enforce privacy, even when up to t talliers are controlled by the adversary, where t is some threshold. In this case, it is assumed that no more than t malicious talliers may collude to break privacy. However, it is common not to trust the talliers at all when it comes to verifiability.

Finally, the **auditors** are responsible for verifying any accessible data. In general, it is assumed that at least one of them is honest, will perform any possible verification and report

Table 1: The typical trust assumptions in electronic voting

Participants	Trust assumption
Voter	Not trusted
Voting Device	Trusted / not trusted for individual verifiability Trusted for privacy
Registrars	Trusted as a group for verifiability and privacy
Talliers	At most t of them can collude to break privacy Not trusted for verifiability
Auditors	At least one of them is honest for auditing
Public board	Trusted as a append-only shared dataset
Server	Untrusted for communication channels Trusted to provide public data
Election authorities	Trusted to provide the voting protocol

Table 2: The most common phases in electronic voting

Phase	Participants	Goal
Setup	Talliers	Generate the public cryptographic materials
Registration	Registrars Voters	Give their voting credential to the voters
Voting	Voters Server	Collect the ballots from the voters
Tally	Talliers	Compute the result from the collected ballots

any anomaly.

The most common phases in electronic voting. Apart from defining the participants and the corresponding trust assumptions, a protocol also defines their behavior and their interactions. In electronic voting, we usually divide the protocol into phases, summarized in Table 2.

The first phase is the **setup phase**, during which the talliers generate all the necessary cryptographic materials, such as the public encryption key. In general, the setup phase can also involve the registrars but, for simplicity, we consider that the latter will only intervene during the registration phase.

The **registration phase** allows the voters to get their credentials from the registrars. For this purpose, they may have to authenticate themselves using a certain method. This is often left unspecified in electronic voting, and could involve providing an official document such as an identity card, or using an online authentication platform.

During the **voting phase**, the voters submit their votes in the form of encrypted ballots. The server checks the eligibility of the voter, the validity of the ballot and adds it to the ballot box. Also, it is generally required that the voter checks that the desired ballot was indeed added to the board, since we usually do not fully trust the server.

Finally, the **tally phase** happens last. During this phase, the talliers compute the result of the election from the encrypted ballots and produce a proof that the result is valid with respect to the public board.

Note that, depending on the protocol, the public board and the server may or may not be active during any of the aforementioned phases. For instance, the public encryption key can be added to the public board during the setup, and the voters may use the server to register

themselves. In general, an electronic voting protocol may use another set of participants and phases: we only presented a generic structure that suits this thesis.

1.1.2 Formalization and notations

To assess the security of a voting protocol, it is important to have a precise mathematical framework so that we can model it, as well as the security properties that we consider.

Counting function. The goal of a voting protocol is to evaluate the result of a counting function on the voting options chosen by the voters. More formally, let $\mathcal{V}$ be the set of all possible voting options and $\mathcal{I}$ be a set which represent the identities (or aliases) of the voters. Let $\mathcal{R}$ be the set of all possible results. For any set A, we denote A^* the free monoid generated by A, that is the set all of finite sequences of elements of A. The natural law of this monoid is the concatenation, denoted $||$. In this thesis, we consider that a counting function is a function $\mathsf{count} : \mathcal{V}^* \longrightarrow \mathcal{R}$, that is invariant over the permutations of its arguments. The invariance property means that the order in which the voters cast their ballot is not important. Consequently, it is assumed that a *revote policy* is applied before the counting function, so that there is a unique voting option for each voter. Indeed, assume that revoting is allowed, so that a voter who has already voted for some option ν_1 can revote for a (potentially) different option ν_2. Then the revoting policy may instruct keeping the first choice (revoting is allowed but does not change the choice), to keep the last choice or even to *combine* the choices in some specific way (for instance, set the final choice as the average choice). In this thesis, we denote $\mathsf{cleanse} : (\mathcal{I} \times \mathcal{V})^* \to \mathcal{V}^*$ the revote policy, which takes care of the duplicated $\mathsf{id} \in \mathcal{I}$.

Remark that our definition for counting function is *identity-oblivious*, which means that once the revoting policy has been applied, the identity of the voters is no longer relevant, so that the final choices of two voters can be swapped without altering the result. This is not always the case in general: for instance, consider a vote between stakeholders where some voters are given more weight than the others. For more generic, *identity-dependent* counting methods, the notion of counting function may not be suitable. However, we only considered identity-oblivious counting functions in this thesis, so that this distinction is only relevant when we define privacy in Section 1.3. On this occasion, we will use the terminology *tally* function, which can be seen as a function $\mathsf{tally} : (\mathcal{I} \times \mathcal{V})^* \to \mathcal{R}$.

Multiset. Because of the invariance property, it is interesting to introduce the notion of multisets. A *multiset* of size n is a n-tuple modulo the permutations. It can be seen as a set where there can be duplicates, or a sequence in which the order is not important. Given any sequence $\boldsymbol{A}$, we denote $\{\!\{\boldsymbol{A}\}\!\}$ the corresponding multiset (*e.g.*, a, b, c becomes $\{\!\{a, b, c\}\!\}$). By abuse of notation, if A is a set, we also denote $\{\!\{A\}\!\}$ the corresponding multiset. To express that two sequences (or sets) $\boldsymbol{A}$ and $\boldsymbol{B}$ are equal as multisets, we use the notation $\boldsymbol{A} \equiv \boldsymbol{B}$. When A and B are multisets, we can also use the notation $\mathrm{A} \equiv \mathrm{B}$ instead of $\mathrm{A} = \mathrm{B}$, to insist on the fact that A and B are multisets. The union of two multisets A and B is usually denoted $\mathrm{A} \uplus \mathrm{B}$. This gives a notion of inclusion. Finally, we say that a multiset A (resp. a sequence $\boldsymbol{A}$) contains n elements of the multiset B if there exists a multiset C of size n which is included in both A (resp. $\{\!\{\boldsymbol{A}\}\!\}$) and B. Since a counting function is invariant over permutations, we can define it on the finite multisets instead of the finite sequences. By abuse of notation, we often use this representation.

Partial tally. For a counting function, an interesting property is the partial tally. We say that $\mathsf{count} : \mathcal{V} \to \mathcal{R}$ has the *partial tally* property if there exists an associative composition law $\bigstar$ in $\mathcal{R}$ such that, for all $\boldsymbol{V}_1, \boldsymbol{V}_2 \in \mathcal{V}^*$, $\mathsf{count}(\boldsymbol{V}_1||\boldsymbol{V}_2) = \mathsf{count}(\boldsymbol{V}_1) \bigstar \mathsf{count}(\boldsymbol{V}_2)$. A key example is the counting function which returns the number of times each voting option was chosen.

Alternatively, the counting function which returns the multiset of the voting options chosen also has this property. However, not every function has this property. For instance, suppose that there are two candidates, A and B, so that the voting options are $\mathcal{V} = \{A, B\}$. In this setting, consider the majority counting function, with the possible results $\mathcal{R} = \{A, B\}$, where A wins in case of an equality. Then this function does not have the partial tally property: since $\mathsf{count}(A, A||B) = \mathsf{count}(A, A)$, the associativity required for $\bigstar$ would mean that $\mathsf{count}(A, A||B^n) = \mathsf{count}(A, A) = A$ for all n, which is not the case.

Adversary. In this thesis, we model the adversary $\mathbb{A}$ as a probabilistic polynomial Turing machine (PPT), which means that it can perform any computation, has access to a source of randomness but is computationally bounded: there exists a polynomial P such that, when $\mathbb{A}$ is called with the security parameter λ, it terminates after at most $P(\lambda)$ transitions. By contrast, a *non-uniform* adversary is a PPT which also has a different meaningful auxiliary input $z(\lambda)$ for each value of the security parameter; this can model non-polynomial time precomputation.

Voting system. In this thesis, a *voting system* for the counting function count is a tuple of algorithms and protocols $(\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{Check}, \mathrm{Valid}, \mathsf{Tally}, \mathrm{Verify})$. We consider that they have the following signatures:

Setup(λ, n_T, t) is an interactive protocol between the talliers. It takes as input a security parameter λ, a number of participants n_T and a threshold t. It outputs a public key pk; a secret key sk; some pairs $(h_i, s_i)_{i=1}^{n_T}$ where s_i is the secret share of the ith participant while h_i is a public commitment on s_i; and a transcript Π. In general, the setup can have many other forms, so that this description is specific to our setting. It assumes that the key pair $(\mathsf{pk}, \mathsf{sk})$ is generated using a distributed generation key (DKG) protocol (see Section 2.2.3), which is not always the case.

Register(pk, n) is an interactive protocol between n voters and the registrars. We use the notation $(c_i, \pi_i)_{i=1}^n, \Pi \longleftarrow \mathsf{Register}(\mathsf{pk}, n)$ to express that the registration produced the credential c_i for each voter, numbered from 1 to n. In addition to their credential, the voters may also receive an individual (potentially empty) transcript π_i – for instance to convince the voter that the credential they received is actually valid. Similarly, the protocol outputs a (possibly empty) public transcript Π; for instance a public commitment of each credential, or a proof that the protocol was executed correctly.

Vote is an interactive protocol between the voter, the server and the public board. During this protocol, the voter uses an algorithm $\mathrm{Vote}_{\mathsf{pk}}(\nu, c)$ that produces a ballot for the voting option ν, with the credential c. Afterward, the voter can initiate the protocol **Check(ν, c, B, PB)** to verify that their ballot B indeed contains the chosen voting option ν and was actually added to the ballot box in PB. This protocol may involve several other participants, such as the voting device, an external auditor and the voting server. Note that we also included the credential c in the inputs required for the voter; however, some other intermediate values obtained during the voting protocol may also be needed. In addition, given an encrypted ballot B and the current public board PB, the algorithm **Valid(B, PB)** returns 0 or 1, depending on the validity of the ballot with respect to the public board.

Tally$(\mathsf{PB}, \{s_i\})$ is a protocol during which the talliers, who each have a secret s_i, collectively compute the result of the election from the encrypted ballots. It outputs a result $r \in \mathcal{R}$, along with the verification transcript Π. The output of this protocol can be verified thanks to the algorithm $\mathrm{Verify}(\mathsf{PB}, \Pi, r)$, which returns either 0 or 1.

Technically, a voting system is just one part of the specification of a voting protocol. The latter must also articulate all the phases together (which includes the establishment of communication channels) and give instructions about how to behave if a step fails, which is why we used a different terminology. In this thesis, we never describe a voting protocol as a whole and

we restrict ourselves at the system level. Therefore, we do not give a formalization for the other layers of a voting protocol.

Small quantities. In cryptography, it is usual to define a negligible quantity as follows. Suppose that a quantity $\mu(\lambda)$ is a function of the security parameter. We say that μ is *negligible* (in λ) if, for all integer k, we have $|\mu(\lambda)| \leq 1/\lambda^k$ when λ is large enough. A quantity which is not negligible is said *non-negligible*. For probabilities, we also have the notion of large and overwhelming probabilities: a probability $p(\lambda)$ is *large* if there exists a real $\varepsilon > 0$ such that $\varepsilon < p(\lambda)$ when λ is large enough. Finally, p is *overwhelming* is $1 - p$ is negligible. Note that these notions only make sense when considering asymptotically large security parameters, so that $1/2^{128}$ is technically a large probability. To better quantify large probabilities, it is natural to look for an upper bound. For $\varepsilon > 0$, we say that a quantity $p(\lambda)$ is ε-*bounded* if there exists a negligible function μ such that $|p(\lambda)| \leq \varepsilon + \mu(\lambda)$ when λ is large enough.

Game-based definitions. Once the voting protocol is modeled, one must also give a formal definition of the desired security properties. A common approach is to use a *game-based* definition. Such a definition models the protocol inside a game (or experiment), where an adversary tries to reach a specific goal. The adversary may impersonate some participants, who are therefore *corrupted*. However, the game describes the order of activation of the participants, in which channels they may communicate and to which information (*i.e.* inputs) they have access. It also describes the expected format of the output for each activation of the adversary and how they are processed. At the end of the game, an output is produced (typically 0 or 1) depending on whether the adversary reached its goal or not. When the output is 1, we say that the adversary wins the game. Finally, it is sometimes preferable to consider the *advantage* of the adversary rather than its probability to win. In this thesis, the advantage of the adversary in a game is the absolute value of the difference between the adversary's probability to win and 1/2. In some other works, it is common to define the advantage as twice this value; however, as we usually demand that the advantage should be negligible, both definitions are equivalent and we may use any of them interchangeably.

Oracle queries. In the description of a game, it is common to encounter oracle queries. Intuitively, they capture the interactive nature of a protocol and represent the other participants. Typically, if the adversary wants to wait for a voter to send a ballot, this can be modeled by a query to an oracle $\mathcal{O}_{\mathsf{vote}}$. An oracle is an algorithm whose inputs are chosen by the adversary and which returns its output to the adversary. It is not required that the oracles are computationally bounded. When the adversary can make queries to the oracles $\mathcal{O}_1, \cdots, \mathcal{O}_{\mathsf{n}}$, we use the notation $\mathbb{A}^{\mathcal{O}_1,\cdots,\mathcal{O}_{\mathsf{n}}}$. It means that the adversary can make any (polynomial) number of queries to the given oracles, in any order. However, it can only have a black-box access to the oracles, and has no mean to learn anything else than the output. The oracles may have access to all inner states of the game, including the value of the secret key or the random coin. In addition, they can also modify those states, for instance by adding something in the public board. Consequently, the oracles are considered stateful: they have their own inner state and remember any previous query made by the adversary, as well as how it was answered.

1.1.3 The Helios voting protocol

A canonical example is Helios, an online open-source electronic voting protocol introduced in [Adi08]. As the original version had some limitations, an iteration, known as Helios 2.0, was presented in [dMPQ09], on the occasion of the election of the president of the university of Louvain-la-Neuve. The resulting voting platform, accessible at [Hel], is used by the IACR. The

key idea is to encode the voting options as a vector of bits and to use the exponential ElGamal encryption scheme, presented in Section 2.2.2, to encrypt it. This way, a homomorphic tally can be used, where the product of all the ciphertexts is decrypted to reveal the result, while no individual ballot is decrypted, hence preserving privacy. The whole protocol can be summed up as follows.

Registration phase. Every voter receives a mail which contains a link to a secure registration website, which plays the role of the registrar. On this website, the voters authenticate themselves using an identity card or a password provided by the election authorities. The registrar assigns the voter a random alias al and a password, which form the voter's credential. The voter is given the credential, and the registrar also sends it to the voting server.

Voting phase. The voter is asked a multiple choice question, and can pick some of the choices depending on the election's rules. For instance, at most one option may be chosen for uninominal voting. For each of the choices, the voter encrypts either 0 or 1 using the exponential ElGamal encryption scheme, depending on whether they want to pick the choice or not. This produces a sequence of ciphertexts $\boldsymbol{C}$. Then, the voter provides a Zero Knowledge Proof π (ZKP, see Section 2.3) that $\boldsymbol{C}$ is well-formed *i.e.* that it only contains encryptions of 0 and 1 and that the election rules were respected.

Then the voters authenticate themselves with the voting server using their password and send their alias, as well as their ballot $B = (\boldsymbol{C}, \pi)$. The server checks that the password is correct with respect to the given alias, checks that the ballot is valid with respect to the public board and adds an entry $(\mathsf{al} : B)$ in the ballot box. Note that the voters can revote: if there is already an entry of the form $(\mathsf{al} : B')$, B' is replaced by B.

Valid ballots. In Helios, a valid ballot $B = (\boldsymbol{C}, \pi)$ has a valid ZKP π. Also, for all ciphertext $C \in \boldsymbol{C}$, C must not already be on the public board. This means that, for all ballot $B' = (\boldsymbol{C'}, \pi')$ in the board, C must not be one of the ciphertexts of the sequence $\boldsymbol{C'}$.

Open audit. After the voting phase, the ballot box is fixed and the voters are given a few days to check that the ballot that appears right to their alias is correct. If this is not the case, the voter can complain.

Tally. The talliers compute the element-wise product of all the ciphertexts. For instance, suppose that there are three options, so that all the ballots have the form (C_1, C_2, C_3, π). Then the element-wise product of the ciphertexts of two ballots B and B' would be $(C_1C'_1, C_2C'_2, C_3C'_3)$. (In the ElGamal encryption scheme, a ciphertext is a pair of group elements, so that the product of two ciphertexts can be defined by similarly computing the element-wise product.) Let $\boldsymbol{C}$ be the final product. The talliers use a threshold decryption protocol (see Section 2.2.3) to decrypt all the ciphertexts in $\boldsymbol{C}$. By the homomorphic property of the ElGamal encryption scheme, this reveals the number of voters that choose each voting option. This threshold decryption protocol produces a transcript Π which consists of ZKPs of correct decryption.

The main weakness of the Helios protocol is that its security requires a honest server. Indeed, the ballots in the board are not authenticated, so that a cheating server can add ballots arbitrarily, thus committing ballot stuffing. An alternative to Helios is the Belenios protocol [CGG19], where the registrar generates a cryptographic signature key while the server generates the password. Consequently, both need to be corrupted to break the eligibility.

To mitigate the lack of eligibility, Helios has an audit phase where the voters can verify their ballot. This imposes some assumptions on the voters, as auditing is not mandatory: in the UCL election mentioned in [dMPQ09], only 30% of the voters verified their ballot. In addition, it is not reasonable to assume that the voters who choose to abstain would verify their ballot. Nevertheless, the solution from Helios can be considered preferable compared to that of Belenios:

in Belenios, there is no open audit phase before the tally, so that if the registrars and the server are both corrupted, they can *undetectably* add some illegitimate ballots. By contrast, the server in Helios has a chance of being detected when cheating. This illustrates that no trust assumption is perfect: depending on the point of view, a given solution may seem reasonable or unacceptable. This is an inherent difficulty in electronic voting, where we need to convince people that a given protocol is secure. Even if it is actually secure under reasonable trust assumptions, there will always be numerous people that will not trust the system, as illustrated by the recent events in Brazil and the USA.

1.1.4 Classical attacks in electronic voting

Now that we have given a concrete example of an electronic voting protocol, this is a good opportunity to list the most classical attacks in electronic voting.

Replay attack. In academic electronic voting, it is usual to consider that the ballot box is public and may be consulted at any given time. Also, we do not suppose that the contained ballots are anonymous. Therefore, a typical ballot will not contain the chosen voting option but rather an encryption. However, suppose that the adversary wants to learn Alice's choice. Then it may try to replay Alice's ballot using as many corrupted voters as necessary, for instance by copying Alice's ballot. If the replay attack is successful, then Alice's choice contribute many times more than it should have, and can be deduced from the result. In theory, this breaks the different definitions of privacy (see Section 1.3), and an analysis done in [Dav22] reveals that these attacks may be harmful in practice. To prevent them, the usual counter-measure is ballot *weeding*, where duplicate ballots are not accepted on the public board. This is the main reason why Helios' Valid algorithm does not only verify the ZKP, but also checks that there is no duplication. Note that since the ballots of Helios may contain many independent ciphertexts, rather than checking that the whole ballot is not a duplicate, the protocol checks that no ciphertext is a duplicate.

Now, recall that the ElGamal encryption scheme is *malleable*: as explained in Section 2.2.2, it is possible, from the ballot of Alice, to produce some ciphertexts which are different but encrypt the same voting option, hence defeating the ballot weeding strategy. For this reason, it is commonly accepted that the minimal requirement is that the ballot should be non-malleable, and not only indistinguishable from random (see Section 2.2.1 for a definition of IND-CPA and NM-CPA security). In Helios, the non-malleability of the ballot comes from the ZKP: although it is possible to produce some *ciphertexts* from Alice's ballot, creating a valid ZKP for those ciphertexts requires to known the randomness which was used to produce them. Yet, if this randomness was known, one could open Alice's ballot.

Ballot dropping. An other classical attack is based on ballot dropping. Suppose that an adversary (typically the server) is able to arbitrarily *drop* any ballot, *i.e.* that it can decide whether any given ballot will be tallied or not. Obviously, this adversary is able to break individual verifiability since the voters are not guaranteed that their vote will be counted. However, this also breaks privacy (in theory): indeed, if the adversary wants to know Alice's choice, it can drop all the other ballots and deduce the choice from the result of the tally. To defeat such an attack, it is generally assumed that the voters check that their ballot was indeed added in the ballot box; in Helios, for instance, this is the role of the public audit phase.

Replacing a ballot. Now, dropping *all* but one ballot will most likely be detected. A related attack is, however, to *replace* the ballots by a ballot that contains a known voting option. This is possible, for instance, when revoting is allowed while the eligibility is breached: the attacker can wait for a voter to vote, and then replace the legitimate ballot by sending an illegitimate one in the name of the voter. As for ballot dropping, this not only breaks individual verifiability, but

also privacy. To defeat such a scenario, an usual academic "solution" is to assume that the voters check their ballot *at the end* of the voting phase, and not right after casting it. Once again, this is the case in Helios, where the public audit phase takes place after the voting phase, when the ballot can no longer be replaced. The problem with this approach is that it is not clear what we are supposed to do if a problem occurs. Canceling the election would be the safe option, but will most likely be perceived as unacceptable by most people. Also, deciding on a threshold of complains above which the election should be canceled is not easy, as many voters could falsely blame the server for dropping their ballot. Therefore, it is preferable to provide a way for the voters to *vote and go*, *i.e.* to check that their ballot is present during the voting phase. This way, if a problem is detected, the voter can revote.

Nevertheless, using a vote-and-go strategy is not that easy. For instance, consider the following scenario, where the server is dishonest. Assume that the voter would typically vote, say, 3 times before complaining. Then the server can drop the first two ballots and accept the last one; however, it can use the first two ballots to launch a replay attack, using some corrupted eligible voters to cast the two ballots. Indeed, since they were never added to the board, this would not be prevented by the weeding strategy. Helios, unfortunately, seems vulnerable to this attack, as this is the case for most electronic voting system. This illustrates the difference between security in theory and security in practice. In theory, one can argue that the voter may complain as soon as the first ballot is not added to the board, and therefore that the protocol is secure. This, indeed, prevents the attack since the protocol which always abort is technically secure. However, aborting the protocol as soon as a voter complains is not a practical solution. See for instance [BBMP21] for a possible counter-measure.

Clash attacks. Another known attack is the clash attack introduced in [KTV12]. In this attack, the voting devices are considered corrupted; more precisely, it is assumed that the attacker can have two voters use the exact same ballots when they choose the same voting option. Then, even if the voters check that their ballot is indeed in the board, it is possible that only one of the two ballots was actually added, while the other was dropped. Hence, individual verification is breached. Interestingly, the usual cast-as-intended mechanisms do not prevent such an attack since the voting device actually encrypts the desired voting option. In addition, it may be difficult to detect that the randomness used are rigged, since the voting device might use a pseudo-random generator, but which has the same seed as another device.

In Helios, the ballot box contains elements of the form $(\mathsf{al} : B)$ where al is an alias while B is an encrypted ballot. Therefore, it seems that the scheme is not vulnerable to clash attacks. However, suppose that the aliases are generated in some untrusted manner, *i.e.* that the adversary is able to decide which voter gets which alias, and therefore to produce a situation in which two voters have the same alias al_1 instead of al_1 and al_2. Then, as explained is [KTV12], a clash attack would allow not only to break individual verifiability by allowing the adversary to drop some ballots, but also to undetectably break eligibility since the attacker would be able to safely add a ballot, using the remaining alias al_2.

Individual verifiability and privacy. In most of the above attacks, we remark that a breach in the individual verifiability can be transformed into an attack against privacy. This is an illustration of the generic result of [CL18], where privacy is shown to imply individual verifiability under some assumptions. More generally, this shows how the different notions of security in electronic voting are tightly related to each other, which explains why it is so difficult to formalize them individually.

1.2 Addressing verifiability

One of the most important properties of an electronic voting protocol is its verifiability, which prevents the participants to temper with the result of the election. This gives the notion of end-to-end verifiability. However, to enforce end-to-end verifiability, the common strategy is to verify each step individually.

1.2.1 Step by step verification

Intuitively, a voting system is verifiable if there is a way to verify that its result is correct with respect to the voting options chosen by the voters. However, the latter are unknown and the whole point of electronic voting is to determine the former. Consequently, it is not possible to verify the result as a whole; instead, we check that every step of the voting protocol was performed correctly. The first step is to encrypt a voting option into a ballot, which is done thanks to a voting device. If the latter is untrusted, it may encrypt another voting option than the one chosen by the voter; yet, due to the encryption, the voter would not be able to detect it immediately. Therefore, the first step is to provide a way to verify that the voting device did not cheat, which is called *cast-as-intended* verification. In the literature, there are two main methods to provide cast-as-intended verification: return codes (*e.g.*, [GGP15]) and the so-called Benaloh challenge [Ben06], which is presented in Section 9.7 (see [MZR+21] for a more detailed categorization). A recent alternative to the Benaloh challenge was proposed in [BCC+22].

Once the ballot is generated, it is added in a ballot box, which is not necessarily public. However, the communication channels and the server are not trusted, so that the voter must gain some evidence that their ballot actually appears in the ballot box. This is called *recorded-as-cast* verification. In academic electronic voting, the ballot box is a part of the public board, so that the voter can check that the ballot output by the voting device appears on the board.

After the voting phase, the ballots are tallied, and we must also verify that the result of the tally actually corresponds to that of the counting function applied on the voting option encrypted in the ballot box. This is the *tallied-as-recorded* verification. Usually, it is done using ZKP of correct tally (see Section 2.3 for more details about ZKPs).

Finally, the last problem that remains is that of the *eligibility*: all counted ballot must come from an eligible voter, and all eligible voter must have at most one counted ballot. For this purpose, various authentication protocols may be used. In Helios, the voters directly prove their eligibility to the server using an identity card or an authentication website. In other protocols, they prove their eligibility to the registrars, who give them a credential in return, and only ballots cast with a valid credential are counted.

Intuitively, cast-as-intended and recorded-as-cast are the two components of individual verifiability; however, they are not sufficient. Indeed, in the clash attack presented in [KTV12], two voters perform those verifications, but one of them is prevented from voting. Overall, it is difficult to construct a satisfying notion of verifiability from the verification of each step.

1.2.2 End-to-end verifiability

Many attempts were made to formalize the notion of verifiability (*e.g.*, [Ben87, KRS10, KTV11]); see [CGK+16] for a survey. A natural approach is to define verifiability as a whole. For instance, [KTV11] proposes a generic framework where we can define a predicate γ on the execution of the protocol. Intuitively, γ is a *goal* that the protocol must achieve, such as a formalization of "the result is correct". In this context, giving this exact goal that must be verified is the same as

giving a definition of verifiability: if γ is a goal, a voting scheme is verifiable with respect to γ if, whenever $\mathsf{Verify}(\mathsf{PB}, \Pi, r)$ outputs 1, γ is satisfied with overwhelming probability. A straightforward way to satisfy this definition would be to have Verify always output 0. Therefore, it is also required that Tally and Verify must be consistent: for all public board PB, if (r, Π) is the output of $\mathsf{Tally}(\mathsf{PB}, \{s_i\})$, then $\mathsf{Verify}(\mathsf{PB}, \Pi, r) = 1$. Other similar requirements may be used; for instance, we may require that Verify outputs 1, but only for honestly generated public board, or accept that it might output 0 with some negligible probability.

In related works (*e.g.*, [KTV10b, KTV12, KTV14]), a specific verifiability goal is defined with respect to a parameter k: intuitively, γ_k (given by Definition 1) is satisfied when the adversary can cheat for at most k ballots. This gives Definition 2, which can be seen as a formalization of *end-to-end* verifiability: the final result is compared to what is obtained from the choices of the voters. The main weakness of this definition is that it allows an adversary to change arbitrarily the vote of up to k honest voters, and therefore submit k more ballots than there are corrupted voters. In addition, it is not necessarily clear which values of k are acceptable: $k = 0$ would be ideal but too strong since it typically requires that every voter checks their vote; on the other hand, if k is the number of honest voter, then the corresponding notion of verifiability is too weak.

Definition 1 ([KTV11])**.** *Let n be the number of voters and n_H be the number of honest voters. Let $\nu_1, \cdots, \nu_{n_H}$ be the choices of the honest voters and r the result of the election. Recall that $\mathcal{V}$ denotes the set of all possible voting options while* count *denotes the counting function. The goal γ_k is satisfied if there exists $\tilde{c}_1, \cdots, \tilde{c}_n \in \mathcal{V}$, which contains at least $n_H - k$ elements of the multiset $\{\!\{\nu_1, \cdots, \nu_{n_H}\}\!\}$, such that $r = \mathsf{count}(\tilde{c}_1, \cdots, \tilde{c}_n)$.*

Definition 2 (k-verifiability)**.** *Let k be some integer. We say that a voting protocol is k-verifiable if, for all execution of the protocol such that* $\mathsf{Verify}(\mathsf{PB}, \Pi, r) = 1$*, γ_k is satisfied with overwhelming probability.*

An alternative to this definition is given in [CGGI14], in which the authors divide the voters into three subsets: first, the happy voters are the honest voters who submitted a ballot and successfully checked that it was added to the board (*i.e.* they ran the Check protocol, which returned 1); second, the lazy voters are the honest voters who submitted a ballot, but did not check; finally, the corrupted voters are under the control of the adversary. To model the execution of a voting protocol, three game-based definitions are given. In the first, the registrars are honest and the server is malicious; in the second, the server is honest and the registrars are malicious; in the third, the registrars and the server are honest. (In the paper, one can read "dishonest bulletin board"; this refers to the server and not the public board, which is supposed honest.)

Generally, the registrars are trusted as a group (but some may be corrupted) and the server is untrusted. Therefore, we present the first game in Fig. 2 and refer to [CGGI14] for the others. This game can be read as follows. First, the setup takes place at line 1 and a public encryption key is generated. Then, to express that the verifiability must hold for any number of voters n, we let the adversary choose n at line 2 (in reality, the number of voters is fixed by the election authorities, before the setup phase). However, we want n to be at most polynomial in λ, therefore we ask the adversary to write n using sticks, hence the notation 1^n (one operation is required per voter). Afterwards, the registration takes place at line 3 and a credential is generated for each voter. After the registration, the adversary can produce an arbitrary public board PB, an arbitrary result r and a transcript Π, with the restriction that $\mathsf{Verify}(\mathsf{PB}, \Pi, r) = 1$. For this purpose, it can make queries to the oracles $\mathcal{O}_{\mathsf{corrupt}}$ (which corrupts a voter) and $\mathcal{O}_{\mathsf{vote}}$. The latter keeps three tables HV, L and $\mathsf{Checked}$ updated: $\mathsf{HV}_{\mathsf{id}}$ represents the last voting chosen by

$\mathrm{Exp}^{\mathsf{verb}}(\lambda, n_T, t, \mathbb{A})$

1 $\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_{i=1}^{n_T}, \Pi^{\mathsf{S}} \leftarrow \mathsf{Setup}(\lambda, n_T, t)$;
2 $1^n \longleftarrow \mathbb{A}(\mathsf{pk}, \Pi^{\mathsf{S}})$;
3 $(c_i, \pi_i)_{i=1}^{n}, \Pi^{\mathsf{R}} \leftarrow \mathsf{Register}(\mathsf{pk}, n)$;
4 $\mathcal{CU} \longleftarrow \varnothing$;
5 **for** $i = 1$ *to* n **do**
6 $\quad \mathsf{HV}_i \longleftarrow \perp$; $L_i \longleftarrow \perp$; $\mathsf{Checked}_i \longleftarrow 0$;
7 $(\mathsf{PB}, r, \Pi) \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathsf{corrupt}}, \mathcal{O}_{\mathsf{vote}}}(\Pi^R, \{s_i, i \in [1, n_T]\})$;
8 $\mathbb{A}^{\mathcal{O}_{\mathsf{check}}}$;
9 **if** $\mathrm{Verify}(\mathsf{PB}, \Pi, r) = 0$ **then return** 0;
10 **if** $\exists\, \mathrm{L} \subset \{\!\{(i, \mathsf{HV}_i) \mid i \notin \mathcal{CU}, \mathsf{Checked}_i \neq 1, \mathsf{HV}_i \neq \perp\}\!\}$,
$\exists C$ such that $\mid |C| \leq |\mathcal{CU}|$ **and**
$r = \mathsf{count}(\{\!\{(i, \mathsf{HV}_i) \mid i \notin \mathcal{CU}, \mathsf{Checked}_i = 1\}\!\} \uplus \mathrm{L} \uplus \mathrm{C})$
then return 0 **else return** 1;

$\mathcal{O}_{\mathsf{corrupt}}(\mathsf{id})$

1 $\mathcal{CU} \longleftarrow \mathcal{CU} \bigcup \{\mathsf{id}\}$;
2 **return** c_{id};

$\mathcal{O}_{\mathsf{vote}}(\mathsf{id}, \nu)$

1 $B \longleftarrow \mathsf{Vote}_{\mathsf{pk}}(\nu, c_{\mathsf{id}})$;
2 $\mathsf{HV}_{\mathsf{id}} \longleftarrow \nu$;
3 $L_i \longleftarrow B$;
4 $\mathsf{Checked}_{\mathsf{id}} \longleftarrow 0$;
5 **return** B;

$\mathcal{O}_{\mathsf{check}}(\mathsf{id})$

1 $\mathsf{Checked}_{\mathsf{id}} \leftarrow$
$\mathsf{Check}(\nu, c_{\mathsf{id}}, L_{\mathsf{id}}, \mathsf{PB})$

Figure 2: Definition of end-to-end verifiability [CGGI14]

the honest voter id, L_{id} is the corresponding encrypted ballot and $\mathsf{Checked}_{\mathsf{id}}$ is either equal to 1 when the voter successfully performed the Check procedure, or 0. In the paper, it is not clear when the voter is supposed to check. To address this, we added an oracle $\mathcal{O}_{\mathsf{check}}$ which allows the adversary to have a voter initiate this procedure. When the adversary calls $\mathcal{O}_{\mathsf{vote}}$, it gets back the ballot produced by the honest voter. Since the adversary creates arbitrarily the public board, it can decide to add it to the board or not. Finally, verifiability is achieved if $r = \mathsf{tally}(\mathrm{H} \uplus \mathrm{L} \uplus \mathrm{C})$, where H is the multiset of the options chosen by the happy voters, L is included in the multiset of the options chosen by the lazy voters and C is a multiset of options whose size is at most the number of corrupted voters.

Definition 3. *A voting scheme* ($\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{Check}, \mathrm{Valid}, \mathsf{Tally}, \mathrm{Verify}$) *is end-to-end verifiable against a malicious server if, for all PPT adversary* $\mathbb{A}$*, for all* n_T *and* $t < n_T$*, the probability* $\Pr(\mathrm{Exp}^{\mathsf{verb}}(\lambda, n_T, t, \mathbb{A}) = 1)$ *is negligible in* λ*.*

In verifiability definitions, it may be preferable to consider fully corrupted talliers, but the definition could easily be adapted by letting the adversary output the result of the setup phase. Another remark is that this definition assumes a perfect registration phase, where the corrupted registrars cannot learn the voters' credentials, nor interfere with their generation. In electronic voting, we consider this as a reasonable assumption, and distributed key generation schemes can be used to achieve this (for instance, Civitas [CCM08] gives a detailed registration phase in the context of coercion-resistance).

Compared to Definition 2, this does not allow the adversary to cast more ballots than there are corrupted voters. In addition, this gives a comprehensive description of the number of ballots that can be tempered with: the adversary can drop the ballots of the lazy voters, but not change their content. Nevertheless, remark that the definition is hard to verify when revoting is allowed. Indeed, if a lazy voter vote twice, the adversary can drop the second ballot and have the first one be counted instead, which is a winning condition. Consequently, when revoting is allowed, it may be necessary to adapt the definition to allow the adversary to have up to one ballot cast by the lazy voters counted, instead of either the last one or nothing. In any case, the main problem with Definition 3 is that it assumes that the voters check *after* the voting phase, and not right

after casting their ballot. In addition to being arguably less user-friendly, this is actually not ideal, as additional issues may occur in case of a dispute. By contrast, if the voter can check during the voting phase, then it is still possible to revote when an error is detected.

1.3 Four notions of privacy

The main reason why verifiability is hard to achieve is because we also want privacy. Numerous concurrent definitions of vote privacy can be found in the literature, which illustrates that this notion is difficult to formalize. In this section, to take into account the possibility for identity dependent tally functions, we consider generic result functions $\mathsf{tally} : (\mathcal{I} \times \mathcal{V})^* \to \mathcal{R}$ instead of counting functions.

1.3.1 Vote swapping and Benaloh privacy

An usual way to define privacy is by the mean of *vote swapping*. Suppose that, whenever two voters swap their votes, the adversary cannot detect the difference. Then, intuitively, it cannot learn Alice's vote, since Bob might as well be the one who cast it (while Alice casts Bob's vote). This idea is used to defined privacy in formal methods, for instance in [DKR09]. However, it is slightly too restrictive. In his PhD thesis [Ben87], Benaloh presents a more generic definition. In this definition, the choices of the honest voters define a sequence V_0 or V_1, where V_0 and V_1 contain elements of the form $(\mathsf{id}, \nu) \in \mathcal{I} \times \mathcal{V}$, such that $\mathsf{tally}(\mathrm{V}_0) = \mathsf{tally}(\mathrm{V}_1)$. Then, the goal of the adversary is to guess whether the honest voters voting according to the sequence V_0 or V_1. More formally, we give Definition 4, which is a modern restatement of Benaloh's definition.

Definition 4. *A voting system* $(\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{Check}, \mathrm{Valid}, \mathsf{Tally}, \mathrm{Verify})$ *for a result function* tally *is Benaloh-private if, for all* n_T *and* $t < n_T$*, for all PPT adversary* $\mathbb{A}$*, the advantage* $|\Pr(\mathsf{BenPriv}(\lambda, n_T, t, \mathbb{A}) = 1) - 1/2|$ *is negligible in* λ*.*

Definition 4 is based on the game presented in Fig. 3, which can be read as follows. First, the setup takes place, which generates a public key pk and a transcript Π^{s}. Afterwards, the credentials $(c_i)_{i=1}^n$ are generated following the specifications of the voting system. Then the adversary is given the possibility to corrupt a subset of voters, denoted A, after which two possible votes, V_i^0 and V_i^1, are declared for each voter i. Also, we flip a random coin and denote $b \in \{0, 1\}$ the result. Then the adversary gets the credentials of the corrupted voters and is given access to the oracles $\mathcal{O}_{\mathsf{voteLR}}$ and $\mathcal{O}_{\mathsf{cast}}$. The query to $\mathcal{O}_{\mathsf{voteLR}}(i, \nu_0, \nu_1)$ returns a honest ballot B from voter i, which encrypts the option ν_0 (when $b = 0$) or ν_1 (when $b = 1$). In any case, this oracle modifies both V_i^0 and V_i^1, and B is added to the board (this assumes that a ballot obtained with $\mathrm{Vote}_{\mathsf{pk}}$ is always valid). As for the oracle $\mathcal{O}_{\mathsf{cast}}$, it allows the adversary to submit an arbitrary ballot to the ballot box (provided that it is valid). After the voting phase, we check that the multisets formed by the V_i^0's and the V_i^1's give the same partial result, otherwise we abort the game. Since we consider the adversary's advantage, we abort by returning the random bit b which leads to an advantage of 0 in this case. We do not abort with 0 because it is much more convenient to use an absolute value when defining the advantage: suppose that there is an adversary which often makes the wrong guess; then the adversary which makes the opposite guess would often be right. Finally, at the end of the game, the adversary must guess the value of b given the output of the Tally protocol.

This definition has several limitations. First, it does not account for individual verifiability, as the votes are automatically added to the board. In general, any attack against individual verifiability would break privacy, as already mentioned in Section 1.1.4. However, it is common not

$\mathsf{BenPriv}(\lambda, n_T, t, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_{i=1}^{n_T}, \Pi^{\mathsf{S}} \leftarrow \mathsf{Setup}(\lambda, n_T, t)$;
$1^n \longleftarrow \mathbb{A}(\mathsf{pk}, \Pi^{\mathsf{S}})$;
$(c_i, \pi_i)_{i=1}^{n}, \Pi^{\mathsf{R}} \leftarrow \mathsf{Register}(\mathsf{pk}, n)$;
$\mathsf{PB} \longleftarrow \Pi^{\mathsf{S}} || \Pi^{\mathsf{R}}$;
$\mathrm{A} \longleftarrow \mathbb{A}(\mathsf{PB})$;
for $i = 1$ *to* n **do**
  $\mathrm{V}_i^0 \leftarrow \bot$; $\mathrm{V}_i^1 \leftarrow \bot$;
$b \xleftarrow{\$} \{0, 1\}$;
$\mathbb{A}^{\mathcal{O}_{\mathsf{voteLR}}, \mathcal{O}_{\mathsf{cast}}}(\{c_i \mid i \in \mathrm{A}\})$;
if $\mathsf{tally}(\{\!\{\mathrm{V}_i^0 \mid i \notin \mathrm{A}; \mathrm{V}_i^0 \neq \bot\}\!\}) \neq \mathsf{tally}(\{\!\{\mathrm{V}_i^1 \mid i \notin \mathrm{A}; \mathrm{V}_i^1 \neq \bot\}\!\})$ **then return** b;
$r, \Pi \longleftarrow \mathsf{Tally}(\mathsf{PB}, \{s_i\})$;
$b' \longleftarrow \mathbb{A}(r, \Pi)$;
if $b' = b$ **then return** 1 **else return** 0;

$\mathcal{O}_{\mathsf{voteLR}}(i, \nu_0, \nu_1)$

if $i \notin \mathrm{A}$ **and** $\nu_0, \nu_1 \in \mathcal{V}$ **then**
  $\mathrm{V}_i^0 \leftarrow (i, \nu_0)$;
  $\mathrm{V}_i^1 \leftarrow (i, \nu_1)$;
  $B \longleftarrow \mathrm{Vote}_{\mathsf{pk}}(\nu_b, c_i)$;
  $\mathsf{PB} \longleftarrow \mathsf{PB} || B$;
  return B;

$\mathcal{O}_{\mathsf{cast}}(B)$

if $\mathrm{Valid}(B, \mathsf{PB})$ **then**
  $\mathsf{PB} \longleftarrow \mathsf{PB} || B$;

Figure 3: Benaloh's definition of privacy

to consider those attacks in the privacy definition, and to assess the verifiability independently. More generally, it appears that this definition actually assumes a somewhat honest server, while the server is usually supposed malicious in electronic voting. Indeed, not only is the adversary unable to drop or replace ballots, but it is also unable to add invalid ballots in the public board. This is because an invalid ballot would be detected by the auditors, so that it is more convenient to abstract away this possibility in the definition. On the other hand, remark that the adversary can cast any valid ballot without any restriction, while normally one would have to authenticate oneself to the server to be able to cast a ballot.

A second limitation is that this definition does not allow to model corrupted talliers or registrars, since the adversary is not activated during the corresponding phases. In general, the registrars are trusted as a group, and it is common in electronic voting to consider a perfect registration, as already discussed in Section 1.2.2. As for the talliers, the most popular trust assumption is that up to t talliers may be corrupted, while at least $t+1$ secret shares are necessary to recover the decryption key. For most tally protocols, the role of the talliers is very restricted and well understood, and a threshold decryption scheme or a decryption mixnet prevents a malicious tallier from learning anything else than the result. Therefore, it is also common in privacy definitions to consider a perfect tally.

While the aforementioned issues are interesting to mention, they are very common in electronic voting. A more significant limitation is that this definition does not cover every counting functions. For instance, suppose that there are two candidates, A and B, so that the voting options are $\mathcal{V} = \{A, B\}$. In this setting, consider the majority counting function, with the possible results $\mathcal{R} = \{A, B\}$, where A wins in case of an equality. Then the adversary can choose $V_0 = \{\!\{(1, A), (2, B)\}\!\}$ and $V_1 = \{\!\{(1, A), (2, A)\}\!\}$, which indeed have the same tally (A wins in both cases). However, when adding a ballot in favor of B, the adversary makes B win in the first case while A still wins in the other. Therefore, when the majority function is used, a voting system can never be Benaloh-private. In fact, the notion of Benaloh-privacy only makes sense when the counting function has the partial tally property, which is introduced in Section 1.1.2.

Algorithm 1: $\mathrm{Exp}^{\text{s-cons}}(\lambda, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_{i=1}^{n_T}, \Pi \leftarrow \mathsf{Setup}(\lambda, n_T, t)$;
$\boldsymbol{B} \longleftarrow \mathbb{A}(\mathsf{pk}, \Pi)$;
$\mathsf{PB} \longleftarrow \varnothing$;
for $B \in \boldsymbol{B}$ **do**
  if $\mathrm{Valid}(B, \mathsf{PB}) = 1$ **then** $\mathsf{PB} \leftarrow \mathsf{PB}||B$;
$r, \_ \longleftarrow \mathsf{Tally}(\mathsf{PB}, \{s_i\})$;
if $r \neq \mathsf{tally}((\mathrm{Extract}_{\mathsf{sk}}(B))_{B \in \mathsf{PB}})$ **then return** 1 **else return** 0;

Algorithm 2: $\mathrm{Exp}^{\text{s-corr}}(\lambda, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_{i=1}^{n_T}, \Pi \leftarrow \mathsf{Setup}(\lambda, n_T, t)$;
$\mathsf{id}, \nu, \boldsymbol{B} \longleftarrow \mathbb{A}(\mathsf{pk}, \Pi)$;
if $(\mathsf{id}, \nu) \notin (I \times \mathcal{V})$ **then**
  return 0
$\mathsf{PB} \longleftarrow \Pi||\boldsymbol{B}$;
$B \longleftarrow \mathsf{Vote}_{\mathsf{pk}}(\nu, \mathsf{id})$;
if $\mathrm{Valid}(B, \mathsf{PB}) = 0$ **then return** 1 **else return** 0;

1.3.2 Ballot privacy and ideal tally

When the counting function has partial tally, Benaloh's definition is perfectly suitable and easy to use; however, it does not apply to tally functions which do not have partial tally. In an effort to capture more attacks and to cover more cases, [BCG+15b] gives a definition based on *ballot privacy*. Intuitively, a voting system is ballot private if no information about the voting options chosen by the honest voters can be inferred by observing and submitting ballots. Consequently, in a ballot private voting system, the only information that the adversary has is the result of the tally, which is considered unavoidable. To define ballot privacy more formally, we first introduce the notions of *strong consistency* and *strong correctness*, which describe two assumptions on the voting system. For this purpose, we use the conventions of [BCG+15b], where the authors do not consider the registration phase: eligibility and authentication is considered to be taken care of in an independent way. Consequently, their voting process takes as arguments the voting option ν and the identity id instead of the voting option and the credential. (In general, the identity – or alternatively an alias – can be a part of the credential.)

Strong consistency. A voting system typically provides a Tally protocol that computes a result from the encrypted ballots. On the other hand, a counting function also computes a result, but directly from the voting options. Consequently, it is important to make sure that the Tally protocol actually outputs the same result as the tally function, when applied to the voting options chosen by the voters. This is called correctness. In [BCG+15b], this is captured by the notion of *strong consistency*, defined in Definition 5. Compared to the natural notion of correctness, it is required that the Tally protocol is correct, even if the public board was created maliciously.

Definition 5. *A voting system* (Setup, Register, Vote, Check, Valid, Tally, Verify) *for a result function* $\mathsf{tally} : (\mathcal{I} \times \mathcal{V})^* \longrightarrow \mathcal{R}$ *has strong consistency if there exists an* Extract *algorithm such that, for all key pair* $(\mathsf{pk}, \mathsf{sk})$*, for all* $(\mathsf{id}, \nu) \in (\mathcal{I} \times \mathcal{V})$*,* $\mathrm{Extract}_{\mathsf{sk}}(\mathsf{Vote}_{\mathsf{pk}}(\nu, \mathsf{id})) = (\mathsf{id}, \nu)$*. In addition, for all PPT adversary* $\mathbb{A}$*, the probability that* $\mathrm{Exp}^{\text{s-cons}}(\lambda, \mathbb{A}) = 1$ *is negligible in* λ*, where* $\mathrm{Exp}^{\text{s-cons}}$ *is defined in Algorithm 1.*

Strong correctness. Another desirable property, which we already used implicitly in Section 1.3.1, is that the Vote and Valid algorithms must be consistent. Namely, if a honest voter produces a ballot B using Vote, then $\mathrm{Valid}(B, \mathsf{PB})$ should output 1 independently of the public board. In [BCG+15b], this is captured by *strong correctness*, defined in Definition 6.

Definition 6. *A voting system* (Setup, Register, Vote, Check, Valid, Tally, Verify) *for a result function* $\mathsf{tally} : (\mathcal{I} \times \mathcal{V})^* \longrightarrow \mathcal{R}$ *has strong correctness if, for all PPT adversary* $\mathbb{A}$*, the probability that* $\mathrm{Exp}^{\text{s-corr}}(\lambda, \mathbb{A}) = 1$ *is negligible in* λ*, where* $\mathrm{Exp}^{\text{s-cons}}$ *is defined in Algorithm 2.*

Ballot privacy. Strong correctness and strong consistency are two desirable properties that any voting system should verify. However, they tell nothing about privacy. This notion is captured by *ballot privacy*, given in Definition 7. This definition is based on the BPRIV game, depicted in Fig. 4, where the adversary has access to the oracles $\mathcal{O}_{\mathsf{voteLR}}, \mathcal{O}_{\mathsf{cast}}, \mathcal{O}_{\mathsf{board}}$. The idea is that there are two concurrent public boards, PB_0 and PB_1. Depending on the value of a random coin b, the adversary can only see one of them. $\mathcal{O}_{\mathsf{voteLR}}(\mathsf{id}, \nu_0, \nu_1)$ allows the adversary to have the voter id submit a ballot for the voting option ν_0 (resp. ν_1) in PB_0 (resp. PB_1); while $\mathcal{O}_{\mathsf{cast}}(B)$ allows the adversary to cast the ballot B in both boards. However, B must be valid with respect to the current board PB_b. Note that this leads to a definition glitch, as the adversary may be able to insert invalid ballots in PB_0 when $b = 1$. For instance, consider a Tally protocol that aborts (*i.e.* returns $r = \perp$ and $\Pi = \varnothing$) if there is an invalid ballot on the board, and suppose that the Valid algorithm checks that the first bit of the ballot is 0. If not, it checks whether the second bit of the ballot is equal to the XOR of all the bits on the board, and rejects the ballot if this is not the case (in addition, it performs the usual operations, such as verifying the ZKPs and weeding). Suppose that the Vote algorithm always result in ballots that begin with two zero bits, so that strong correctness is preserved. Then the adversary can use this algorithm to produce a valid ballot but replace the first bit by 1, and the second by the XOR of all the bits of PB_b. With high probability, when $b = 1$, this ballot is valid in PB_b but not in PB_0, therefore causing the Tally protocol to abort. On the other hand, when $b = 0$, a valid ballot in PB_b will not cause an abortion. A possible fix to this issue would be to check that the cast ballots are valid in both boards instead of just PB_b. However, this needs a careful analysis of whether the remaining properties are preserved, so that we prefer to present the original version instead of the "fixed" one.

In ballot privacy, the goal of the adversary is to guess the value of b. Intuitively, this corresponds to distinguishing the case where the honest voters voted for two arbitrary sequences of voting options, that just have the same length and are no longer restricted to having the same partial tally. Since the result of the tally would a priori leak which distribution was used, we always give the adversary the result obtained from PB_0. To express the fact that the Tally process itself could leak some information, we also demand that there exists a PPT simulator SimProof which is able to simulate the transcript of the tally. In general, the usual verifiability mechanism prevents the talliers from forging a fake tally, which means that SimProof must somehow subvert the verifications performed by the adversary. For this purpose, it is allowed to interact with it in the random oracle model (see Section 2.3.4 for more details about non-interactive zero-knowledge proofs). In addition, if we consider that the transcript of the tally contains some partial decryptions (which are not ZKP), the simulator must also simulate them, which means that it needs the secret shares of the corrupted talliers. (See for instance Theorem 1 to see how to simulate the partial decryptions in the specific case of a honestly generated ElGamal encryption.) In the original definition, the secret key is not shared between several talliers, so that there is a single tallier which is considered honest. We feel that this is a bit restrictive, as it is common in electronic voting to distribute the key among several authorities. Therefore, we took the liberty to modify the definition a bit: in Definition 7, there can be several trusties but they are all considered honest. A similar modification was done in [BPW12, Definition 5], in the specific case where the threshold t is equal to the number of talliers minus 1 (*i.e.* all the talliers need to collaborate in order to decrypt). The intuition behind this simulator which can magically recover the shares of the corrupted participants is that the transcript Π, output by Tally along with the result r, does not contain any useful information that can be exploited by the adversary. Indeed, the latter might have produced Π itself using SimProof. This paradigm is a fundamental notion in ZKP, which are the subject of Section 2.3.

$\mathsf{BPRIV}(\lambda, \mathbb{A})$

1 $\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_{i=1}^{n_T}, \Pi \longleftarrow \mathsf{Setup}(\lambda, n_T, t)$;
2 $\mathsf{PB}_0 \longleftarrow \Pi$; $\mathsf{PB}_1 \longleftarrow \Pi$;
3 $b \xleftarrow{\$} \{0,1\}$;
4 $\mathbb{A}^{\mathcal{O}_{\mathsf{voteLR}}, \mathcal{O}_{\mathsf{cast}}, \mathcal{O}_{\mathsf{board}}}(\mathsf{pk})$;
5 $r, \Pi_0 \longleftarrow \mathsf{Tally}(\mathsf{PB}_0, \{s_i\})$;
6 $\Pi_1 \longleftarrow \mathrm{SimProof}(\mathsf{PB}_1, r)$;
7 $b' \longleftarrow \mathbb{A}(r, \Pi_b)$;
8 **if** $b = b'$ **then return** 1;
9 **else return** 0;

$\mathcal{O}_{\mathsf{voteLR}}(\mathsf{id}, \nu_0, \nu_1)$

1 $B_0 \leftarrow \mathsf{Vote}_{\mathsf{pk}}(\nu_0, \mathsf{id})$, $B_1 \leftarrow \mathsf{Vote}_{\mathsf{pk}}(\nu_1, \mathsf{id})$;
2 $\mathsf{PB}_0 \leftarrow \mathsf{PB}_0 || B_0$; $\mathsf{PB}_1 \leftarrow \mathsf{PB}_1 || B_1$;

$\mathcal{O}_{\mathsf{cast}}(B)$

1 **if** $\mathrm{Valid}(B, \mathsf{PB}_b) = 1$ **then**
2 $\quad \mathsf{PB}_0 \leftarrow \mathsf{PB}_0 || B$; $\mathsf{PB}_1 \leftarrow \mathsf{PB}_1 || B$;

$\mathcal{O}_{\mathsf{board}}()$

1 **return** PB_b;

Figure 4: Ballot privacy game

Definition 7. *A strongly consistent and strongly correct voting system* (Setup, Register, Vote, Check, Valid, Tally, Verify) *for a result function* tally *has ballot privacy if there exists an algorithm* SimProof *s.t. for all PPT* $\mathbb{A}$*, the advantage* $|\Pr(\mathsf{BPRIV}(\lambda, \mathbb{A}) = 1) - 1/2|$ *is negligible in* λ.

Interestingly, remark that although we said that strong correctness and strong consistency were unrelated to privacy, they are actually required in Definition 7. Indeed, consider a "normal" tally function which returns the number of votes for each candidate, and a Tally protocol which has correctness (*i.e.* the output r, Π is such that r is the same as the output of tally when Tally is applied to honestly generated ballots) but not strong consistency. Then there is no guarantee about what the output of Tally should be when the board contains maliciously generated ballots. In particular, Tally could output a list of elements of the form (name : vote), which would intuitively break privacy. However, since the adversary only learns the tally of the left world (when $b = 0$), this does not help when it comes to attacking ballot privacy. For this reason, it is required that the voting system has strong consistency. Also, strong correctness is required to make sure that $\mathcal{O}_{\mathsf{voteLR}}$ does not result in any invalid ballot being added to a board.

Compared to Benaloh's definition, ballot privacy is not restricted to any specific result function. Although [BCG+15b] does not consider the registration, it is possible to adapt their definition to add a registration phase and credentials. Another advantage of their definition is that, when it is verified, then the voting protocol is proven to be equivalent to an ideal functionality which returns the output of the result function, evaluated on the choices made by the voters. This result uses an universally composable framework, as the one presented in Section 3.3.

The notion of ballot privacy has a lot of advantages, but still implies some limitations. First, it is fairly difficult to understand, to use and to adapt. Second, it assumes a honest public board, as this is often the case in electronic voting (see [CLW20] for an attempt to define ballot privacy against a malicious board). However, the most prominent limitation is that it assumes a perfect tally, where the adversary is not activated. Nevertheless, for a result function with no partial tally, the usual strategies to compute the tally, namely homomorphic tally and mixnets, cannot be used. Therefore, the tally protocol could be complex and it is no longer justified to consider a perfect tally. Yet, in ballot privacy, the adversary must decide if a public board PB_b is the real board or not. Since Tally requires the public board as an input, the adversary would trivially win the game if it could impersonate a tallier.

1.3.3 A quantitative definition of privacy

In most definitions of privacy, we abstract away the information given by the result of the election. For instance, Benaloh's definition asks for the two multisets V_0 and V_1 to yield the same partial result. As for ballot privacy, it states that the adversary has no more information than that of the result. However, the result itself can already contain a lot of information. As an example, if there are two new votes for a right-wing party after a couple arrived in a small left-wing village, then the villagers can deduce the political orientation of the couple. Consequently, a complementary approach to qualitative privacy is to quantify the actual level of privacy provided by a protocol. This is the approach of [KTV11], which gives a generic framework for this purpose.

In their framework, the participants are modeled as interactive Turing machines (ITM), and the voting protocol as a whole is modeled as a concurrent execution of the participants. This is an extremely generic approach, which can be applied to any voting protocol, with any trust assumption, without any assumption on the counting function. For instance, this allows to properly take into account the presence of corrupted authorities or to introduce a malicious public board. The honest participants are merged into a single process e, which depends on a parameter $\overrightarrow{p}$ that represents the probability for each voting option to be chosen. Concretely, if there are three voting options, $\overrightarrow{p}$ is a vector of three positive reals whose sum is 1. (To simplify, we include abstention into the possible voting options.) If a voting protocol is the description of all the honest participants as ITM, then e can be seen as a subset of the protocol which determines the honest participants in an instantiation of the protocol. In addition to the honest participants (which can include the voters, the talliers, the registrars or any additional authority), there is one specific voter (say, Alice) which is not corrupted but not included in e; instead, she is considered *under observation*. For a given voting option ν, we use the notation $\pi_v(\nu)$ to denote Alice's algorithm (which is the same as the other honest voters, except that she always chooses the option ν). In [KTV11], the goal of the adversary is to guess Alice's behavior. In this paper, the adversary (denoted π_o) is an arbitrary process which can impersonate the participants other than Alice and the honest ones. As such, the adversary can deviate from the protocol. With these notations, the execution of a voting protocol is denoted $\pi_o||\pi_v(\nu)||e$, where $||$ stands for the concurrent execution. For such an execution, we only consider the output of the adversary. When the latter is 1, we use the notation $\pi_o||\pi_v(\nu)||e \mapsto 1$. We are now ready to give Definition 8, extracted from [KTV11].

Definition 8. *Let P be a voting protocol, $\pi_v \in P$ a voter under observation and $e \subset (P \backslash \{\pi_v\})$ a set of honest participants. For $\delta \in [0,1]$, we say that e achieves δ-privacy if, for all PPT adversary π_o, for all voting options ν_0, ν_1, the difference $|\Pr(\pi_o||\pi_v(\nu_0)||e \mapsto 1) - \Pr(\pi_o||\pi_v(\nu_1)||e \mapsto 1)|$ is δ-bounded.*

A first remark is that δ-privacy is defined with respect to a set e of honest participants. If one wants the privacy level of a given protocol under a specific set of trust assumptions, one should find a δ such that, for any e compatible with the trust assumptions, e achieves δ-privacy. This is tedious, as finding one relevant δ for a specific e is already difficult. Another remark is that, even though this definition is very generic, it restricts privacy to guessing how a specific voter voted, which is simplistic compared to the other definitions presented in this section. Finally, it only gives a quantitative definition of privacy, which does not include a comprehensive way to determine which value of δ is acceptable or not. Consequently, this gives a complementary approach rather than a concurrent one.

1.3.4 Our approach: comparing a real and an ideal process

In this thesis, we analyze the security of electronic voting systems which rely on complex tally protocols. Hence, it is natural to use a notion of privacy that captures the possibility of corrupted talliers, which is not possible in most existing game-based definitions (such as those presented in Sections 1.3.1 and 1.3.2). An alternative is to use the approach of [KTV11], presented in Section 1.3.3. However, while being able to quantify privacy is certainly of interest, we prefer to give a qualitative definition which allows deciding whether a given protocol provides privacy or not. In addition, describing the participants as ITMs is not a common practice in electronic voting. Rather, a protocol is given for each phase. For this reason, we propose another approach, which is closer to the usual game-based definitions. The idea is to compare the privacy level of a voting system to that granted by an ideal functionality which collects the choices of the voters (including the corrupted ones), evaluates the tally function and returns the result to the adversary. To define the level of privacy, we used the idea of [KTV11]: the privacy level of a protocol is given by the infimum of the δs for which the protocol provides δ-privacy.

To use this approach, we need to design two games to define the privacy of a voting protocol. The first one, the ideal game, presents our model of privacy: it describes the winning conditions of the adversary and the information it has access to. For instance, the adversary may know the probability distribution for the voting options and learn the result of the election. In addition, it may also learn the number of ballots cast, which can be deduced from the public board. This is all modeled in the ideal game, which should remain as simple as possible. A not-quite-so-simple example is given in Algorithm 4, where the adversary tries to guess Alice's vote as in [KTV11]. Alice is modeled as the voter j, while the corrupted voters are modeled by the set A. In this ideal game, we consider a fixed set $[1, n_C]$ of valid voting options (excluding abstention), and a family of distributions $\mathscr{B}$. For all set of aliases $\mathcal{I}$ and all number of possible voting options n_C, $\mathscr{B}(\mathcal{I}, n_C)$ is a random variable over $(\mathcal{I} \times \mathcal{V})^*$, with $\mathcal{V} = [1, n_C]$. Intuitively, the probability distribution of $\mathscr{B}(\mathcal{I}, n_C)$ represents the probability that each possible sequence is chosen by the honest voters, whose aliases are described in $\mathcal{I}$. Since we only consider the finite sequences, there is a countable number of possible outcomes so that we are still in the domain of discrete probabilities. We assume that the adversary knows a perfect description of $\mathscr{B}$, which is a generalization of the assumptions made in [KTV11] where the distribution $\overrightarrow{p}$ is known. This generalization may seem abusive since the adversary should only be able to handle information of polynomial size, while a full description of $\mathscr{B}$ would technically require a countably infinite bitsize. To fix ideas, the reader can consider that for each $(\mathcal{I}, n_C)$, there are only a finite number of sequences that can be chosen with a non-zero probability, which solves this technical issue. Remark that in the ideal game, we do not let the adversary choose the number of voters n. This is because the ideal game is to be compared to a real game, where the adversary cannot choose n lest it would always make the most advantageous choice. Consequently, we ask that the probability to win in the real game should be the same as in the ideal game, this for all possible parameters (including n and $\mathscr{B}$). Another remark is that in the ideal game, the adversary is given the sequence I of the identity of the honest voters. This is because we consider that the votes are not cast anonymously; however, it is possible to adapt the definition to consider an ideal game where the votes are cast anonymously: in this case, the adversary is given $|B|$ instead of I, where $|B|$ is the number of ballots cast by a honest voter.

Once the ideal game is defined, we need to design the real game, which should be as close as possible to the real protocol. The description of the real game factors the trust assumptions, the assumptions on the communication channels and, of course, the notion of privacy (which should be the same as in the ideal game). In this thesis, we used the game defined in Algorithm 3,

Algorithm 3: $\text{Real}^{\mathsf{Priv}}$

Requires: $\lambda, n_T, C_t, n, n_A, n_C, \mathscr{B}, \mathbb{A}$

1 $\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_{i=1}^{n_T}, \Pi^{\mathsf{S}} \leftarrow \mathsf{Setup}(\lambda, n_T, t)$;
2 $(c_i, \pi_i), \Pi^{\mathsf{R}} \longleftarrow \mathsf{Register}(\mathsf{pk}, n)$;
3 $\mathsf{PB} \longleftarrow \Pi^{\mathsf{S}} || \Pi^{\mathsf{R}}$;
4 $A \longleftarrow \mathbb{A}(\mathsf{pk}, \mathsf{PB}, \{s_i \mid i \in C_t\})$;
5 $j, \nu_0, \nu_1 \longleftarrow \mathbb{A}(\{c_i \mid i \in A\})$;
6 (* chooses the voter to observe *);
7 **if** $|A| \neq n_A \vee j \notin [1, n] \backslash A$ **then**
8 **return** 0;
9 $B \xleftarrow{\$} \mathscr{B}([1, n] \backslash A, n_C)$;
10 **for** $(i, \nu_i) \in B$ **do**
11 $\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(i, \mathsf{PB})$;
12 $\mathsf{PB} \longleftarrow \mathsf{PB} || \mathsf{Vote}_{\mathsf{pk}}(\nu_i, c_i)$;
13 $\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(i, \mathsf{PB}, \texttt{"end for"})$;
14 $b \xleftarrow{\$} \{0, 1\}$;
15 $\mathsf{PB} \longleftarrow \mathsf{PB} || \mathsf{Vote}_{\mathsf{pk}}(\nu_b, c_j)$;
16 $\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(\mathsf{PB})$;
17 $r, \Pi \longleftarrow \mathsf{Tally}^{\mathbb{A}}(\mathsf{PB}, \{s_i\})$;
18 $b' \longleftarrow \mathbb{A}()$;
19 *if* $\nu_0, \nu_1 \in [1, n_C] \wedge b' == b$ **then** **return** 1 **else return** 0;

Algorithm 4: $\text{Ideal}^{\mathsf{Priv}}$

Requires: $\lambda, n_T, C_t, n, n_A, n_C, \mathscr{B}, \mathbb{A}$

1 ;
2 ;
3 ;
4 $A \longleftarrow \mathbb{A}(\lambda)$;
5 $j, \nu_0, \nu_1 \longleftarrow \mathbb{A}()$;
6 (* chooses the voter to observe *);
7 **if** $|A| \neq n_A \vee j \notin [1, n] \backslash A$ **then**
8 **return** 0;
9 $B \xleftarrow{\$} \mathscr{B}([1, n] \backslash (A \bigcup \{j\}), n_C)$;
10 $(\nu)_{i \in A} \longleftarrow \mathbb{A}(I)$;
11 $B \longleftarrow B || (i, \nu_i)_{i \in A, \nu_i \in [1, n_C]}$;
12 ;
13 ;
14 $b \xleftarrow{\$} \{0, 1\}$;
15 $B \longleftarrow B || (j, \nu_b)$;
16 ;
17 $r \longleftarrow \mathsf{tally}(B)$;
18 $b' \longleftarrow \mathbb{A}(r)$;
19 **if** $\nu_0, \nu_1 \in [1, n_C] \wedge b' == b$ **then** **return** 1 **else return** 0;

Figure 5: Definition of privacy, λ is the security parameter, n_T the number of talliers, t the threshold, C_t the set of the corrupted talliers, n the number of voters, n_A the number of corrupted voters, n_C the number of voting options (excluding abstention) and $\mathscr{B}$ the distribution.

which can be read as follows. First, the setup and the registration take place in an ideal way. As discussed previously, this is a common abstraction which can be enforced with DKG protocols. However, to take into account the possibility of a static corruption of up to t talliers, where t is the threshold, we give the secret shares of the corrupted talliers to the adversary at line 4. Since we consider static corruption (and not dynamic corruption), we consider that the set C_t of the corrupted talliers is fixed in advance, as a parameter of the privacy experiment. Afterwards, the adversary must corrupt exactly n_A voters and chooses a voter under observation j. For this voter, it also chooses two possible voting options, ν_0 and ν_1. Then the honest voters vote according to a distribution $\mathscr{B}$ and, thanks to an oracle access to $\mathcal{O}_{\mathsf{cast}}$, we let the adversary freely insert any number of ballots between two ballots cast by a honest voter, provided that the ballots are valid. To model the fact that the ballots are not sent anonymously, we give away the identity of the voter to the adversary at line 12. Also, to express that the voter under observation could revote a certain number of times, we include j as a possible identity for the honest voters at line 9. Then the voter under observation (re)votes with either ν_0 or ν_1 and the adversary is given a last opportunity to cast ballots; after this, the tally is computed with the Tally protocol, during which the adversary can impersonate the corrupted talliers. To emphasize the fact that the adversary is active during the tally phase, we use the notation $\mathsf{Tally}^{\mathbb{A}}$. Finally, the adversary must guess whether ν_0 or ν_1 was chosen.

Once the two games are defined, we define privacy by comparing the probability that the adversary wins in both games. If the difference is non-negligible, it means that the cryptographic protocol gave the adversary some information to exploit. On the other hand, if the difference is negligible, then the adversary has no more information than in the ideal game, and we say that the protocol is private. A critical point to understand is that we require the difference to be negligible for *all* $\mathscr{B}$. For some distribution (*e.g.*, when all the voters choose the same option), there may be no privacy at all; for others, it may be more difficult to determine Alice's vote; in any case, the privacy level of the real game should be the same as the one in the ideal game.

Definition 9. *A voting system* (Setup, Register, Vote, Check, Valid, Tally, Verify) *for a result function* tally *guarantees* vote privacy *if, for all parameters* n_T, t, n, n_A, n_C *with* $t < a$ *and* $n_A \leq n$*, for all subset* $C \subset [1, n_T]$ *of size at most* t*, for all familly of distributions* $\mathscr{B}$ *and for all PPT adversary* $\mathbb{A}$*, there exists a PPT adversary* $\mathbb{B}$ *and a negligible function* μ *such that*

$$|\Pr(\mathrm{Real}^{\mathsf{Priv}}(\lambda, n_T, C_t, n, n_A, n_C, \mathscr{B}, \mathbb{A}) = 1) - \Pr(\mathsf{Ideal}^{\mathsf{Priv}}(\lambda, n_T, C_t, n, n_A, n_C, \mathscr{B}, \mathbb{B}) = 1)| \leq \mu(\lambda).$$

1.3.5 Encountering a new definition: the survival manual

Numerous definitions of privacy can be found in the literature. Indeed, when one wants to prove the privacy of a specific voting protocol, one may encounter several difficulties in adapting the existing definitions to the specificities of their protocol. Typically, we defined a voting system as a tuple (Setup, Register, Vote, Check, Valid, Tally, Verify), where Setup is assumed to use a secret sharing scheme. However, other authors may want to abstract away registration; to divide the Check algorithm into two (one for the cast-as-intended verification, the other for the recorded-as-cast verification); to have the voters perform checks in several steps (for instance, by the use of return codes); or to see the voting protocol as a more interactive process, which may lead to several intermediate values being added to the board or exchanged through potentially unsecure channels. Another example is coercion-resistance, which is the subject of Part III: in this setting, we usually require an additional algorithm to produce a fake credential. In addition, a protocol may want to specify the exact role of the server, and have several processes that can be run in interaction with it, such as voter authentication. Therefore, it is common to design one's own definition of privacy. Conversely, a definition of privacy is often to be understood in a protocol's context, and, although there might be similarities, it is difficult to come up with a generic definition that is actually usable. Therefore, it is important to have a good methodology for encountering a new definition.

The most important thing to understand is what is actually being modeled. If this information is not given by the authors, the reader should try to find out what the adversary must do to break privacy, as defined in the definition. For this purpose, the comparison to existing definitions may be useful. In Benaloh's definition, the adversary must distinguish two distributions V_0 and V_1, with the restriction that they must have the same partial tally; in ballot privacy, the adversary must distinguish any two distributions of the same size; in Definition 9, the adversary tries to guess whether a specific voter votes for the option ν_0 or the option ν_1. If the adversary's goal does not intuitively match breaking privacy, or seems too simplistic compared to existing definitions, it is possible that the definition is not relevant or too restrictive. For instance, Definition 9 gives a less refined notion of privacy compared to Benaloh's definition or ballot privacy. However, it is similar to the existing definition of [KTV11].

Understanding the model also means understanding the inlined assumptions and abstractions, and whether they allow modeling the desired trust assumptions. Ideally, this should be discussed by the authors, but it is important to verify that nothing was omitted. In ballot privacy, the

public board and the talliers are honest, the ballots are automatically added to the board and the server is completely abstracted away. Also, registration is not considered, which means that additional verifications must be done to ensure eligibility, otherwise privacy may be lost. Clearly, this deviates from the desired trust assumptions where the server is malicious while some registrars and up to a threshold of talliers may be corrupted. This is quite common in the literature and we gave some justifications in Section 1.3.1.

Once the abstractions and simplifications are clear, the reader can compare them to usual abstractions, see how they are justified and look for attacks that are prevented due to those abstractions. In Definition 9, we have the same abstractions as in ballot privacy, except that this definition considers corrupted talliers during the tally phase. In particular, we do not let the adversary drop any ballot, which certainly prevents some attacks. Clearly, a server that drops all but Alice's ballot will deduce Alice's choice from the result of the tally, but such an attack would be prevented by the individual privacy mechanism. Nevertheless, other attacks may be possible, depending on the protocol. As an example, consider the Helios voting system, and assume that Alice casts the ballot B. Suppose that the server drops this ballot, but that Alice detects it because individual verifiability is enforced. At this point, Alice will most likely revote with another ballot that encrypts the same voting option. Indeed, she has no reason to change her mind and the incident may have been caused by a network issue. Before that, however, the server can submit B using a corrupted voter (so that this attack is still possible when eligibility is enforced). At this point, B is valid and added to the board, so that the server successfully performed a replay attack on Alice's ballot. This specific attack can easily be prevented by making the couple (id, B) non-malleable (see Section 2.3 for more details), but this is not the case in Helios. A similar attack against Belenios is described in [BBMP21], where the adversary is able to swap two revotes of the same voter, hence breaking individual verifiability. This illustrates that some attacks may be missed because of the definition. To detect them, the reader may try all known attacks, such as replay attacks or clash attacks, and try to push the trust assumptions to the limits by considering a malicious server, some corrupted registrars and voters, and up to a threshold of talliers that collude to learn Alice's choice. Finally, an interesting exercise to gain confidence about a definition is to look for systems that do and do not verify it. The reader may use their own home-made systems, or known voting systems, to see if they are private with respect to the considered definition.

To close this discussion, we comment on Definition 9, that we used in this thesis. We did not use Benaloh's definition because we consider several counting functions which did not have partial tally (see Part II), and we did not use ballot privacy either because we wanted to allow the adversary to impersonate the talliers during the tally phase. For these reasons, we came up with Definition 9. In hindsight, this definition is more complex than ballot privacy and more restrictive than Benaloh's definition, so that it is far from being ideal. Creating a simple and comprehensive definition of privacy that can apply to *any* counting function while still allowing to take into account the corruption of up to a threshold of talliers would be an interesting future work.

Chapter 2

Cryptography in electronic voting

The main difficulty in electronic voting is to provide privacy and verifiability simultaneously. For this purpose, electronic voting protocols rely a lot on public key cryptography, especially on homomorphic encryption and zero knowledge proofs. To better understand the remaining of this thesis, it is necessary to have a look at the main cryptographic primitives used.

2.1 Computational assumptions in electronic voting

2.1.1 Algebraic notations for cryptography

In public key cryptography, it is common to use algebraic structures, such as rings or groups. Technically, a group is a tuple $(G, \cdot)$ while a ring is a tuple $(R, +, \times)$; however, it is common to drop the composition laws when denoting a group or a ring. In this thesis, the only rings that we consider are the ring of the integers $\mathbb{Z}$ and, for $n > 1$, the finite ring $\mathbb{Z}_n$ of the integers modulo n. For a ring R, the multiplicative group of its invertible elements is denoted $R^\times$. The monoid of the non-negative integers is denoted $\mathbb{N}$. For groups, we consider generic finite abelian groups (for instance, elliptic curves). Although the convention for elliptic curves is to use an additive notation, we use a multiplicative notation for all groups (the additive notation is only used for ring elements).

Now, each group G has a structure of $\mathbb{Z}$-module: for any integer $n \in \mathbb{Z}$ and $a \in G$, a^n is obtained by composing n times the neutral element 1 with a (if $n < 0$, $a^n = 1/a^{-n}$). This is called an exponentiation. In particular, $a^0 = 1$. (In some specific cases where there could be an ambiguity, we use the notation 1_G to denote the neutral element.) As we only consider finite groups, this $\mathbb{Z}$-module is actually a $\mathbb{Z}_q$-module, where q is the order of the group. In this case, for any two integers $a, b \in \mathbb{Z}$ such that b is coprime with q, there exists an element $c \in \mathbb{Z}$ such that $a = bc$ modulo q. Hence, for any $g \in G$, $g^{a/b}$ can be defined as g^c. In this thesis, we usually consider cyclic groups of prime order, so that $\mathbb{Z}_q$ is a field and any non-neutral element is a generator. Given the above remark, it follows that such groups have the structure of a vector space over $\mathbb{Z}_q$.

2.1.2 The decisional Diffie-Hellman assumption

As soon as one sees a vector space, one naturally wants to use the known effective results from linear algebra. This is not possible because of the discrete logarithm problem. The latter consists, given two group elements $g, h \in G$, to find an integer n such that $g^n = h$. This problem is easy in $(\mathbb{Z}_q, +)$ (where what we denoted an exponentiation is actually a regular multiplication), and

can be solved thanks to the extended Euclidean algorithm. However, it is hard in general. Two related problems are the computational Diffie-Hellman and the decisional Diffie-Hellman (DDH) problems. Given g, g^a and g^b, the first problem is to compute g^{ab}, and the second requires to distinguish it from a random group element. In other words, given two vectors (g_1, g_2) and (g_3, g_4) in the vector space G^2, the DDH problem is to decide whether they are collinear or not. When this is the case, we say that (g_1, g_2, g_3, g_4) is a DDH tuple. Clearly, the discrete logarithm problem is harder than the computational Diffie-Hellman problem, which is harder than DDH. Nevertheless, the DDH problem is widely considered hard in computer science: it is known to require an exponentially large number of group operations in a generic group (see [Sho97]), and there is currently no other algorithm than generic algorithms (*e.g.*, Pollard's rho) for well chosen elliptic curves.

Given the hardness of the DDH problem in practice, one standard computational assumption in cryptography is the DDH assumption, which can be formalized as follows. Let $\mathcal{G}$ be a process that, given a security parameter λ, generates a group G (typically, the order of G is exponential in λ). We suppose that this group and its elements can be efficiently represented (*i.e.* using a number of bits which is polynomial in λ) and that its composition law, the group inversion (*i.e.* computing the inverse g^{-1} of a group element g) and the random sampling are efficiently computable (*i.e.* with polynomial-time algorithms). For instance, $\mathcal{G}$ can output an elliptic curve, or a subgroup of the invertible elements of $\mathbb{Z}_n$. The DDH assumption on $\mathcal{G}$ states that, for all PPT adversary $\mathbb{A}$, $\mathbb{A}$ wins the DDH game (see Algorithm 5) with a negligible advantage. In the DDH game, $\mathbb{A}$ gets a tuple (g_1, g_2, g_3, g_4) which is either perfectly uniform ($b = 0$) or a random DDH tuple ($b = 1$). Given this tuple, $\mathbb{A}$ must deduce b.

Although the DDH problem is hard in general, it does not mean that it is hard for just any $\mathcal{G}$. For instance, suppose that for all λ, the order of $\mathcal{G}(\lambda)$ has a non-trivial factorization $n = pq$ with a polynomial p. Then, with probability $1/p$ the elements g_1 and g_3 of a DDH tuple will be of order q while this only happens with probability $1/p^2$ for a random tuple. In this case, computing g_1^q and g_3^q leads to a polynomial-time distinguisher that wins the DDH game with a non-negligible advantage. For this reason, we generally consider groups of prime order. The DDH assumption is the main computational assumption that we consider in this thesis.

Algorithm 5: DDH$(\mathcal{G}, \lambda, \mathbb{A})$

$G, q \longleftarrow \mathcal{G}(\lambda)$ (* q is the order of the group *);
$g_1 \xleftarrow{\$} G$;
$a, b, c \xleftarrow{\$} \mathbb{Z}_q$;
$g_2 \leftarrow g^a$; $g_3 \leftarrow g^b$; $g_4 \leftarrow g^c$;
$b \xleftarrow{\$} \{0, 1\}$;
if $b = 1$ **then** $g_4 \longleftarrow g^{ab}$;
$b' \longleftarrow \mathbb{A}(g_1, g_2, g_3, g_4)$;
if $b = b'$ **then return** 1 **else return** 0;

2.1.3 The random oracle model

Apart from the DDH assumption, we also use an abstract model known as the programmable random oracle model (ROM). A random oracle is an oracle $\mathcal{O}_{\mathsf{RO}}$ that, given a query m, returns a perfectly random bitstring of fixed size (typically, 2λ, where λ is the security parameter).

However, if the query m was made previously, the oracle gives the same output as before (see Algorithm 6). This allows to model the unpredictable nature of a hash function, whose outputs can only be known by actually computing the hash. In the ROM, we give the adversary an oracle access to $\mathcal{O}_{\mathsf{RO}}$ and, for any verification based on a result of a hash function, we use the output of the random oracle instead. For instance, consider the game illustrated in Fig. 6, where Bob tries to guess a random bit chosen by Alice. Bob does not trust Alice; hence, if he loses, he wants her to provide a proof that her bit is actually different from his guess. Consequently, Alice must first send a *commitment* on this bit, denoted h. For this purpose, she generates $\lambda - 1$ additional random bits and forms the bitstring m, which is hashed into h, and sends that to Bob. Once Bob receives Alice's commitment, he sends her his guess and Alice has to reveal m. Bob then checks that $h = \text{hash}(m)$. Finally, Bob wins if m begins with his guess (denoted $g \preceq m$); otherwise, he loses.

Algorithm 6: $\mathcal{O}_{\mathsf{RO}}$

Requires: λ, the security parameter
Variables: A hashmap H
Inputs: x, any bitstring

```
if x is a key of H then return the corresponding key H(x);
else
    h ←$ {0,1}^{2λ};
    Add the key x in H with the value h;
    return h;
```

To model this game in the ROM and make sure that Alice cannot cheat, we can use Algorithm 7. In this game, Alice is modeled by the adversary $\mathbb{A}$, but the hash function is replaced by the random oracle. For this purpose, we give to Alice an access to the oracle $\mathcal{O}_{\mathsf{RO}}$ and we check that h is the output of this oracle, when applied to m. In this model, it is possible to show that Alice cannot cheat, *i.e.* that she cannot win with a non-negligible advantage. For this purpose, we consider three possibilities.

Case 1. The first possibility is that, during her first activation, Alice did not make a query to $\mathcal{O}_{\mathsf{RO}}$ which was answered by h. We denote α_1 the probability of this. In this case, during the second activation, each query to $\mathcal{O}_{\mathsf{RO}}$ has a probability of at most $1/2^{2\lambda}$ to be answered with h, so the condition $\mathcal{O}_{\mathsf{RO}}(m) = h$ happens with probability $\varepsilon_{\mathsf{prei}} \leq q/2^{2\lambda}$, where q is the number of oracle queries. We denote β_1 the probability that Alice wins if $\mathcal{O}_{\mathsf{RO}}(m) = h$ in Case 1.

Case 2. The second possibility is that, during her first activation, Alice made at least two different queries $m_1 \neq m_2$ which were both answered with h. This is called a collision, which happens with probability $\alpha_2 \leq 1 - \prod_{i=0}^{q-1}(1 - i/2^{2\lambda})$. Since $q = o(2^{2\lambda})$, this probability is equal to approximately $\frac{q(q-1)}{2^{2\lambda+1}}$, which is why we consider that we need $q \approx 2^{\lambda}$ hash queries to obtain a collision: this is the birthday paradox. We denote β_2 the probability that Alice wins in Case 2.

Case 3. The last possibility is that, during her first activation, Alice made a single query m' which was answered with h. Then, since g is chosen at random, $g \preceq m'$ with probability $1/2$. Therefore, if Alice wants to win with some advantage, she must find a second preimage m of h, which happens with probability $\varepsilon_{\mathsf{snd\text{-}prei}} \leq q/2^{2\lambda}$. We denote α_3 the probability of Case 3, and β_3 the probability that Alice wins if she finds a second preimage in Case 3.

Conclusion. Now, the three cases are mutually exclusive and cover all the possibilities,

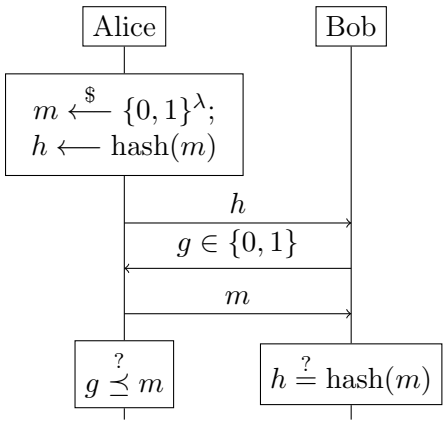


Figure 6: Guess coin game

Algorithm 7: $\mathrm{Bob}(\lambda, \mathbb{A})$

1 $h \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathsf{RO}}}(\lambda)$;
2 $g \xleftarrow{\$} \{0,1\}$; $b \xleftarrow{\$} \{0,1\}$;
3 $m \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathsf{RO}}}(g)$;
4 **if** $\mathcal{O}_{\mathsf{RO}}(m) \neq h$ **then return** b;
5 **if** $g \preceq m$ **then return** 0 **else return** 1;

therefore Alice's global probability to win is

$$p = \alpha_1 \left(\frac{1}{2} (1 - \varepsilon_{\mathsf{prei}}) + \beta_1 \varepsilon_{\mathsf{prei}} \right) + \alpha_2 \beta_2 + \alpha_3 \left(\frac{1}{2} (1 - \varepsilon_{\mathsf{snd\text{-}prei}}) + \beta_3 \varepsilon_{\mathsf{snd\text{-}prei}} \right).$$

Since $\alpha_1 + \alpha_2 + \alpha_3 = 1$, Alice advantage is

$$\begin{aligned} \left| p - \frac{1}{2} \right| &= \left| \alpha_1 \left(\beta_1 - \frac{1}{2} \right) \varepsilon_{\mathsf{prei}} + \alpha_2 \left(\beta_2 - \frac{1}{2} \right) + \alpha_3 \left(\beta_3 - \frac{1}{2} \right) \varepsilon_{\mathsf{snd\text{-}prei}} \right| \\ &\leq \frac{1}{2} \varepsilon_{\mathsf{prei}} + \frac{1}{2} \alpha_2 + \frac{1}{2} \varepsilon_{\mathsf{snd\text{-}prei}} \\ &\leq \frac{q}{2^{2\lambda}} + \frac{q^2}{2^{2\lambda+2}}. \end{aligned}$$

The above example shows that the ROM captures the three main properties of a hash function: first preimages, second preimages and collisions should be hard to find. Actually, the ROM is even stronger than that. Indeed, consider an adversary $\mathbb{A}$ in the ROM. Then we can construct an adversary $\mathbb{B}$ which interacts with $\mathbb{A}$ by simulating the random oracle (hence the programmable adjective). Indeed, since $\mathbb{A}$ can only make black-box queries to $\mathcal{O}_{\mathsf{RO}}$, this can be modeled as $\mathbb{A}$ (seen as a PPT) writing a bitstring query in a specific tape, and reading the answer in some other tape. Consequently, we can make the following assumptions:

- $\mathbb{B}$ can read the hash queries made by $\mathbb{A}$;
- therefore, $\mathbb{B}$ can recover the preimages of a hash output by $\mathbb{A}$;
- $\mathbb{B}$ can choose the output of the oracle, provided it is uniformly distributed for all the new queries, and remains the same if the query is made several times.

The last assumption allows $\mathbb{B}$ to give trapdoored hash values to $\mathbb{A}$. For instance, suppose that every bitstring encodes a public key. Then, instead of generating random bitstring as expected, $\mathbb{B}$ can generate a random secret key and deduce the corresponding public key. Therefore, $\mathbb{B}$ will know the secret keys associated to the hash values produced by $\mathbb{A}$. Compared to what is expected from a hash function, the ROM can be considered too strong. More precisely, since any hash function can be implemented by an efficient explicit algorithm, the notion of "random

oracle" cannot be instantiated. Worse, an uncomfortable consequence is that some schemes can be proven secure in the ROM while being completely insecure in the standard model (see for instance [Nie02, CGH04]). Consequently, some authors prefer to only rely on the properties of the hash function (*i.e.* collision resistance), and to use another assumption, known as the common reference string (CRS) instead of the ROM. Although the CRS assumption is more sensible than the ROM, the corresponding primitives are usually less efficient. Consequently, it is extremely common to use the ROM in electronic voting.

2.2 Encrypting a ballot to preserve privacy

In electronic voting, an encryption scheme is used to preserve the privacy of the voters. Since we want every voter to be able to encrypt their choice, while none should be able to decrypt the ballot of the others, we use a public key encryption scheme. Also, since we do not want a single authority to be able to recover the secret key and decrypt the ballots, we use a secret sharing scheme and rely on threshold cryptography.

2.2.1 Public key encryption

In public key cryptography, a single person has a secret key sk while a public key pk can be revealed to anyone. When it comes to encryption schemes, sk allows to decrypt and pk allows to encrypt. More formally, given a security parameter λ, let $\mathcal{S}(\lambda)$ be the set of the secret keys and $\mathcal{P}(\lambda)$ be the set of the public keys. Also, we denote $\mathscr{P}(\lambda)$ the set of the plaintexts, $\mathscr{R}(\lambda)$ the set of the randomness and $\mathscr{C}(\lambda)$ the set of the ciphertexts. A public key encryption scheme is a tuple $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ such that

- For all $\mathsf{pk} \in \mathcal{P}(\lambda)$, there exists $\mathsf{sk} \in \mathcal{S}(\lambda)$ such that, for all $(m, r) \in \mathscr{P}(\lambda) \times \mathscr{R}(\lambda)$, $\mathrm{Dec}_{\mathsf{sk}}(\mathrm{Enc}_{\mathsf{pk}}(m, r)) = m$. Such a pair $(\mathsf{pk}, \mathsf{sk})$ is called a *key pair*.
- $\mathrm{Gen}(\lambda)$ is a non-deterministic polynomial algorithm that outputs a random key pair $(\mathsf{pk}, \mathsf{sk})$.

The encryption algorithm allows to encrypt a plaintext message into a ciphertext, using some randomness. The idea is that a ciphertext should be indistinguishable from a random element of $\mathscr{C}$, which is captured by the IND-CPA property (given in Definition 10). This property stands for indistinguishability under chosen plaintext attacks. It means that for two chosen messages m_0, m_1, the adversary cannot tell an encryption of m_0 from an encryption of m_1.

Definition 10. *An encryption scheme* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ *is IND-CPA secure if, for all PPT adversary* $\mathbb{A}$*,* $\mathbb{A}$ *wins the IND-CPA game (defined in Algorithm 8) with a negligible probability.*

Algorithm 8: $\text{IND-CPA}(\lambda, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;
$m_0, m_1 \longleftarrow \mathbb{A}(\mathsf{pk})$;
$b \xleftarrow{\$} \{0, 1\}$;
$r \xleftarrow{\$} \mathscr{R}$;
$C \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(m_b, r)$;
$b' \longleftarrow \mathbb{A}(C)$;
if $b = b'$ **then return** 1 **else return** 0;

Algorithm 9: $\text{Exp}^{\mathsf{ind\text{-}cpa}}(\lambda, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;
$m \longleftarrow \mathbb{A}(\mathsf{pk})$;
$b \xleftarrow{\$} \{0, 1\}$;
$r \xleftarrow{\$} \mathscr{R}$;
$C_1 \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(m, r)$; $C_0 \xleftarrow{\$} \mathscr{C}$;
$b' \longleftarrow \mathbb{A}(C_b)$;
if $b = b'$ **then return** 1 **else return** 0;

To illustrate that the notion of IND-CPA security indeed captures that the ciphertexts are indistinguishable from random ciphertexts, we also give Algorithm 9 which is another formulation of the IND-CPA game. Suppose that there exists an adversary $\mathbb{A}$ which wins $\text{Exp}^{\mathsf{ind\text{-}cpa}}$ with some probability p. Then we construct an adversary $\mathbb{B}$ for the IND-CPA game as follows. First, $\mathbb{B}$ gets pk from the game and forwards it to $\mathbb{A}$ which answers with some m. Then $\mathbb{B}$ must output m_0, m_1 in the IND-CPA game. For this purpose, it samples m_0 at random and sets $m_1 = m$. It is given a ciphertext C which it forwards to $\mathbb{A}$. Finally, $\mathbb{B}$ plays $\mathbb{A}$'s output in the IND-CPA game. Clearly, $\mathbb{B}$ also wins with probability p. Conversely, suppose that there exists an adversary $\mathbb{A}$ which wins the IND-CPA game with some probability p. We construct an adversary $\mathbb{B}$ for $\text{Exp}^{\mathsf{ind\text{-}cpa}}$ as follows. Just as in the previous reduction, $\mathbb{B}$ gets pk from the game and forwards it to $\mathbb{A}$. However, $\mathbb{A}$ now returns two plaintexts m_0, m_1 and $\mathbb{B}$ can only play one of them in $\text{Exp}^{\mathsf{ind\text{-}cpa}}$. Consequently, $\mathbb{B}$ flips a coin $c \in \{0, 1\}$ and play m_c in $\text{Exp}^{\mathsf{ind\text{-}cpa}}$ to get an encryption C. $\mathbb{B}$ forwards C to $\mathbb{A}$ which answers with b'. If $b' = c$, $\mathbb{B}$ returns 1; otherwise, it returns 0. Now, suppose that $\mathbb{B}$ got a random encryption of m_c ($b = 1$ in $\text{Exp}^{\mathsf{ind\text{-}cpa}}$). Then $\mathbb{B}$ played a perfect simulation of the IND-CPA game to $\mathbb{A}$, and therefore wins with the same probability p as $\mathbb{A}$. On the other hand, if $\mathbb{B}$ got a random element $C \in \mathscr{C}$ ($b = 0$ in $\text{Exp}^{\mathsf{ind\text{-}cpa}}$), $\mathbb{A}$'s view (*i.e.* pk, C) is perfectly independent from c, so that $b' = c$ with probability $1/2$. Hence, $\mathbb{B}$ also wins with probability $1/2$. Now, since both situation are equiprobable, it follows that $\mathbb{B}$ wins with probability $1/2(p + 1/2)$, which gives the advantage $1/2|p - 1/2| = \varepsilon/2$, where ε is $\mathbb{A}$'s advantage in the IND-CPA game. This shows that both games are indeed equivalent.

The IND-CPA security is the minimal property that an encryption scheme must provide. However, it is not sufficient in electronic voting. Indeed, suppose that a public-key encryption scheme allows anyone, given an encryption C of some plaintext m, to forge an encryption $C' \neq C$ for a (possibly unknown) plaintext m' which is somehow related to m (for instance, $m' = m$). Such an encryption scheme would allow the adversary to launch a replay attack, which would break privacy (see Section 1.1.4).

A more suitable notion of security for electronic voting is the stronger notion of NM-CPA security, which stands for non-malleability under chosen plaintext attacks. Intuitively, this property means that, given an encryption $C^\star$ of m_1, one cannot produce an encryption $C' \neq C^\star$ whose plaintext m' is related to m_1. To model a chosen plaintext attack, in this context, it is usual to ask the adversary to choose the possibilities for the plaintext m_1, which are represented by the set M at line 2 of the NM-CPA game (see Algorithm 10). Alternatively, M can also be seen a polynomial-time sampling (probabilisic) algorithm. From this set, the challenger randomly samples an element m_1 for which it produces an encryption $C^\star$. Then, to capture the fact that the relationship between m_1 and m' could be any efficiently decidable relation $\mathcal{R}$, we let the adversary choose $\mathcal{R}$ at line 6. It must also output a list of ciphertexts $\boldsymbol{C}$ which contains a ciphertext $C' \neq C^\star$ such that $\mathcal{R}(m_1, m')$ is more likely to be verified than $\mathcal{R}(m_0, m')$, where $m' = \text{Dec}_{\mathsf{sk}}(C')$ while m_0 is a random element in M. This gives Definition 11, which is adapted from [BDPR98].

Definition 11. *An encryption scheme* $(\text{Gen}, \text{Enc}, \text{Dec})$ *is NM-CPA secure if, for all PPT adversary* $\mathbb{A}$, $\mathbb{A}$ *wins the NM-CPA game (defined in Algorithm 10) with a negligible advantage.*

A related but not equivalent security notion is the indistinguishability under adaptive chosen ciphertext attacks (IND-CCA), which can be formalized in the IND-CCA game (see Algorithm 11). In this game, the adversary has access to a decryption oracle $\mathcal{O}_{\mathsf{Dec}}$ when it chooses the plaintexts, and can also make queries to a decryption oracle $\mathcal{O}_{\mathsf{Dec}}{}^\star$ when making its guess (however, $\mathcal{O}_{\mathsf{Dec}}{}^\star$ cannot decrypt the challenge ciphertext $C^\star$). In [BDPR98], IND-CCA is proven to be

Algorithm 10: $\mathrm{Exp}^{\mathsf{nm\text{-}cpa}}(\lambda, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;
$M \longleftarrow \mathbb{A}(\mathsf{pk})$;
$m_0, m_1 \xleftarrow{\$} M$;
$r \xleftarrow{\$} \mathscr{R}$;
$C^\star \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(m_1, r)$;
$\mathcal{R}, \boldsymbol{C} \longleftarrow \mathbb{A}(C^\star)$;
$\boldsymbol{m} \longleftarrow (\mathrm{Dec}_{\mathsf{sk}}(C))_{C \in \boldsymbol{C} \setminus C^\star}$;
$b \xleftarrow{\$} \{0,1\}$;
if $\exists m \in \boldsymbol{m} \mid \mathcal{R}(m_b, m)$ **then return** 1
else return 0;

Algorithm 11: $\mathrm{Exp}^{\mathsf{ind\text{-}cca}}(\lambda, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;
$m_0, m_1 \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathsf{Dec}}}(\mathsf{pk})$;
$b \xleftarrow{\$} \{0,1\}$;
$r \xleftarrow{\$} \mathscr{R}$;
$C^\star \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(m_b, r)$;
$b' \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathsf{Dec}}{}^\star}(C^\star)$;
if $b = b'$ **then return** 1
else return 0;

strictly stronger than NM-CPA. Therefore, the IND-CCA security is also suitable for electronic voting, although NM-CPA is enough.

2.2.2 The ElGamal encryption scheme

One very popular encryption scheme is the ElGamal cryptosystem. To present it, we consider that the process $\mathscr{G}$, which generates a group description from a security parameter, is fixed. If G is a group which can be output by $\mathscr{G}(\lambda)$ for some λ, we use the notation $G \in \mathscr{G}$. In this section, for all $G \in \mathscr{G}$, we consider that G is a cyclic group of (publicly) known prime order q, which is at least exponential in λ. For instance, $q \approx 2^{2\lambda}$ in the case of elliptic curves. With that in mind, the plaintext space is $\mathscr{P} = G$, the randomness space is $\mathscr{R} = \mathbb{Z}_q$ and the ciphertext space is $\mathscr{C} = G \times G$. Then, the three algorithms of the ElGamal encryption scheme are defined as follows, where $G = \mathscr{G}(\lambda)$.

$\mathbf{Gen}(\boldsymbol{\lambda})$ picks a random group generator $g \in G \setminus \{1\}$ and a random secret key $\mathsf{sk} \in \mathbb{Z}_q$. Then, the public encryption key is given by the pair $\mathsf{pk} = (g, g^{\mathsf{sk}})$.

$\mathbf{Enc}_{\mathsf{pk}}(\boldsymbol{m}, \boldsymbol{r})$: to encrypt a plaintext $m \in G$ with the public key $\mathsf{pk} = (g, h)$, we pick a random $r \in \mathbb{Z}_q$ and we compute $(x, y) = (g^r, mh^r)$. The pair (x, y) is the ciphertext $\mathrm{Enc}_{\mathsf{pk}}(m, r)$.

$\mathbf{Dec}_{\mathsf{sk}}(\boldsymbol{C})$: to decrypt a ciphertext $C = (x, y)$, we return $m = yx^{-\mathsf{sk}}$.

The ElGamal encryption scheme is known to provide IND-CPA security under the DDH assumption on $\mathscr{G}$. Indeed, suppose that there exists an adversary $\mathbb{A}$ which wins the IND-CPA game with probability p. We construct an adversary $\mathbb{B}$ for the DDH game as follows. First, $\mathbb{B}$ receives a challenge tuple $(g_1, g_2, g_3, g_4) \in G$. If $g_1 = 1$ (this happens with probability $1/q$), $\mathbb{B}$ can easily decide whether (g_1, g_2, g_3, g_4) is a DDH tuple or not, so that it wins with probability 1. Otherwise, (g_1, g_2) has the same distribution as an output of Gen, therefore $\mathbb{B}$ can forward this to $\mathbb{A}$ as the pair pk. The latter answers with some $m_0, m_1 \in G$. $\mathbb{B}$ flips a coin $c \in \{0, 1\}$ at random and compute the ciphertext $C = (g_3, m_c g_4)$ which it gives to $\mathbb{A}$. Finally, if $\mathbb{A}$ outputs c, $\mathbb{B}$ outputs 1; otherwise, $\mathbb{B}$ outputs 0. Now, remark that when the challenge is a random tuple, $\mathbb{A}$'s view (*i.e.* $(g_1, g_2, g_3, m_c g_4)$) is perfectly uniform in G^4 and therefore independent from c. Therefore, $\mathbb{A}$ outputs c with probability $1/2$ and $\mathbb{B}$ wins with probability $1/2$. On the other hand, if the challenge is a DDH tuple, $(g_3, m_c g_4)$ follows the same distribution as an ElGamal encryption of m_c with the public key (g_1, g_2), therefore $\mathbb{B}$ played a perfect simulation of the IND-CPA game to $\mathbb{A}$ and wins with probability p. Overall, when $g_1 \neq 1$, $\mathbb{B}$ wins with probability $1/2(p + 1/2)$.

Since $g_1 = 1$ with probability $1/q$, $\mathbb{B}$'s probability to win the DDH game is

$$\frac{1}{q} + \frac{1}{2}\left(1 - \frac{1}{q}\right)\left(p + \frac{1}{2}\right) = \frac{1}{2}\left(p + \frac{1}{2}\right) + \frac{3 - 2p}{4q} = \frac{1}{2} + \frac{1}{2}\left(p - \frac{1}{2}\right) + \frac{3 - 2p}{4q}.$$

Since q is exponential in λ while $p \in [0, 1]$, this means that $\mathbb{B}$'s advantage in the DDH game is about half $\mathbb{A}$'s advantage in the IND-CPA game. Conversely, if there exists an adversary $\mathbb{A}$ for the DDH game, we can easily construct an adversary which wins $\mathrm{Exp}^{\mathsf{ind\text{-}cpa}}$ with the same probability. Therefore, the IND-CPA security of the ElGamal encryption scheme is equivalent to the DDH assumption on $\mathscr{G}$. Now, we mention that, in general, this equivalence requires that the generator g from the public key $\mathsf{pk} = (g, h)$ is chosen at random: we explicitly used this in the security reduction. For instance, assume that there is a non-trivial subgroup $H \subset G$ for which there is an efficient membership test, and that g is chosen in H. Then an encryption (x, y) of $m \in H$ will always have $y \in H$ while an encryption of $m \notin H$ will never have $y \notin H$. In this case, the encryption scheme is not IND-CPA, even if the DDH problem is hard in G. In practice, it is common to use a fixed g for convenience, since a group description usually comes with a fixed generator. In this case, an alternative definition of DDH is used, where g_1 is the fixed generator output by $\mathscr{G}$ instead of a random group element. In theory, this is a strictly stronger assumption, even when the order of the group is a known prime number (see [BMZ19] for an analysis in the generic group model). In practice, although using a random generator may protect against intensive precomputations, using a fixed generator is the norm and is not considered less secure.

Homomorphic property. Apart from its strong semantic security, the ElGamal cryptosystem has another interesting property: it actually realizes a group isomorphism from $G \times \mathbb{Z}_q$ to $G \times G$. Consequently, for all public key pk, for all plaintext messages $m_1, m_2 \in G$ and for all randomness $r_1, r_2 \in \mathbb{Z}_q$, we have

$$\mathrm{Enc}_{\mathsf{pk}}(m_1, r_1)\mathrm{Enc}_{\mathsf{pk}}(m_2, r_2) = \mathrm{Enc}_{\mathsf{pk}}(m_1 m_2, r_1 + r_2).$$

This property is extremely useful in general, all the more so in electronic voting. Indeed, this allows to use a strategy known as *homomorphic tally* (as seen in Section 1.1.3 when we presented Helios). Suppose that there are two possible voting options, say yes or no. Then, given a fixed generator a (typically, $a = g$ or $a = h$), a yes vote can be encoded as the group element a while a no vote is encoded into 1. Then, after that the voters submitted the encryptions $C_1, \cdots, C_n$, it is possible to decrypt the product $C_1 \cdots C_n$ to recover a group element b. Finally, the discrete logarithm of b in base a is equal to the number of voters who voted yes. Solving the discrete logarithm problem for small n, even $n \approx 10^{10}$, can be done within a second using the baby-step, giant-step algorithm [Sha71]; therefore, it is not a problem here.

More generally, given a generator a, the *exponential* ElGamal encryption of an integer n is the ElGamal encryption of a^n. This gives an additively homomorphic encryption scheme, which is useful in electronic voting. Its drawback is that n must be in a reasonably small interval for the decryption to be possible; otherwise, solving the discrete logarithm problem becomes too hard.

A direct consequence of the homomorphic property of the ElGamal encryption scheme is that it is not NM-CPA secure. Indeed, given an encryption C of some (possibly unknown) plaintext m, one can create another encryption $C' \neq C$ of m by multiplying C by any encryption of 1. This is called reencryption (or rerandomization). Although the possibility to reencrypt is interesting, it means that the ElGamal encryption scheme should never be used as it is in electronic voting: it is not sufficient to provide privacy. In general, the voters also have to provide a *proof of*

knowledge, which is a ZKP that they can open their ballot (see Section 2.3); a classical result from [BPW12] shows that this provides NM-CPA security.

2.2.3 Threshold cryptography

In a public key encryption scheme, a single person holds the secret decryption key and is the only one who is able to decrypt ciphertexts. However, in electronic voting, we try as much as possible not to trust a single entity for a specific purpose. If an entity (*e.g.*, the public board) is trusted, it actually means that its behavior can be enforced in practice (*e.g.*, using consensus algorithms). Yet, it is not possible to prevent someone to decrypt publicly available ciphertexts if this person holds the secret decryption key. Therefore, we use a strategy known as *secret sharing*, which was introduced by Shamir.

Distributed key generation. In a distributed key generation (DKG) protocol, we consider a group of n_T participants and a threshold t. We want to share a secret $\mathsf{sk} \in \mathbb{Z}_q$ for some prime number q, such that if t or less participants collude, they have no information about sk. However, if $t+1$ or more participants collaborate, they can recover the secret key. To achieve this with any $t < n_T$, we assume ideally authenticated private channels (*i.e.* no participant can send a message in the name of another, and any two participants can exchange private messages). Such channels can be obtained by the mean of symmetric cryptography, after that every pair of participants performed a key exchange, *e.g.*, using the classical sign-and-mac paradigm [Kra03]. This assumes that the participants are well identified, for instance by a public key.

A popular DKG protocol for the discrete logarithm setting is due to Pedersen [Ped91a], that is illustrated in Fig. 7. In this protocol, the participants collectively choose a random polynomial $f = \sum_{j=1}^{n_T} f_j$ of degree t, by each choosing a random secret polynomial f_j. Once f is (implicitly) chosen, the secret key is $\mathsf{sk} = f(0)$, and every participant j has a share $s_j = f(j)$ of this secret. Hence, using Lagrange polynomials, $t+1$ or more participants can recover the secret; however, t or less shares contain no Shannon information about sk.

Note that in general, Step 1 and Step 2 are presented as a single step where the participants broadcast their commitments $c_{i,k}$. However, this allows the adversary to rig the distribution of the public encryption key, as mentioned in [GJKR99]. Indeed, suppose – to begin with – that the threshold t and the number of participants n_T are such that $t \geq n/2$. Suppose that, by corrupting up to t participants, the adversary produces a situation where there is at most t honest participants. Then it can proceeds as follows: first, choose the secret key sk; wait for the other commitments to be published and, for a specific corrupted participant i, choose $c_{i,0} = g^{\mathsf{sk}} / \prod_{j \neq i} c_{j,0}$; afterwards, for all honest participant j, choose $s_{i,j} \in \mathbb{Z}_q$ at random and, using Lagrange interpolation, compute the $(c_{i,k})_{k=0}^{t}$ so that $g^{s_{i,j}} = \prod_{k=0}^{t} c_{i,j}^{j^k}$ for all honest j. This way, the adversary is undetectably able to *choose* the secret key, which is clearly undesirable.

The standard way to prevent the above attack is to ask for the participants to provide a PoK (see Section 2.3.2) for their secret share s_i, and to abort if they are unable to do so. This solution is preferable in practice because it requires less communications than using a round of synchronization as suggested in Fig. 7. However, as mentioned in [GJKR99], it is still possible for the adversary to (honestly) choose its polynomial *after* the other commitments are revealed, compute the resulting pk and try over and over again until some specific pattern (*i.e.* the first 23 bits are 0) is met, hence preventing the public key to be uniformly random as required for the security of the scheme. For this reason, we prefer to present this version of the DKG, in which the adversary cannot temper with the distribution of the key without being detected and blamed. Another difference compared to the original Pedersen's DKG is that it does not provide fairness: even if there are $t+1$ honest participants, the adversary can force the protocol to abort.

1. Participant P_i chooses a random polynomial $f_i = \sum_{k=0}^{t} \alpha_{i,k} X^k \in \mathbb{Z}_q[X]$ and computes the commitments $c_{i,k} = g^{\alpha_{i,k}}$ for all k. Then P_i computes an aggregated commitment $b_i = \text{hash}(c_{i,0}|| \cdots ||c_{i,t})$. Finally, P_i broadcasts b_i.

2. Once all the participants have broadcast their b_j, P_i broadcast $c_{i,0}, \cdots, c_{i,t}$. It checks that the commitments of the other participants are consistent with their aggregated commitment, *i.e.* that $b_j = \text{hash}(c_{j,0}|| \cdots ||c_{j,t})$ for all j. If this is not the case for some j (including i), the protocol aborts and j is blamed. Otherwise, for all j, P_i computes $s_{i,j} = f_i(j)$ and privately sends this to P_j.

3. Upon receiving a share $s_{j,i}$, P_i verify that $g^{s_{j,i}} = \prod_{k=0}^{t} c_{j,k}^{i^k}$. If this is not the case, P_i broadcasts a complain against P_j. The secret share of P_i is defined as $s_i = \sum_{j=0}^{n_T} s_{j,i}$.

4. If some P_j broadcasts a complain against P_i, P_i broadcasts $s_{i,j}$. If $g^{s_{i,j}} \neq \prod_{k=0}^{t} c_{i,k}^{j^k}$, the protocol aborts and P_i is blamed.

5. The public key is defined as $\mathsf{pk} = (g, h)$, where $h = \prod_{j=0}^{n_T} c_{j,0}$. The secret key is defined implicitly as $\mathsf{sk} = \log_g(h)$. The public commitment of P_i can be computed as $h_i = \prod_{j=0}^{n_T} \prod_{k=0}^{t} c_{j,k}^{i^k}$. Finally, the transcript can be $\mathsf{pk}, (h_i)_{i=1}^{n_T}$, signed by all the participants.

Figure 7: Pedersen's DKG (with abort), with a fixed generator g

However, we still have accountability: if the protocol aborts, at least one malicious participant will be blamed. Therefore, this is considered acceptable in the context of electronic voting; all the more so as the DKG takes place ahead of time, when there is no real incentive to produce a result right away.

Threshold decryption. One very interesting property of Shamir's secret sharing scheme is that it allows the participants to implicitly use the secret key without ever explicitly recovering it. This gives threshold cryptography, which consists of doing the polynomial interpolation "in the exponent". In electronic voting, the most commonly used protocol is that of *threshold decryption*, which allows the participants to collectively decrypt an ElGamal cyphertext (x, y) without learning anything about each other's shares, nor about the secret key. The idea is that each participant P_i broadcasts a *partial decryption* $w_i = x^{s_i}$, where s_i is P_i's share. To ensure that the corrupted participants cannot cause the protocol to output an incorrect decryption, each participant must also give a ZKP that w_i is well-formed (with respect to the public commitment $h_i = g^{s_i}$ of their share). Then, given a subset of size $t+1$ of participants whose ZKP is valid, x^{sk} is recovered using Lagrange polynomials. Similar protocols are available for threshold signature.

When a threshold decryption is available, we can define a threshold encryption scheme $(\mathsf{Setup}, \text{Enc}, \text{PartDec}, \text{Dec})$, where Setup is the algorithm obtained by running the DKG with honest participants; Enc is the usual encryption algorithm; PartDec is an algorithm which takes a ciphertext and a secret share and outputs the partial decryption of the ciphertext; and Dec is an algorithm which allows to recombine the partial decryptions in order to recover the plaintext. As stated above, a threshold decryption protocol does not explicitly recover the secret key. Nevertheless, other information (such as the partial decryptions w_i and the ZKP) are revealed beside the result of the decryption. Therefore, it is important to assess whether this can be exploited by a PPT adversary or not, for instance to recover a share given sufficiently many partial

decryptions. First, the ZKP are discussed in Section 2.3 and reveal no information except that the partial decryptions are well-formed. However, given the public commitment h_i of P_i's share, there is a single possibility for w_i so that the well-formed partial decryptions actually contain some information about the shares. More precisely, suppose that the adversary produces the ciphertext $(x, y) = (g^r, h^r)$ with some random $r \in \mathbb{Z}_q$. Then the partial decryption of (x, y) by participant P_i is $w_i = h_i^r$, so that the adversary is able to distinguish w_i from any other group element. Nonetheless, the relationship between w_i and x is the same as the one between h_i and g (*i.e.* $\log_g(h_i) = \log_x(w_i) = s_i$), therefore the intuition is that w_i does not contain more information than what was already contained in h_i.

In general, to capture that *something* (*e.g.*, w_i) does not contain any useful information for the adversary, we use the simulation paradigm: we construct a simulator that the adversary can use to produce w_i itself. Then, we argue that no adversary can spot the difference between the real w_i and the simulated one. The fact that the partial decryption in the ElGamal setting can be simulated is a folklore result. However, we could not find a proper formulation of this, let alone a proof. Since this is a folklore result, it must be stated (and proved) properly *somewhere*, for instance in a cryptography course book. Nevertheless, the fact that we could not find it fast enough means that it does not hurt to state it properly again in this thesis. In addition, there are three conditions for the folklore simulator to succeed, listed below. If one condition is not met, then the folklore simulator will fail and another one must be used (if any). In some cases, this could represent a gap in the security proof, which may be an issue.

- The simulator must have access to the secret shares of the corrupted participants;
- The adversary must not know the secret key nor the share of a single honest participant;
- The ciphertext to decrypt must be honestly generated, and not chosen by the adversary.

To capture those conditions, we propose the notion of ZK-TCPA security, which stands for zero knowledge of the threshold decryption protocol under chosen ciphertext attack (see Definition 12). Intuitively, the ZK-TCPA security means that the threshold decryption protocol does not give any additional information compared to the result of the decryption algorithm, since the partial decryptions can be simulated.

Definition 12. *We say that a threshold encryption scheme* $(\mathsf{Setup}, \mathrm{Enc}, \mathrm{PartDec}, \mathrm{Dec})$ *is ZK-TCPA if there exists a simulator* Sim *such that, for all PPT adversary* $\mathbb{A}$*,* $\mathbb{A}$ *wins the ZK-TCPA game (defined in Algorithm 13) with a negligible advantage.*

Definition 12 is based on the ZK-TCPA game, defined in Algorithm 13. In this game, the DKG is ra honestly but the adversary can corrupt up to t talliers afterwards, and hence learn their respective shares (recall that $t + 1$ shares are necessary to recover the secret key). Then the adversary chooses an arbitrary plaintext m which is honestly encrypted into a ciphertext C. Finally, the adversary is either given the partial decryptions of the honest participants, or a simulation of them: it must decide which.

In Theorem 1, we claim that the ElGamal threshold encryption scheme is ZK-TCPA under the DDH assumption. For this purpose, we exhibit an explicit simulator in Algorithm 12: given the ciphertext (x, y), its decryption m, the set A of the corrupted participants and their corresponding shares $(s_i)_{i\in A}$, the adversary can compute $\mathrm{Sim}_{n_T,t}((x, y), m, A, (s_i)_{i\in A})$ to simulate the partial decryption w_i of each honest participant. In this algorithm, we denote $\mathrm{Complete}(A, n_T)$ the process which returns the set of the $t + 1 - |A|$ first elements of $[0, n_T]\backslash A$ (this always includes 0).

Algorithm 12: $\mathrm{Sim}_{n_T,t}((x,y),m,A,(s_i)_{i\in A})$

Requires: $A \subset [1,n_T]$ has size $|A| \leq t$

$S \longleftarrow A \bigcup \mathrm{Complete}(A, n_T)$;
for $(i,j) \in ([1,n_T]\backslash A) \times S$ **do**
  $\Lambda_{i,j}^S \longleftarrow \prod_{k\in S\backslash\{j\}} \frac{i-k}{j-k}$;
$w_0 \longleftarrow y/m$;
for $i \in A$ **do** $w_i \longleftarrow x^{s_i}$;
for $i \in S\backslash(A\bigcup\{0\})$ **do** $w_i \xleftarrow{\$} G$;
for $i \in [1,n_T]\backslash S$ **do** $w_i \longleftarrow \prod_{j\in S} w_j^{\Lambda_{i,j}^S}$;
return $(w_i)_{i\in[1,n_T]\backslash A}$;

Algorithm 13: $\mathrm{Exp}^{\mathsf{ZK\text{-}TCPA}}(\lambda,\mathbb{A})$

$\mathsf{pk},\mathsf{sk},(h_i,s_i)_{i=1}^{n_T},\Pi \leftarrow \mathsf{Setup}(\lambda,n_T,t)$;
$A \leftarrow \mathbb{A}(\mathsf{pk},(h_i)_{i=1}^{n_T})$;
$b \xleftarrow{\$} \{0,1\}$;
if $|A| > t$ **or** $A \not\subset [1,n_T]$ **then return** b;
$m \longleftarrow \mathbb{A}((s_i)_{i\in A})$;
$r \xleftarrow{\$} \mathscr{R}$;
$C \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(m,r)$;
$S_0 \longleftarrow \mathrm{Sim}_{n_T,t}(C,m,A,(s_i)_{i\in A})$;
$S_1 \longleftarrow (\mathrm{PartDec}(C,s_i))_{i\notin A}$;
$b' \longleftarrow \mathbb{A}(C,S_b)$;
if $b' = b$ **then return** 1 **else return** 0;

Theorem 1. *The threshold ElGamal encryption scheme is ZK-TCPA in the ROM and under the DDH assumption.*

The key idea is that there exists a polynomial f of degree t such that, for all participant i, the secret key is $s_i = f(i)$, and the corresponding partial decryption is $w_i = x^{s_i}$, where (x,y) is the ciphertext to decrypt. Since the simulator is given the shares of the corrupted participants, it can deduce the corresponding partial decryption. In addition, as the simulator is given the plaintext $m = yx^{-\mathsf{sk}}$, it can deduce $w_0 = x^{\mathsf{sk}} = x^{f(0)}$. This gives the simulator at most $t+1$ constraints to respect. Hence, using Lagrange interpolation, it can produce a fake partial decryption $w_j = x^{g(j)}$ for all j, where g is a random polynomial of degree t such that $g(i) = f(i)$ for all corrupted i and for $i = 0$. Note that this only produces a computationally indistinguishable tuple, and not the exact partial decryptions. However, this is sufficient since we consider PPT adversaries.

The proof of this theorem uses a strategy called *game hops*, which is introduced in Section 3.1.1. Also, it is actually not trivial. Therefore, we prefer to present it in Appendix A rather than here. We already used the notion of ZK-TCPA elsewhere, so that the proof can also be found in [CGY21, Lemma I.1], which is the full version of [CGY22a].

2.2.4 The Paillier encryption scheme

In electronic voting, the ElGamal encryption scheme is very popular because of its homomorphic property and the possibility to distribute the secret key with Pedersen's DKG. However, it is not additively homomorphic so that an exponential version must be used, where not every ciphertext can be decrypted. An alternative to this is given in [Pai99], which provides an additively homomorphic encryption scheme. Although Paillier's encryption scheme is not often used in electronic voting, we compare some of our contributions to preexisting works which rely on it. For this reason, we present the Paillier cryptosystem in this section.

In the Paillier cryptosystem, n is a strong RSA modulus, so that n is coprime with its Euler's totient $\phi(n)$, which is the cardinality of $\mathbb{Z}_n^\times$. The plaintext space is $\mathscr{P} = \mathbb{Z}_n$, the randomness space is $\mathscr{R} = \mathbb{Z}_n^\times$ and the ciphertext space is $\mathscr{C} = \mathbb{Z}_{n^2}^\times$. The public key is $\mathsf{pk} = n$, and since n is coprime with $\phi(n)$, there exists an element $\mathsf{sk} \in \mathbb{Z}$ which is congruent to 1 modulo n and to 0 modulo $\phi(n)$. This element can be seen as an element of $\mathbb{Z}_n \times \mathbb{Z}_{\phi(n)}$, and hence can be represented as an integer in the range $[0, n^2]$.

Key generation. Pick two random safe primes p and q and compute $\mathsf{pk} = n = pq$ and

$\phi(n) = (p-1)(q-1)$. Then, use the extended Euclid algorithm to find a Bézout relation $un + v\phi(n) = 1$. Finally, set $\mathsf{sk} = v\phi(n)$ modulo $n\phi(n)$.

Encryption. To encrypt a message $m \in \mathbb{Z}_n$, one picks a random $r \in \mathbb{Z}_n^\times$ and computes $(1+n)^m r^n$ modulo n^2. Since $(1+n)$ is an element of order n in $\mathbb{Z}_{n^2}^\times$, $(1+n)^m$ is well defined when $m \in \mathbb{Z}_n$. Readily, we can remark that this defines a group isomorphism: for all $m_1, m_2 \in \mathbb{Z}_n$ and $r_1, r_2 \in \mathbb{Z}_n^\times$, $\mathrm{Enc}_{\mathsf{pk}}(m_1, r_1)\mathrm{Enc}_{\mathsf{pk}}(m_2, r_2) = \mathrm{Enc}_{\mathsf{pk}}(m_1 + m_2, r_1 r_2)$.

Decryption. To decrypt a cyphertext $C \in \mathbb{Z}_{n^2}^\times$, we first compute C^{sk} which is cast into an integer in $[0, n^2 - 1]$. Then, we deduce $m = (C^{\mathsf{sk}} - 1)/n$.

The security of the Paillier's encryption scheme relies on the DCRA assumption, which stands for decisionnal composite residuosity assumption. It states that a nth degree residue modulo n^2 is indistinguishable from a random element. More precisely, the DCRA assumption states that no PPT adversary can win the DCRA game (defined in Algorithm 14) with a non-negligible advantage. Compared to the DDH problem, the DCRA problem is less standard. However, there is currently no better method to attack it than to factor n, so that it is still considered hard in computer science. Consequently, the size of the key n can be chosen following the same recommendations as for factorization; see for instance [Key]. It can be shown that the Paillier's encryption scheme is IND-CPA under the DCRA assumption.

Algorithm 14: DCRA$(\lambda, \mathbb{A})$

Sample two random safe primes p and q of size $\mathsf{keysize}(\lambda)$;
$n \longleftarrow pq$;
$b \xleftarrow{\$} \{0,1\}$;
$C_0 \xleftarrow{\$} \mathbb{Z}_{n^2}^\times$;
$C_1 \longleftarrow C_0^n$;
$b' \longleftarrow \mathbb{A}(n, C_b)$;
if $b = b'$ **then return** 1 **else return** 0;

Threshold cryptography. Since the secret key sk lies within a group of unknown order, Shamir's secret sharing scheme cannot be readily used. There exists two concurrent strategies to distribute a Paillier secret key. The first one was presented in [FPS00] and an example of a DKG protocol which relies on this technique can be found in [NS10]. A similar but slightly different approach is presented in [DJN10]; an example of a DKG which relies on this can be found in [HMRT12]. Both strategies rely on the fact that the denominator in the Lagrange coefficients are coprime with n, so that it is still somehow possible to perform a polynomial interpolation. In any case, the corresponding DKG protocols are way more complex and computationally involved than Pedersen's DKG. In addition, they are less generic: the solution from [NS10] requires a honest majority, which is restrictive and undesirable in electronic voting; the solution from [HMRT12] is best suited for the two-party setting. In the resulting threshold encryption schemes, the participants have a share s_i and a public commitment $h_i = v^{s_i}$ to this share, where v is a generator of the group of the invertible squares modulo n^2 (*e.g.*, $v = 4$). Just as for an ElGamal ciphertext, the threshold decryption of a Paillier ciphertext C requires the authorities to reveal C^{s_i} as well as a ZKP of wellformedness. Intuitively, the ZK-TCPA security of the schemes would be obtained using the same arguments as for the ElGamal encryption scheme, except that we need to make the DDH assumption on $\mathbb{Z}_{n^2}^\times$ in addition to the DCRA assumption. We do not prove this because we do not use Paillier's encryption in this thesis.

2.3 Zero Knowledge Proofs in electronic voting

Zero knowledge proofs are the usual strategy that allows to reconcile privacy and verifiability in cryptography. In this section, we introduce the notion of zero knowledge proof and present a generic and standard method to obtain non-interactive zero knowledge proofs. More specific examples are given in Section 2.4.

2.3.1 Introduction to Zero Knwoledge Proofs

A NP language is a subset $\mathscr{L} \subset \{0,1\}^*$ for which there exists an efficiently decidable relation $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$ and a polynomial P such that, for all $z \in \{0,1\}^*$, $z \in \mathscr{L}$ if and only if there exists a witness $w \in \{0,1\}^*$ of size at most $P(|z|)$ such that $\mathcal{R}(w,z)$ is true (which is often denoted $w\mathcal{R}z$). The relation $\mathcal{R}$ can also be seen as a *non-deterministic* polynomial decisional algorithm: when only $z \in \{0,1\}^*$ is given, $\mathcal{R}$ does not help a polynomial adversary to decide whether $z \in \mathcal{L}$; however, if w is also given, deciding whether $w\mathcal{R}z$ is polynomial. In general, a NP language is not efficiently decidable. For instance, suppose that Alice has the secret key sk of an ElGamal public key $\mathsf{pk} = (g,h)$. Given a pair $(m,C) \in G \times G^2$, $C = (x,y)$ is an encryption of m if and only if $m = yx^{-\mathsf{sk}}$. In this example, $\mathscr{L}$ is the set of the pairs (m,C) such that C is an encryption of m, the witness is sk and $\mathcal{R}(\mathsf{sk},(m,C))$ can be efficiently decided by checking whether $m = yx^{-\mathsf{sk}}$. However, given a pair (m,C), deciding whether C is an encryption of m is hard and requires to solve an instance of the DDH problem.

Now, in the context of electronic voting, consider a voting protocol where a homomorphic tally is used: the result r of the election is obtained by decrypting a public ciphertext C. Then it is hard to decide whether a given result r is actually correct, *i.e.* that C is indeed an encryption of r. Of course, Alice could convince Bob by revealing her witness sk, but this would allow him to decrypt the other ballots individually, hence compromising privacy. In such a situation, we use a zero knowledge proof (ZKP), which is often derived from a Σ-protocol.

A Σ-protocol is an interactive protocol during which a prover (Alice) can convince a verifier (Bob) that they hold a witness w such that $w\mathcal{R}z$, where z is a public element that they both agree on (in our example, z is the pair (m,C) and m is given by Alice while C is a public ciphertext). During such a protocol, three messages are exchanged before a final verification step. First, the **commitment**: Alice sends some commitment c to Bob. Typically, c is chosen at random and does not contain any information. Then, the **challenge**: Bob chooses a random challenge d with enough entropy and sends it to Alice. Third, the **answer**: Alice answers the challenge with some a. Finally, the **verification**: given a transcript c,d,a, Bob can efficiently decide whether the transcript is valid or not. If this is the case, Bob accepts the transcript and is convinced that $z \in \mathscr{L}$. Otherwise, Bob rejects the transcript. A Σ-protocol typically provides two interesting properties:

- **Correctness.** If Alice follows the protocol (and actually knows w such that $w\mathcal{R}z$), Bob accepts the transcript.
- **Zero knowledge.** There exists a simulator which, given z and the challenge d, generates a transcript c,d,a which is perfectly indistinguishable from the transcript of the protocol.

Intuitively, the zero knowledge property means that Bob cannot learn anything by interacting with the prover, since he might as well generate the transcript himself. This is the simulation paradigm. However, we also want the Σ-protocol to actually *prove* something. For this reason, we consider two additional properties:

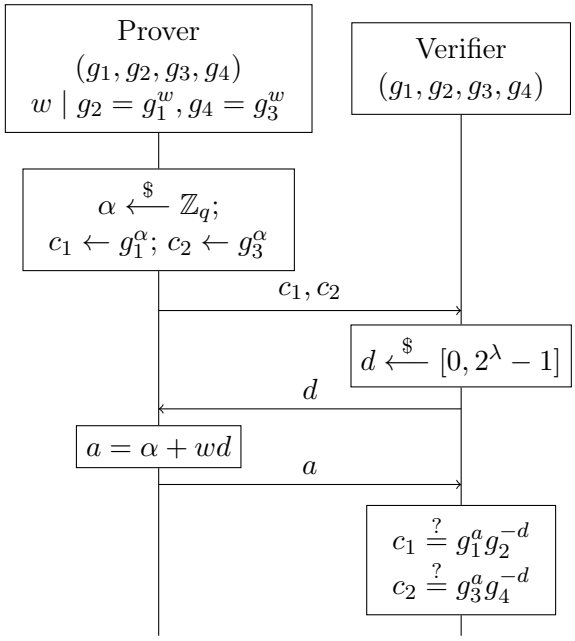


Figure 8: Proof of equality of discrete log

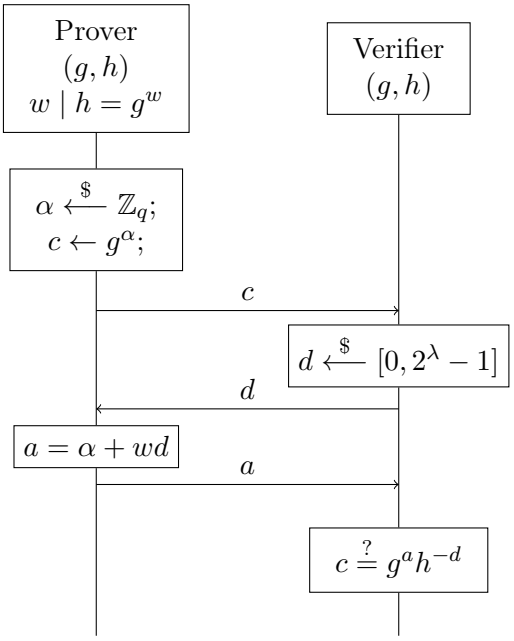


Figure 9: Proof of knowledge of discrete log

- **Computational soundness.** If $z \notin \mathscr{L}$, then Bob rejects with overwhelming probability.

- **Special soundness.** Given any two valid transcripts (c, d_1, a_1) and (c, d_2, a_2) which share the same commitment but have two different challenges, anyone can extract a witness w such that $w\mathcal{R}z$ in polynomial time.

The computational soundness property means that the prover cannot convince the verifier that a false statement is true. However, as this is the case for the example of the discrete logarithm relation (which is bijective), the soundness property can be "empty" as $\mathscr{L}$ may be too large. Therefore, it is sometimes required that the prover should also *know* a witness in order to convince the verifier. In this case, we use the denomination *proof of knowledge* (PoK), and we need the special soundness property, which is stronger than computational soundness.

To illustrate the notion of ZKP, we present two Σ-protocols in the discrete logarithm setting, whose security will be discussed in Section 2.3.2. In both cases, we consider a group G of known prime order q.

Chaum-Perdersen proof. Given a DDH tuple g_1, g_2, g_3, g_4, the Chaum-Pedersen protocol allows to prove that $\log_{g_1}(g_2) = \log_{g_3}(g_4)$, knowing their common discrete logarithm. The most common use case is to prove that an ElGamal encryption (x, y) is an encryption of 1 with a public key pk. In this case, $(g_1, g_2) = \mathsf{pk}$, $(g_3, g_4) = (x, y)$ and the witness is the randomness r used to encrypt 1. Going back to the example of electronic voting, a proof of correct decryption for the pair (m, C) can be done using $(g_1, g_2) = \mathsf{pk}$, $g_3 = x$, $g_4 = y/m$ and the witness sk, where $C = (x, y)$. The Chaum-Pedersen protocol is illustrated in Fig. 8.

Proof of knowledge. Given two group elements (g, h), a proof of knowledge of the discrete logarithm allows Alice to prove that she knows $\log_g(h)$. This is useful for authentication protocols and is used in the Schnorr sigature. The corresponding Σ-protocol is illustrated in Fig. 9.

2.3.2 Generalization

The two Σ-protocols given in the previous section are very similar. In fact, they are both a special case of a more generic Σ-protocol, which is the proof of knowledge of a preimage. More formally, consider two groups G and H, and a group homomorphism $\varphi : G \to H$. We suppose that the order of H does not have a divisor smaller than 2^λ (except 1). This suits the discrete logarithm setting where H's exponent is a publicly known prime number. (The exponent of a group is the smallest non-zero integer $n \in \mathbb{N}$ such that $g^n = 1$ for all g. For instance, $G \times G$ has the same exposant as G but not the same order.) Given $h \in H$, it is usually hard to decide whether $h \in \varphi(G)$; however, given a preimage $x \in G$, this becomes easy. Hence, if we use the notations from Section 2.3.1, we have $\mathscr{L} = \varphi(G)$; a witness for $z \in \mathscr{L}$ is a preimage w of z and we have $w\mathcal{R}z$ if and only if $\varphi(w) = z$. An example is the exponentiation $\mathbb{Z}_q \to G \times G$ which maps an element $r \in \mathbb{Z}_q$ to the pair (g^r, h^r), where (g, h) is public and fixed. Now, in this generic context, we consider the following Σ-protocol, where Alice tries to convince Bob that she knows a preimage w of an element $z \in H$.

Commitment. Alice picks a random $\alpha \in G$ and sends $c = \varphi(\alpha)$ to Bob.

Challenge. Bob picks a random $d \in [0, 2^\lambda - 1]$ and sends this to Alice.

Answer. Alice computes $a = \alpha w^d$ and sends this to Bob.

Verification. Bob checks that $c = \varphi(a)z^{-d}$.

We now discuss about the security properties of this protocol, and show that it provides the desired properties:

- **Completeness.** If Alice is honest, $\varphi(a)z^{-d} = \varphi(\alpha w^d)\varphi(w)^{-d} = \varphi(\alpha) = c$.

- **Zero knowledge.** Given $d \in [0, 2^\lambda - 1]$, one can pick a random $a \in G$ and compute $c = \varphi(a)z^{-d}$. Since a is chosen at random, c is also uniformly random in $\varphi(G)$ and follows the same distribution as in the real protocol. Similarly, in the real protocol, $a = \alpha w^d$, which is indeed uniform in G since α is uniform. Hence, the couple (c, a) is a uniform couple such that $c = \varphi(a)z^{-d}$.

- **Special soundness.** Let $z \in H$ and $c \in H$. Suppose that there exists two distinct values $d_1, d_2 \in [0, 2^\lambda - 1]$ for which there exists a valid answer $a_1 \in G$ (resp. $a_2 \in G$), such that $c = \varphi(a_1)z^{-d_1}$ and $c = \varphi(a_2)z^{-d_2}$. Then $\varphi(a_1)z^{-d_1} = \varphi(a_2)z^{-d_2}$, so that $\varphi(a_2/a_1) = z^{d_2-d_1}$. Now, since the order n of H has no divisor smaller that $2^\lambda > |d_2 - d_1|$, $d_2 - d_1$ is prime with n and there exists $\alpha \in \mathbb{Z}$ such that $\alpha(d_2 - d_1) = 1$ modulo n, which gives $z = \varphi((a_2/a_1)^\alpha)$; hence $z \in \varphi(G)$. By contraposition, if $z \notin \varphi(G)$, for all $c \in H$, there exists at most one $d \in [0, 2^\lambda - 1]$ for which a valid answer a can be found. Hence Bob accepts with probability at most $2^{-\lambda}$.

The special soundness does not contradict the zero knowledge property: Bob can generate the transcripts himself and wait for a collision, in which case he will be able to extract a preimage. This shows that finding a preimage by a one-way group homomorphism it is not harder than waiting for a collision between two random elements in $\varphi(G)$. This is because a homomorphism has a lot more structure than a hash function, for which computing a preimage is harder than finding a collision. In any case, the special soundness property is the main reason why we use the terminology *proof of knowledge*.

2.3.3 Proof of partial knowledge

The generic proof of Section 2.3.2 is very versatile and allows to prove various propositions in the discrete logarithm setting. However, it is not sufficient in electronic voting. Indeed, suppose that we want to compute the result of a yes/no vote. As seen in Section 2.2.2, the usual strategy is to fix a generator $\tilde{g} \in G \backslash \{1_G\}$ and to encrypt either $\tilde{g}^0$ (no vote) or $\tilde{g}^1$ (yes vote). Then the product of all the eligible ciphertexts is computed and decrypted. Thanks to the homomorphic property of the ElGamal encryption scheme, the number of yes votes can be deduced from the discrete logarithm of the decryption in base $\tilde{g}$. This is the principle of homomorphic tally. Yet, in electronic voting, we usually do not trust the voters: some may try to encrypt $\tilde{g}^{42}$ or $\tilde{g}^{-23}$, allowing them to have their choice be counted multiple times. Therefore, we want the voters to prove that they either voted for yes or for no. For this purpose, we use a proof of partial knowledge, which is based on [CDS94].

In this paper, Cramer, Damgård and Schoenmakers develop an abstract framework which allows to produce a proof of a partial knowledge. For the sake of simplicity, we present a slightly less generic framework. First, we restrict ourselves to the case of a proof of knowledge of a preimage, as introduced in Section 2.3.2. This is the most standard case in electronic voting. Second, we instantiate their notion of *monotonic access structure* by using Shamir's secret sharing scheme, which is introduced in Section 2.2.3. This not only allows to fix the idea, but also is directly useful in threshold cryptography.

Generic construction. Let $G_1, \cdots, G_n, H_1, \cdots, H_n$ be some groups and, for all i, $\varphi_i : G_i \to H_i$ be some group homomorphism. We suppose that there exists a common known prime number q such that, for all i, H_i is group of exponent q. This way, we are in the same context as in both Section 2.3.2 and Section 2.2.3. Given $z_1 \in H_1, \cdots, z_n \in H_n$ and a threshold $t < n$, Alice wants to convince Bob that she knows a preimage for at least t of these elements, but she does not want to reveal which one. For this purpose, she forms a set $S \subset [1, n]$ of size t such that, for all $i \in S$, she knows a preimage w_i such that $\varphi_i(w_i) = z_i$. (If she knows more that t preimages, she can ignore this extra knowledge.) Then the protocol consists of the following steps.

Commitment.

- For all $i \in S$, Alice picks a random element $\alpha_i \in G_i$ and computes $c_i = \varphi_i(\alpha_i)$.
- For all $i \in [1, n] \backslash S$, Alice picks a random $d_i \in \mathbb{Z}_q$, chooses a random $a_i \in G_i$ and computes $c_i = \varphi_i(a_i) z_i^{-d_i}$.
- Alice sends her commitments $c_1, \ldots, c_n$ to Bob.

Challenge. Bob picks a random $d \in [0, 2^\lambda - 1]$ and sends this to Alice.

Answer. Alice splits the challenge into n challenges and answers each of them individually:

- Alice sets $d_0 = d$.
- For $i \in S$, she computes $d_i = \sum_{j \in [0,n] \backslash S} \prod_{k \in [0,n] \backslash (S \bigcup \{j\})} \frac{i-k}{j-k} d_j$ modulo q and deduces $a_i = \alpha_i w_i^{d_i}$.
- Alice sends $(d_i, a_i)_{i=1}^n$ to Bob.

Verification. Bob verifies that the two following statements; the second one can be checked efficiently using Lagrange interpolation.

- For all i, $c_i = \varphi(a_i) z_i^{-d_i}$.
- There exists a polynomial $P \in \mathbb{Z}_q[X]$ of degree at most $n - t$ such that $P(i) = d_i$ for all $i \in [0, n]$, with $d_0 = d$.

The above protocol is perfectly *witness indistinguishable*: Bob is convinced that Alice knows a preimage for at least t of the given elements, but does not learn which one. In addition, it inherits the correctness, the special soundness and the zero knowledge properties from the Σ-protocol presented in Section 2.3.2 (see [CDS94] for a proof of these claims).

Disjunctive proofs. A common special case is the disjunctive proof, where Alice wants to prove that she knows at least one witness (*i.e.* $t = 1$). In this case, Shamir's secret sharing scheme becomes linear. More precisely, if i is the index of the element z_i for which Alice knows a preimage w, she can proceed as follows:

- Alice picks a random $\alpha \in G_i$ and compute $c_i = \varphi_i(\alpha)$. For $j \neq i$, she chooses a random $d_j \in \mathbb{Z}_q$ as well as a random $a_j \in G_j$ and computes $c_j = \varphi_j(a_j) z_j^{-d_j}$.
- Bob receives the commitments $c_1, \cdots, c_n$ and sends a random challenge $d \in [0, 2^\lambda - 1]$.
- Alice computes $d_i = d - \sum_{j \neq i} d_j$ and $a_i = \alpha w^{d_i}$. She sends $(d_i, a_i)_{i=1}^n$ to Bob.
- Bob checks that $c_j = \varphi_j(a_j) z_j^{-d_j}$ for all j and that $d = \sum_{j=1}^n d_j$.

In electronic voting, disjunctive proofs are used to prove the validity of a ballot: if there are n possible voting options, the voter can use a disjunctive proof to prove that the ballots encrypts one of them. Alternatively, when the voter is asked many independent binary questions, the voter can produce that many independent ciphertexts to prove that they answered yes or no to all the questions. More concrete examples are given in Section 2.4.

2.3.4 Non-interactive proofs

Up until now, all the ZKP that we presented were interactive. Yet, in electronic voting, there are two reasons why interactive proofs are not desirable. First, an interactive proof does not leave any trace. After that Alice has convinced Bob, the latter cannot use the transcript to convince any third party since he might have produced the transcript himself, using the simulator. This means that interactive proofs cannot be used to achieve universal verifiability. Second, interactive proofs require the verifier to be active during the proof, which is not the case in electronic voting.

The Fiat-Shamir heuristic. The usual way to obtain a non-interactive ZKP from a Σ-protocol is to use the Fiat-Shamir transformation [FS86], which consists of replacing the challenge by a hash of the commitment. This produces a proof $\pi = (c, a)$ which can be verified by computing the commitment d from the hash function and checking that the transcript (c, d, a) is valid. This heuristic is standard and is the one that we consider in this thesis. Usually, we use the abbreviation ZKP to designate a non-interactive zero knowledge proof obtained from a Σ-protocol for a disjunctive proof (see Section 2.3.3), using the Fiat-Shamir transformation. In the specific case where the non-interactive proof is derived directly from a Σ-protocol for proving the knowledge of a preimage (see Section 2.3.2), we may also use the abbreviation PoK (for proof of knowledge). In Algorithm 15, we illustrate the Fiat-Shamir heuristic by giving the algorithm which produces a PoK for the equality of discrete logarithm. This can be used to provide a proof of correct decryption, with $(g_1, g_2) = \mathsf{pk}$ and $(g_3, g_4) = (x, y/m)$, where pk is the

Algorithm 15: EQLOG

Requires: A group G of prime order q
A hash function
$(g_1, g_2, g_3, g_4) \in G$
A witness $w \in \mathbb{Z}_q$ s.t. $g_3 = g_1^w$ and $g_4 = g_2^w$

1 $\alpha \xleftarrow{\$} \mathbb{Z}_q$;
2 $c_1 \longleftarrow g_1^\alpha$;
3 $c_2 \longleftarrow g_2^\alpha$;
4 $d \longleftarrow \text{hash}(g_1||g_2||g_3||g_4||c_1||c_2)$;
5 $a \longleftarrow \alpha + dw$;
6 **return** (c_1, c_2, a);

Algorithm 16: 0/1 proof

Requires: G of order q and generator b
A ciphertext (x, y) obtained with the public key (g, h)
$i \in \{0, 1\}$ s.t. (x, y) encrypts b^i
The corresponding randomness r

1 $\alpha \xleftarrow{\$} \mathbb{Z}_q$; $d_{1-i} \xleftarrow{\$} \mathbb{Z}_q$; $a_{1-i} \xleftarrow{\$} \mathbb{Z}_q$;
2 $c_{i,x} \longleftarrow g^\alpha$; $c_{i,y} \longleftarrow h^\alpha$;
3 $c_{1-i,x} \longleftarrow g^{a_{1-i}} x^{-d_{1-i}}$;
4 $c_{1-i,y} \longleftarrow h^{a_{1-i}} (y/b^{1-i})^{-d_{1-i}}$;
5 $d \leftarrow \text{hash}(g||h||b||x||y||c_{0,x}||c_{0,y}||c_{1,x}||c_{1,y})$;
6 $d_i \longleftarrow d - d_{1-i}$; $a_i \longleftarrow \alpha + r d_i$;
7 **return** $(c_{0,x}, c_{0,y}, c_{1,x}, c_{1,y}, d_0, d_1, a_0, a_1)$;

public ElGamal encryption key, (x, y) is the ciphertext and m is the claimed plaintext. To verify the validity of such a PoK $\pi = (c_1, c_2, a)$, one computes d from the hash function, and check that $c_1 = g_1^a g_3^{-d}$ and $c_2 = g_2^a g_4^{-d}$. Another extremely common ZKP is given in Algorithm 16, which produces a ZKP that a ciphertext (x, y) is indeed an encryption of either 0 or 1, when the exponential ElGamal encryption is used with a base b (usually, b is either g or h, where (g, h) is the public key). To verify the validity of the output $\pi = (c_{0,x}, c_{0,y}, c_{1,x}, c_{1,y}, d_0, d_1, a_0, a_1)$, one first computes d from the hash function, checks that $d = d_0 + d_1$ and verifies that $c_{i,x} = g^{a_i} x^{-d_i}$ and $c_{i,y} = h^{a_i} (y/b^i)^{-d_i}$ for both $i \in \{0, 1\}$.

In both examples, remark that the challenge is not computed from the commitment alone, but also from other pieces of context. We discuss more about what exactly should be included in the hash in Section 3.2.2.

Security in the random oracle model. The security of the Fiat-Shamir transformation can be proven in the ROM. Indeed, consider a Σ-protocol P for a NP language $\mathscr{L}$ (*i.e.* three algorithms Com, Ans and Ver). Suppose that P is zero knowledge, and let Sim be the simulator algorithm. Then, for any prefix pre, a non-interactive ZKP in which the challenge is chosen from the commitment c as $d = \text{hash}(\text{pre}||c)$ can be simulated in the ROM, given only $z \in \mathscr{L}$. Indeed, consider an adversary $\mathbb{A}$ in the ROM, which is given a honestly generated ZKP π at some point. Then we can construct an adversary $\mathbb{B}$ which interacts with $\mathbb{A}$ by simulating the random oracle, but is given z instead of π. The idea is that $\mathbb{B}$ picks a random $d \in [0, 2^\lambda]$, computes $\pi = \text{Sim}(d)$, handles this proof to $\mathbb{A}$ and then outputs whatever $\mathbb{A}$ might output. To make sure that the simulated proof actually looks like a valid one to $\mathbb{A}$, whenever the latter makes an oracle query with input pre$||c$ (where $\pi = (c, a)$), $\mathbb{B}$ answers with d. Since d was chosen at random anyway, this does not change the distribution of $\mathbb{A}$'s view, and therefore the output of $\mathbb{B}$ will have the same distribution as that of $\mathbb{A}$. Using the same strategy, $\mathbb{B}$ can even *forge* fake ZKP that will look valid to $\mathbb{A}$, even if the statement is false. For this reason, a non-interactive ZKP can be safely *removed* from $\mathbb{A}$'s view.

Now, suppose that the Σ-protocol is computationally sound. Then the corresponding non-interactive ZKP is also computationally sound. Indeed, for a given $z \notin \mathscr{L}$, suppose that there exists an adversary $\mathbb{A}$ that outputs a valid ZKP π with some non-negligible probability. Then we construct an adversary $\mathbb{B}$ for the Σ-protocol as follows. First, let q be a polynomial such that $\mathbb{A}$ can make at most q oracle queries. Then $\mathbb{B}$ picks a random $i \in [1, q]$ and runs a copy of $\mathbb{A}$ in the ROM. At the ith query (if any), $\mathbb{B}$ parses the input as pre$||c$ (if possible) and uses this

c as the commitment. The honest verifier answers with a uniformly random d, so that $\mathbb{B}$ can use this as the output of the random oracle (for simplicity, we consider that $\mathbb{A}$ does not make two queries to the oracle with the same inputs). When $\mathbb{A}$ terminates, it outputs a valid proof $\pi = (c, a)$ with some non-negligible probability p, in which case $\mathrm{Ver}(c, \mathcal{O}_{\mathsf{RO}}(\mathrm{pre}||c), a) = 1$. Yet, the computational soundness assumption means that this cannot happen with a non-negligible probability if $\mathcal{O}_{\mathsf{RO}}(\mathrm{pre}||c)$ is uniform. Hence, $\mathcal{O}_{\mathsf{RO}}(\mathrm{pre}||c)$ is not uniform which means that $\mathbb{A}$ made a query to the random oracle with the input $\mathrm{pre}||c$. Since the choices were independent, $\mathbb{B}$ has picked this very query with probability $1/q$; therefore the verifier will accept the answer a with the non-negligible probability p/q.

Thanks to the above arguments, the ROM allows to show that the Fiat-Shamir transformation preserves computational soundness while still being "zero knowledge" with overwhelming probability (indeed, if the prover makes two colliding commitments, the adversary that interacts with the real prover might be able to extract a witness; on the other hand, the adversary that interacts with the simulator might notice that the same oracle query was answered with two different challenges). However, this might be not enough in many cases, for instance in the example of the PoK depicted in Fig. 9. Indeed, in most case, we not only want the prover to prove that $z \in \mathscr{L}$, but also to prove that they know a witness. This is important, for instance, for signatures or authentication protocols. In this situation, we need more involved arguments that are presented in Section 3.2.1.

Another remark is that the above proof sketches assumed that $\mathscr{L}$ and z are fixed; *i.e.* hard-coded in the verifier's algorithm. In Section 3.2.2, we show that a non-interactive proof made for a specific $\mathscr{L}$ and z can be considered valid for another $\mathscr{L}'$ and z', which intuitively breaks soundness (this does not contradict the above arguments which assumed that the adversary was unable to choose z and $\mathscr{L}$). For this reason, it is important that the prefix in the hash contains enough pieces of context, which prevents the proof from being used outside of its context. For instance, a description of G, $\mathscr{L}$, the claim z or even a unique identifier which defines at which step of the protocol the proof was needed can also be included in the hash. This can be a lot when there is too much context: think of an electronic voting protocol which uses this specific counting function, takes place at that specific date and the other. However, a great part of this context is actually fixed during the whole protocol and can be factored between several ZKPs, for instance by using a hash of the context instead of the whole context.

2.3.5 Short proofs and what they can really do

The standard proofs that we introduced so far all have a similar complexity when it comes to generating or verifying them. While it is natural that the generation of a proof for a statement becomes harder as the statement grows more complex, the intuition is that verifying a proof should be easier. Yet, in electronic voting, while the talliers can have access to a lot of computational resources, this is not always the case for the auditor. On the contrary: if we want a protocol to be "universally" verifiable, it is preferable that the verification be as easy as possible. For this reason, it could be beneficial to use short zero knowledge proofs.

A popular technique to obtain short proofs is to use recursive inner product arguments. This is the main idea behind the bulletproofs [BBB+18], which allow to obtain efficient range proofs in the discrete logarithm setting. Although range proofs may be useful in electronic voting, the relevant ranges are so small that it is often more efficient to use a regular disjunctive proof. Bulletproofs can also be used for any generic NP language. In this case, the proof is logarithmic with the size of the statement but the cost of the verification is proportional to the cost of the generation. Consequently, they are not commonly used in electronic voting.

Groth's SNARK. Another popular techique was introduced in [Gro16]. In this article, Jens Groth explains how to form a short zero knowledge proof for the arithmetic circuit satisfiability decisional problem, which is known to be NP complete. Hence, from Groth's proof strategy, one can derive a short zero knowledge proof for any NP language $\mathscr{L}$. In [Gro16], a circuit is represented by a family of n quadratic equations of the form

$$\sum_{i=0}^{m} a_i u_{i,q} \sum_{i=0}^{m} a_i v_{i,q} = \sum_{i=0}^{m} a_i w_{i,q},$$

where the a_i's are variables while the $u_{i,q}$'s, $v_{i,q}$'s and $w_{i,q}$'s are public parameters that define the qth equation. Since those equations feature multiplications and additions, the variables and the parameters all belong to a common ring (we actually need it to be a field $\mathbb{K}$ for the purpose of polynomial interpolation). In general, some of the variables (say $a_0, \cdots, a_\ell$) are fixed, otherwise the satisfiability would be trivial. With those constraints, the corresponding statement is to claim that there exists $a_{\ell+1}, \cdots, a_m \in \mathbb{K}$ such that the n equations are simultaneously satisfied.

An interesting specific case is that of boolean circuit satisfiability: a boolean variable can be represented as an element of $\mathbb{Z}_2$, a logical and is a multiplication, a logical xor is an addition and the logical negation of the variable b is simply $1 - b$. In this setting, Groth's proof strategy allows to create a proof which only consists of 3 group elements, so that its size only depends on the security parameter and not on the complexity of the statement or the size of the circuit. Computing such a proof requires n exponentiations in a pairing-friendly curve, but verifying it only requires ℓ group multiplications in this curve and 3 pairings.

Computing a SNARK. Now, consider the example of a homomorphic tally: let $C = (x, y)$ be an ElGamal ciphertext with public key $\mathsf{pk} = (g, h)$ and let m be the claimed plaintext. Then we must prove that there exists $\mathsf{sk} \in \mathbb{Z}_q$ such that $g^{\mathsf{sk}} = h$ and $yx^{-\mathsf{sk}} = m$. To express this as an arithmetic circuit, the only obvious way is to use a boolean circuit (*i.e.* with $\mathbb{K} = \mathbb{Z}_2$) which realizes a constant-time implementation (*e.g.*, using the Montgomery ladder) of the exponentiations and then performs the equality tests. To fix ideas, suppose that we use a Weierstrass elliptic curve of prime order q, whose group elements are represented using projective coordinates in $\mathbb{Z}_p$, where p is some prime number. Then, given a security parameter λ, an exponentiation typically requires $\Theta(\lambda)$ group multiplications, while each group multiplication requires a few operations in $\mathbb{Z}_p$, including some multiplications that cost $\Theta(\lambda^2)$ logical operations (asymptotically better complexities are available, but the corresponding algorithms are usually not faster than Montgomery multiplication for a 256 bits p). Consequently, the naive boolean circuit would require $n = O(\lambda^3)$ exponentiations for the prover, which are to be compared with the 2 exponentiations required in the corresponding standard PoK. Consequently, making an efficient use of Groth's proofs in electronic voting is not an easy task: this is the contribution of Kryvos [HKK+22], a recent academic work which demonstrates the feasibility of this approach.

Verifying a SNARK. Groth's proofs have this very interesting property that their verification always require ℓ group multiplications and 3 pairings, no matter the complexity of the circuit (*i.e.* the parameters n and m). In our example of a homomorphic tally, $\ell = \Theta(\lambda)$ since the fixed variables are defined by the group elements x, y, m. Therefore, verifying the proof requires $\Theta(\lambda)$ group multiplications and 3 pairings. Compared to the 4 exponentiations required to verify the standard PoK, this is not especially interesting. However, in Part II, we consider more complex tally protocols whose verifications using standard ZKPs are quite intensive. Using a SNARK, the verification would typically require $O(n_V \lambda)$ group multiplications and 3 pairings, where n_V is the number of voters. This would definitely be a lot faster.

Trapdoor CRS. An important point to mention is that Groth's proofs require a common reference string (CRS). This CRS contains many group elements and scalars that are useful to

speed up the algorithms of both the prover and the verifier. However, it is possible to *trapdoor* a CRS, for instance by choosing the discrete logarithm of the said group elements instead of them directly. Using the knowledge of the trapdoor, it is possible to forge fake proofs for false statements. In general, this is not an issue because the CRS can be generated using a public coin protocol, so that no one can exploit the trapdoor. However, the CRS used in [Gro16] is constrained by some algebraic relationships, so that it cannot be derived from some random string. As shown in [BCG+15a], it is possible to have several authorities jointly generate the CRS so that none of them is able to trapdoor it (except if they are all corrupted). This means that using Groth's proofs requires an additional trust assumption on the participants of this CRS generation protocol, otherwise verifiability will be lost.

2.4 The most commonly used ZKP

The typical use case of ZKP in electronic voting is to prove the validity of a ballot and to prove the correctness of the tally. In this section, we review the most standards ZKPs, apart from the proofs that we already detailed in Section 2.3.4.

2.4.1 A basic example: proving the validity of a ballot

In electronic voting, there are two main strategies to perform the tally: homomorphic tally and mixnets. Helios 2.0 [dMPQ09], which is presented in Section 1.1.3, uses a homomorphic tally. In this voting system, the voters are ask a question for which there are several possible answers, numbered from 1 to k. The voters can select between k_1 and k_2 answers, where $0 \leq k_1 \leq k_2 \leq k$. This choice can be encoded into a k bits string $a_1, \cdots, a_k$, depending on whether each answer was picked or not. Consequently, the voting options in Helios $\mathcal{V}_H$, where

$$\mathcal{V}_H = \left\{ a_1, \cdots, a_k \in \{0,1\} \mid \sum_{i=1}^{k} a_i \in [k_1, k_2], \right\},$$

which can be decomposed as

$$\mathcal{V}_H = \bigcup_{s \in [k_1, k_2]} \left\{ a_1, \cdots, a_k \in \{0,1\} \mid \sum_{i=1}^{k} a_i = s \right\}.$$

To encrypt such a voting option, one can produce k ciphertexts $C_1, \cdots, C_k$, using an exponential ElGamal encryption with a base b. Then, to prove that the ciphertexts encrypt a valid voting option, one can proceed as follows:

- For $i \in [1, k]$, produce a ZKP $\pi_i^{0/1}$ that C_i is an encryption of 0 or 1, using Algorithm 16. Those are the *individual proofs*.
- Produce a disjunctive ZKP π_s that the product $C_1 \cdots C_k$ is an encryption of either k_1, $k_1 + 1$, $\cdots$, or k_2. This is the overall proof.
- Return the proof $\pi = \pi_1^{0/1} || \cdots || \pi_k^{0/1} || \pi_s$.

For completeness, we give in Algorithm 17 an algorithm that the voter can use to produce π_s. In our example, (x, y) is the product $(C_1 \cdots C_k)$; $n = k_2 - k_1 + 1$; the v_i's are the elements

Algorithm 17: Disjunctive proof for ElGamal encryption

Requires: A group G of prime order q
An ElGamal ciphertext (x, y) obtained with public the key (g, h)
n possibilities $v_1, \cdots, v_n \in G$
$i \in \{1, n\}$ s.t. (x, y) encrypts v_i
The corresponding randomness $r \in \mathbb{Z}_q$

1 $\alpha \xleftarrow{\$} \mathbb{Z}_q$; $c_{i,x} \longleftarrow g^\alpha$; $c_{i,y} \longleftarrow h^\alpha$;
2 **for** $j \neq i$ **do**
3 $\quad d_j \xleftarrow{\$} \mathbb{Z}_q$; $a_j \xleftarrow{\$} \mathbb{Z}_q$;
4 $\quad c_{j,x} \longleftarrow g^{a_j} x^{-d_j}$; $c_{j,y} \longleftarrow h^{a_j}(y/v_j)^{-d_j}$;
5 $d \longleftarrow \text{hash}(g||h||v_1||\cdots||v_n||x||y||c_{1,x}||c_{1,y}||\cdots||c_{n,x}||c_{n,y})$;
6 $d_i \longleftarrow d - \sum_{j \neq i} d_j$; $a_i \longleftarrow \alpha + r d_i$;
7 **return** $(c_{j,x}, c_{j,y}, d_j, a_j)_{j=1}^n$;

of $\{b^s \mid s \in [k_1, k_2]\}$; and r is the sum of all the k randomness that have been used to produce the ciphertexts.

This shows that a large variety of counting functions can be covered by a homomorphic tally, which is not restricted to a single yes / no question. As the 0/1 proof is the basis of many proofs of validity in electronic voting, we mention that it is possible to optimize Algorithm 16 to increase its efficiency for both the verifier and the prover, especially when several bits are encrypted simultaneously; see for instance [DPP22a]. Nevertheless, not every functions are covered by homomorphic tally. For more generic counting functions, the usual strategy is to use a mixnet (see Section 2.4.3), which reveals the chosen voting options in some random order. Hence, anonymity is preserved and the tally can be publicly computed on the cleartexts. However, even when a mixnet is used, it is interesting to demand that the voter proves the validity of their ballot: this protects against Italian attacks based on write-ins and prevents replay attacks based on the malleability of the ElGamal encryption scheme. For this purpose, the disjunctive proof of Algorithm 17 can be used (provided that there is a small number of voting options).

2.4.2 Proof of correct decryption

In Section 2.3, we already discussed about how to produce a ZKP of correct decryption for an ElGamal ciphertext, and we gave Algorithm 15 which does just that. However, the secret key sk is usually shared between several authorities (the talliers), so that they cannot use this algorithm. The usual way around is to have each authority reveal a partial decryption of the ciphertext and to recombine the valid partial decryptions afterwards. This gives Algorithm 18, which not only produces the threshold decryption of some ciphertext (x, y), but also a ZKP π of correct decryption. To verify this proof, verify each ZKP π_i^{PartDec} and, if there are at least $t + 1$ valid ones (otherwise, reject the proof), choose any set S as in line 2 and compute m as in lines 3 and 4. Finally, check that this m is equal to the claimed plaintext.

2.4.3 Mixnets and their applications

Mixnets are widely used in cryptography as a way to anonymize data. The idea is to shuffle encrypted or committed data so that it is no longer possible to trace them back to their original sender. In electronic voting, we consider reencryption mixnets and decryption mixnets.

Algorithm 18: Threshold decryption

Requires: A group G of prime order q
An ElGamal public key (g, h)
A ciphertext (x, y) to decrypt
The public commitments $(h_i)_{i=1}^n$ on the secret shares (see Fig. 7)
Each tallier has a share s_i of the secret key, with the threshold t

1 Each participant i computes $w_i = x^{s_i}$ and π_i^{PartDec} using Algorithm 15 on (g, x, h_i, w_i);
2 Let $S \subset [1, n]$ be a set of size $t + 1$ such that for all $i \in S$, π_i^{PartDec} is valid ;
3 For $i \in S$, compute $\Lambda_i = \prod_{j \in S \setminus \{i\}} \frac{j}{i-j}$;
4 $m \longleftarrow y \prod_{i \in S} w_i^{\Lambda_i}$;
5 $\pi \longleftarrow w_1, \pi_1^{\mathsf{PartDec}} || \cdots || w_n || \pi_n^{\mathsf{PartDec}}$;
6 **return** m, π

Proof of a shuffle. Let $\boldsymbol{C} = C_1, \cdots, C_n$ be a list of ciphertexts. To fix ideas, suppose that the encryption scheme used is rerandomizable: given any ciphertext C, we can produce another ciphertext C' which encrypts the same plaintext but is still indistinguishable from a random ciphertext. This is the case for any homomorphic encryption scheme, such as the ones of ElGamal and Paillier. Then, a *shuffle* of those ciphertexts is a list $\boldsymbol{C'} = C'_1, \cdots, C'_n$ which encrypts the same plaintexts. In other words, we have the equality of the following multisets

$$\{\!\{\mathrm{Dec}_{\mathsf{sk}}(C'_i) \mid i \in [1, n]\}\!\} \equiv \{\!\{\mathrm{Dec}_{\mathsf{sk}}(C_i) \mid i \in [1, n]\}\!\}.$$

When the encryption scheme is rerandomizable, anyone can shuffle $\boldsymbol{C}$; however, it is hard to decide whether a list $\boldsymbol{C'}$ is a shuffle of $\boldsymbol{C}$ or not. Therefore, to prevent the mixers from altering the data, we need them to provide a ZKP, which is called *proof of a shuffle*. Early examples of efficient proofs of a shuffle can be found in [FS01, Nef01]. Later, more efficient proofs of a shuffle were given in [Wik09, TW10]. Those proofs were implemented and compiled into a free open source verifiable mixnet, known as Open Verificatum [Ver].

The strategy in [Wik09] is to first commit on a permutation π, then to prove that $\boldsymbol{C'}$ is *consistent* with respect to the commitment. The second part can be done using standard PoK as introduced in Section 2.3.2, but the construction is a bit involved. If reading [Wik09] seems too scary, we refer to [HKLD17, Section 5.5] for a helpful and detailed explanation. As for the commitment, one can use the technique from [TW10] which we explain below.

Committing to a permutation. Suppose that we have several independent (provably random) group elements $g, h_1, \cdots, h_n \in G$, such that no one knows a non-trivial relation between them. They can be used to make a Pedersen commitment of vectors of n elements; *i.e.* if $\boldsymbol{x} = x_1, \cdots, x_n \in \mathbb{Z}_q$, and $\alpha \in \mathbb{Z}_q$, $\mathrm{Com}(\boldsymbol{x}, \alpha) = g^\alpha \prod_{i=1}^n h_i^{x_i}$. Now, consider a matrix $M \in \mathcal{M}_n(\mathbb{Z}_q)$. One can commit to this matrix by using random $\boldsymbol{s} = s_1, \cdots, s_n \in \mathbb{Z}_q$ and computing $a_i = \mathrm{Com}(s_i, \boldsymbol{m_i})$ for all i, where m_i is the ith row of the matrix. To prove that a committed matrix is a permutation matrix, the main strategy is to remark that if M does not have exactly one non-zero coefficient in each row, then following equality does not hold in $\mathbb{Z}_q[X_1, \cdots, X_n]$:

$$\prod_{i=1}^n \sum_{j=1}^n m_{i,j} X_j = \prod_{i=1}^n X_i. \tag{1}$$

This condition can be efficiently tested using a result known as the Schwartz-Zippel lemma [Zip79, Sch80]: If $f \in \mathbb{Z}_q[X_1, \cdots, X_n]$ is a non-zero multivariate polynomial of degree d, then, given

uniformly random elements $e_1, \cdots, e_n \in \mathbb{Z}_q$, the probability that $f(e_1, \cdots, e_n) = 0$ is at most d/q. Since the degree of the above polynomial is the number n of ciphertexts to shuffle while q is the exponentially large size of the field, it means that this probability is negligible. Hence, the proof strategy is that the verifier (or, alternatively, the Fiat-Shamir transformation) gives to the prover the random vector $\boldsymbol{e}$ and that the prover proves that $\prod_{i=1}^n \sum_{j=1}^n m_{i,j} e_j = \prod_{i=1}^n e_i$. For this purpose, remark that this can be made "on the commitments". Indeed, if we denote $< \boldsymbol{x}, \boldsymbol{y} >$ the inner product $\sum_{i=1}^n x_i y_i$ of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, we have

$$\prod_{i=1}^n a_i^{e_i} = g^{<\boldsymbol{s},\boldsymbol{e}>} \prod_{i=1}^n h_i^{<\boldsymbol{m_i},\boldsymbol{e}>}.$$

Yet, Eq. (1) is verified if and only if there exists some coefficients $c_1, \cdots, c_n$ and a permutation π such that, for all i, $\sum_{j=1}^n m_{i,j} X_j =< \boldsymbol{m_i}, \boldsymbol{X} >= c_i X_{\pi(i)}$, with $\prod_{i=1}^n c_i = 1$. Therefore, by proving the knowledge of some $\boldsymbol{e'} = e'_1, \cdots, e'_n \in \mathbb{Z}_q$ ($e'_i = e_{\pi(i)}$ for all i) and an exponent $\alpha =< \boldsymbol{s}, \boldsymbol{e} >\in \mathbb{Z}_q$ such that $g^\alpha \prod_{i=1}^n h_i^{e'_i} = \prod_{i=1}^n a_i^{e_i}$ while $\prod_{i=1}^n e'_i = \prod_{i=1}^n e_i$, one can prove that the committed matrix has exactly one non-zero coefficient per row, unless with a negligible probability. Finally, to prove that this coefficient is always 1, and therefore that the matrix is a permutation matrix, one can prove the knowledge of $\beta = \sum_{i=1}^n s_i$ such that $\prod_{i=1}^n a_i/g_i = g^\beta$. These proofs can be done using standard ZKP as in Section 2.3.2.

Remark that although the resulting proof of a shuffle is based on standard ZKP, they do not have the special soundness property. Indeed, there is always a negligible probability that, although the preimages $\alpha, \beta, \boldsymbol{e'}$ exist, the polynomial is non-zero so that the matrix is not a permutation matrix. Consequently, a proof of a shuffle is only computationally sound.

Shuffling the rows or the columns of a matrix. In this thesis, we consider a more generic case where the C_i's themselves consist of several ciphertexts. More precisely, suppose that we have a rectangular matrix M of n rows and m columns, and that each coefficient of this matrix is an ElGamal ciphertext, *i.e.* a pair of two group elements. Suppose that we want to shuffle the rows (or, alternatively, the columns) of the matrix, which means that we want to shuffle each column, but using the same permutation π. Then the Terelius-Wikström strategy is especially efficient, since it allows to use the same commitment for many proofs. We denote $\mathrm{ShuffleRow}(M, \pi)$ (resp. $\mathrm{ShuffleColumn}(M, \pi)$), the algorithm that shuffles and reencrypts the rows (resp. columns) of M, and produces a ZKP of a correct shuffle. For completeness, we give Algorithm 19, which combines [TW10, Protocol 1] and [Wik09, Protocol 15]. However, to prove that $\prod_{i=1}^n e'_i = \prod_{i=1}^n e_i$, we use the strategy of [HKLD17, Section 5.5] which consists of creating a chain of Pedersen commitments $\tilde{c}_1, \cdots, \tilde{c}_n$. To verify the proof, one computes $(e_i)_{i=1}^n$ and d from the hash and check that the following equalities hold, where $\hat{c}_0 = h_1$:

$$g^{a_t} \left(\prod_{i=1}^n (a_i^\star / h_i) \right)^{-d} = c_{\bar{1},t}$$
$$\forall i, g^{a_{\hat{r},i}} \hat{c}_{i-1}^{a_{\bar{e},i}} \hat{c}_i^{-d} = c_{\hat{c},i}$$
$$g^{a_k} \prod_{i=1}^n h_i^{a_{\bar{e},i}} \prod_{i=1}^n (a_i^\star)^{-de_i} = c_{\bar{e},k}$$
$$\forall j, \mathrm{Renc}_{\mathsf{pk}} \left(\prod_{i=1}^n M'[i,j]^{a_{\bar{e},i}}, -a_{r,j} \right) \prod_{i=1}^n M[i,j]^{-de_i} = c_{r,j}.$$

A similar algorithm can be used to provably shuffle the columns. Finally, if one wants to shuffle the rows and the columns using the same permutation, ShuffleRow and ShuffleColumn can be combined into ShuffleMatrix. We sum up the complexities of the various proofs of a shuffle in Table 3, which includes the cost to rerandomize the matrix.

Algorithm 19: ShuffleR

Requires: G, a group of prime order q
$g, h_1, \cdots, h_n$, some independent generators
$\mathsf{pk} = (g, h)$, an ElGamal public key
Inputs: M, a matrix of ciphertexts of n rows and m columns
π, a permutation of $[1, \cdots, n]$
Outputs: M', a shuffled and reencrypted matrix
Π, a ZKP of a shuffle

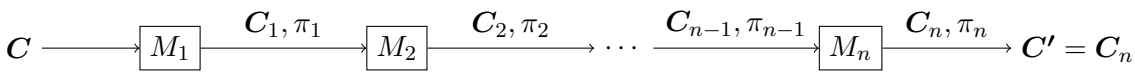
for *all* i, j **do**
  $r_{i,j} \xleftarrow{\$} \mathbb{Z}_q$;
  $M'[i,j] \longleftarrow \mathrm{Renc}_{\mathsf{pk}}(M'[\pi(i), j], r_{i,j})$;
for $i = 1$ *to* n **do**
  $s_i \xleftarrow{\$} \mathbb{Z}_q$; $a_i^\star \longleftarrow g^{s_i} h_{\pi^{-1}(i)}$;
$\alpha_t \xleftarrow{\$} \mathbb{Z}_q$; $c_{\bar{1},t} \longleftarrow g^{\alpha}$;
for $i = 1$ to n **do** $\alpha_{\bar{e}'_i} \xleftarrow{\$} \mathbb{Z}_q$;
$\alpha_k \xleftarrow{\$} \mathbb{Z}_q$; $c_{\bar{e},k} \longleftarrow g^{\alpha_k} \prod_{i=1}^n h_i^{\alpha_{\bar{e}'_i}}$;
$\mathrm{Com}_\pi \longleftarrow (a_i^\star)_1^n, c_{\bar{1},t}, c_{\bar{e},k}$;
for $i = 1$ to n **do**
  $e_i \longleftarrow \mathrm{hash}(\mathsf{pk}||(h_i)_1^n||M||M'||\mathrm{Com}_\pi||i)$;
for $i = 1$ to n **do** $e'_i \longleftarrow e_{\pi(i)}$;
$\hat{c}_0 \longleftarrow h_1$;
for $i = 1$ *to* n **do**
  $\hat{r}_i, \alpha_{\hat{r},i} \xleftarrow{\$} \mathbb{Z}_q$;
  $\hat{c}_i \longleftarrow g^{\hat{r}_i} \hat{c}_{i-1}^{e'_i}$;
  $c_{\hat{c},i} \longleftarrow g^{\alpha_{\hat{r},i}} \hat{c}_{i-1}^{\alpha_{\bar{e}'_i}}$;
$\hat{r}_\top \longleftarrow \sum_{i=1}^n \hat{r}_i \prod_{j=i+1}^n e'_i$;
$\alpha_{\hat{r},\top} \xleftarrow{\$} \mathbb{Z}_q$; $c_{\hat{r},\top} \longleftarrow g^{\alpha_{\hat{r},\top}}$;
$\mathrm{Com}_\pi \longleftarrow \mathrm{Com}_\pi||(\hat{c}_i, c_{\hat{c},i})_1^n||c_{\hat{r},\top}$;
for $j = 1$ *to* m **do**
  $\alpha_{r,j} \xleftarrow{\$} \mathbb{Z}_q$;
  $c_{r,j} \leftarrow \mathrm{Renc}_{\mathsf{pk}}\left(\prod_{i=1}^n M'[i,j]^{\alpha_{\bar{e}'_i}}, -\alpha_{r,j}\right)$;
  $\mathrm{Com}_\pi \longleftarrow \mathrm{Com}_\pi||c_{r,j}$;
$d \longleftarrow \mathrm{hash}(\mathsf{pk}||(h_i)_1^n||M||M'||\mathrm{Com}_\pi)$;
$a_t \longleftarrow \alpha_t + d\sum_{i=1}^n s_i$;
$a_k \longleftarrow \alpha_k + d\sum_{i=1}^n s_i e_i$;
$a_{\hat{r},\top} \longleftarrow \alpha_{\hat{r},\top} + d\hat{r}_\top$;
for $i = 1$ *to* n **do**
  $a_{\bar{e},i} \longleftarrow \alpha_{\bar{e}'_i} + de'_i$;
  $a_{\hat{r},i} \longleftarrow \alpha_{\hat{r},i} + d\hat{r}_i$;
for $j = 1$ to m **do** $a_{r,j} \leftarrow \alpha_{r,j} + d\sum_{i=1}^n e'_i r_{i,j}$;
$\mathrm{Ans}_\pi \longleftarrow a_t, a_k, a_{\hat{r},\top}, (a_{\bar{e},i}, a_{\hat{r},i})_1^n, (a_{r,j})_1^m$;
return M', $(\mathrm{Com}_\pi, \mathrm{Ans}_\pi)$;

Reencryption mixnet. When a proof of a shuffle is used, the mixer knows the permutation that links the two lists $\boldsymbol{C'}$ and $\boldsymbol{C}$. Therefore, the usual strategy is to have several mixers sequentially shuffling the data, each producing a proof of a shuffle. The idea is that all the mixers need to collude in order to retrieve the final permutation: if at least one of them is honest, the permutation remains unknown and random. The generic structure of a mixnet protocol is illustrated in Fig. 10, where each mixer M_i produces a shuffle $\boldsymbol{C}_i$ as well as a ZKP π_i. This produces a transcript which is the concatenation of all those messages. Reencryption mixnets have various applications in cryptography, especially in multi-party computation (MPC) (see for instance Mix and Match, a generic MPC protocol presented in [JJ00]).

Decryption mixnet. In electronic voting, the mixnet is usually applied once, and the ciphertexts are decrypted right away. In this case, it is possible to merge the mixing and the decrypting process: this gives a decryption mixnet, where the mixers are also the talliers who hold the shares of the decryption key. Some examples can be found in [Wik05, AW07].

Table 3: Cost of ShuffleRow, ShuffleColumn and ShuffleMatrix, with n rows and m columns

Proof	Prover (# exp.)	Verifier (# exp.)	Transcript (×256 bits)
ShuffleRow	$4nm + 6n + m + 3$	$4nm + 5n + m + 3$	$2nm + 7n + 3m + 6$
ShuffleColumn	$4nm + 6m + n + 3$	$4nm + 5m + n + 3$	$2nm + 7m + 3n + 6$
ShuffleMatrix	$8n^2 + 13n + 6$	$8n^2 + 12n + 4$	$4n^2 + 19n + 12$



Final transcript: $\boldsymbol{C}||\boldsymbol{C}_1||\pi_1||\cdots||\boldsymbol{C}_n||\pi_n$

Figure 10: Generic structure of a mixnet protocol

2.4.4 Plaintext Equivalence Tests

A plaintext equivalence test (PET) is a now classical protocol introduced in [JJ00]. It allows the key holders to provably reveal whether two ciphertexts have the same plaintext or not. More precisely, suppose that an ElGamal secret key is shared between several authorities, say $P_1, \cdots, P_n$. Suppose that for two given ciphertexts C_1 and C_2, the authorities want to reveal whether they have the same plaintext, but they do not want any other information to leak. Then they can proceed as follows, where q is the prime order of the group G of the plaintexts:

1. Form the ciphertext $C = C_1/C_2$. Each authority P_i chooses $z_i \in \mathbb{Z}_q$ at random and computes $C_i = C^{z_i}$ as well as a ZKP π_i of wellformedness (for this purpose, they can use Algorithm 15). P_i broadcasts a commitment $h_i = \text{hash}(C_i)$.

2. Once all the commitments have been received, P_i broadcasts C_i, π_i. Upon receiving C_j, π_j from another participant, P_i verifies that $\text{hash}(C_j) = h_j$ and that the PoK is valid.

3. Form the ciphertext $D = \prod_{i=1}^{n} C_i$ and decrypt it, using a threshold decryption protocol (see Algorithm 18).

4. Let $m \in G$ be the corresponding plaintext. If $m = 1_G$, then return 1. Otherwise, return 0.

The above protocol allows the authorities to reveal a bit b which states whether the two ciphertexts C_1 and C_2 are equivalent or not. At step 1, they each choose randomly $z_i \in \mathbb{Z}_q$ and compute $C_i = C^{z_i}$, so that $D = C^{z_1+\cdots+z_n}$. The commitment phase is there to prevent a malicious participant to choose z_i depending on the other C_i's, which would rig the distribution of D. This guarantees that $D = C^z$, with some uniformly random $z \in \mathbb{Z}_q$ (as soon as one participant is honest). The idea is that if C_1 and C_2 encrypt the same plaintext, then D is an encryption of 1_G; otherwise, D is an encryption of a uniformly random plaintext. Therefore, by decrypting D, we reveal whether C_1 and C_2 are equivalent, but not any other information. Note that in [MPT20], it was disclosed that it is not sufficient to verify that all the PoK are valid. Indeed, if all the participants are malicious, they can choose their z_i's so that their sum is 0, in which case $D = (1_G, 1_G)$, therefore $m = 1_G$ even though C_1 and C_2 might have two different plaintexts. Therefore, one must also verify that $D \neq (1_G, 1_G)$.

Algorithm 20: DVZKP for EQLOG

Requires: G of prime order q
$(g_1, g_2, g_3, g_4) \in G$
Two public elements
$g, h \in G$
$w \in \mathbb{Z}_q \mid (g_1, g_2)^w = (g_3, g_4)$

$\alpha \xleftarrow{\$} \mathbb{Z}_q$;
$c_1 \longleftarrow g_1^\alpha$; $c_2 \longleftarrow g_2^\alpha$;
$d_2 \xleftarrow{\$} \mathbb{Z}_q$; $a_2 \xleftarrow{\$} \mathbb{Z}_q$;
$c_3 \longleftarrow g^{a_2} h^{-d_2}$;
$d \leftarrow \text{hash}(g_1 || \cdots || g_4 || g || \mathsf{pk}_v || c_1 || c_2 || c_3)$;
$d_1 \longleftarrow d - d_2$;
$a_1 \longleftarrow \alpha + d_1 w$;
return $(c_1, c_2, c_3, d_1, d_2, a_1, a_2)$;

Algorithm 21: DVZKP for EQLOG

Requires: G of prime order q
$(g_1, g_2, g_3, g_4) \in G$
Two public elements
$g, h \in G$
$\mathsf{sk}_v \in \mathbb{Z}_q \mid g^{\mathsf{sk}_v = h}$

$\alpha \xleftarrow{\$} \mathbb{Z}_q$;
$c_3 \longleftarrow g^\alpha$;
$d_1 \xleftarrow{\$} \mathbb{Z}_q$; $a_1 \xleftarrow{\$} \mathbb{Z}_q$;
$c_1 \longleftarrow g_1^{a_1} g_3^{-d_1}$; $c_2 \longleftarrow g_2^{a_1} g_4^{-d_1}$;
$d \leftarrow \text{hash}(g_1 || \cdots || g_4 || g || \mathsf{pk}_v || c_1 || c_2 || c_3)$;
$d_2 \longleftarrow d - d_1$;
$a_2 \longleftarrow \alpha + d_2 \mathsf{sk}_v$;
return $(c_1, c_2, c_3, d_1, d_2, a_1, a_2)$;

2.4.5 Designated Verifier Zero Knowledge Proofs

As explained in Section 2.3.4, a non-interactive ZKP provides a transcript that anyone can verify. In general, this is desirable in electronic voting; however, there can be a few situations where it would be preferable that only a specific entity is convinced by the proof. The main example is in the context of coercion resistance, which is the subject of Part III. To address such situations, designated verifier zero knowledge proofs (DVZKP) were introduced in [JSI96], along with designated verifier signatures. Following this paper, a rich literature was developed (see *e.g.*, [SBWP03, SKM03, LWB05]); some examples of recent academic works are [CC18, BJO+22]. The solution proposed in [JSI96] is extremely simple: suppose that the verifier has a secret key sk_v that corresponds to a public key pk_v (for instance, $\mathsf{pk}_v = (g, h)$ with $h = g^{\mathsf{sk}_v}$). Then, instead of proving a statement ϕ, the prover can make a disjunctive proof that either ϕ is true, either they know the secret key sk (for instance, using a PoK as presented in Section 2.3.2). This way, the verifier is convinced that ϕ is true (unless the secret key was compromised) but cannot use this proof to convince anyone else since they might have produced it using the knowledge of sk.

For completeness, we give in Algorithm 20 an explicit algorithm that can be used to produce a valid DVZKP $\pi = (c_1, c_2, c_3, d_1, d_2, a_1, a_2)$ for the Chaum-Pedersen proof. In other words, given a DDH tuple $(g_1, g_2, g_3, g_4) \in G$ and a witness w, one can prove to the verifier that $\log_{g_1}(g_3) = \log_{g_2}(g_4)$. However, knowing the secret key sk_v, the verifier can forge a valid proof for any tuple, using Algorithm 21. To verify such a proof, one computes d from the hash function, check that $d = d_1 + d_2$ and verify that the three following equations are satisfied:

$$c_1 = g_1^{a_1} g_3^{-d_1};\ c_2 = g_2^{a_1} g_4^{-d_1};\ c_3 = g^{a_2} h^{-d_2}.$$

2.4.6 Cryptographic signatures derived from PoK

In electronic voting, the public board is trusted as a public, append-only shared dataset. However, it does not mean that the data available in the public board (*e.g.*, the ballots) are legitimate. To prevent ballot stuffing and, more generally, to authenticate the data on the public board, the main strategy is to use signatures. Just as for encryption schemes, we denote $\mathscr{S}$ and $\mathscr{P}$ the sets of the secret and public keys. With these notations, a public key signature scheme can be

$\mathrm{Exp}^{\mathsf{suf\text{-}cma}}(\lambda, \mathbb{A})$

1 $\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$; $S \longleftarrow \varnothing$;
2 $m, \sigma \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathsf{Sign}}}(\mathsf{pk})$;
3 **if** $\mathrm{Verif}_{\mathsf{pk}}(\sigma, m) = 1$ **and** $(m, \sigma) \notin S$ **then return** 1 **else return** 0;

$\mathcal{O}_{\mathsf{Sign}}(m)$

1 $\sigma \longleftarrow \mathrm{Sign}_{\mathsf{sk}}(m)$;
2 $S \longleftarrow S \bigcup \{(m, \sigma)\}$;
3 **return** σ;

Figure 11: Strong unforgeability experiment

defined as a tuple $(\mathrm{Gen}, \mathrm{Sign}, \mathrm{Verif})$, such that, for all security parameter λ, $\mathrm{Gen}(\lambda)$ outputs a key pair $(\mathsf{pk}, \mathsf{sk})$, where pk is a public verification key and sk is the secret signature key. For such a key pair, we have

$$\forall m \in \{0,1\}^*, \mathrm{Verif}_{\mathsf{pk}}(\mathrm{Sign}_{\mathsf{sk}}(m), m) = 1.$$

The security of a signature scheme is characterized by its *unforgeability*: without the secret key, it should be unfeasible to forge a valid signature for a given message m. Numerous notions of unforgeability exists. For instance, we give Definition 13 which defines the strong unforgeability under adaptive chosen messages attacks. The unforgeability of a signature scheme in turn allows to guarantee the authenticity and the integrity of a message. Also, since a signature is publicly verifiable, it is usually non-repudiable, which gives a way to provide accountability.

Definition 13. *A signature scheme is SUF-CMA secure if, for all PPT adversary $\mathbb{A}$, the probability* $\Pr(\mathrm{Exp}^{\mathsf{suf\text{-}cma}}(\lambda, \mathbb{A}) = 1)$ *is negligible in λ, where* $\mathrm{Exp}^{\mathsf{suf\text{-}cma}}$ *is defined in Fig. 11.*

Schnorr signature. A classical way to obtain a signature scheme is to use a standard PoK of the signing key, where the message m is also included into the hash. This is the main idea of the Schnorr signature scheme [Sch89], where Gen generates an ElGamal key pair $(\mathsf{pk}, \mathsf{sk})$ while Sign is recalled in Algorithm 22. To verify a signature $\sigma = (c, a)$ for a message m with respect to a public verification key $\mathsf{pk} = (g, h)$, one can compute $d = \mathrm{hash}(g||h||c||m)$ and check that $c = g^a h^{-d}$.

Algorithm 22: Schnorr signature

Requires: A group G of prime order q
Two public elements (g, h)
A message $m \in \{0,1\}^*$
$\mathsf{sk} \in \mathbb{Z}_q$ s.t. $h = g^{\mathsf{sk}}$

1 $\alpha \xleftarrow{\$} \mathbb{Z}_q$; $c \longleftarrow g^\alpha$;
2 $d \longleftarrow \mathrm{hash}(g||h||c||m)$;
3 $a \longleftarrow \alpha + d\mathsf{sk}$;
4 **return** (c, a);

The Schnorr signature is widely used in practice, as it is extremely efficient and proven SUF-CMA secure in the ROM and under the DDH assumption [PS96]. However, up until 2008, the DSA/ECDSA was preferred as Schnorr's signature scheme was patented. For this reason, there are still a lot of protocols which use various versions of DSA/ECDSA.

Chapter 3

Security proofs in electronic voting

In modern cryptography, it is more and more important to provide a security proof that a protocol cannot be breached. In this thesis, we use cryptographic, hand-written proofs to link the security of a protocol (*e.g.*, its privacy or its verifiability) to a computational assumption (*e.g.*, the DDH assumption and the ROM). In this chapter, we present the main proof techniques that we use and we illustrate them by proving various results.

3.1 Cryptographic reductions and game hops

The main strategy to prove a security property is to exhibit a reduction to a known hard problem, such as factorization, discrete logarithm or DDH (or any hard problem). We already used this strategy in Section 2.2.2, where we proved that the ElGamal encryption scheme was IND-CPA under the DDH assumption. However, it is often too difficult to give a direct reduction to the main computational assumption. For this reason, we use game hops.

3.1.1 Game hops

Recall from Section 1.1.2 that we use game-based definitions to assess the security of a protocol. A game (or experiment) can be seen as an interactive Turing machine (ITM) that can interact with the adversary (seen as a PPT) by writing on its input tape and reading its outputs. Intuitively, the game models the behavior of the honest participants, while the adversary is allowed to impersonate the corrupted ones. For instance, consider the ZK-TCPA game in Algorithm 13, which will be our running example in this section. This algorithm defines the program of the game, seen as an ITM. However, we can see that the game must interact with an adversary, which is activated in lines 2, 5 and 10. During the same experiment, we consider that the adversary is *stateful*, *i.e.* that it keeps its memory between each activation. Depending on the actions of the adversary, an experiment outputs either 0 or 1: the adversary wins if the output is 1. This allows to model our security property. To decide whether the adversary wins or not, the game is supposed to have an unbounded computational power, which means that we do not demand that a game is a polynomial-time ITM. For a given game G, it may not be possible to interact with just *any* adversary $\mathbb{A}$. For instance, the adversary is supposed to output a well-formed plaintext at line 5, otherwise the game could not encrypt it. Therefore, there is always an implicit restriction on the adversary, which is supposed to give its outputs in the correct format. In this case, we say that $\mathbb{A}$ is an adversary *for* the game G. In addition to those implicit requirements, we may give some additional explicit restrictions, as they may not be obvious when just analyzing the

type of the variables. For instance, we ask that the set A output at line 4 should be a subset of $[1, n_T]$ of size at most t.

During the security analysis, we often refer to the *view* of the adversary. Formally, the view is a random variable over $\{0,1\}^*$ which consists of the concatenation of all the inputs that was given to the adversary. Intuitively, a game can be defined by the view that it gives to the adversary and how it computes the final output from the outputs of the adversary. When two views follow the same distribution, we say that they are the same. If two different games give the same views to the same adversary and compute their output in the same way, the output of both games will follow the same distribution. Similarly, if the views are not the same but computationally indistinguishable, the outputs in both games must also be computationally indistinguishable. This is a first example of a *game hop*, where we slightly modify a game into a more simple game, in such a way that the final output remains approximately the same.

An important specific case in game-based definition is the *decisional game*. A decisional game is parametrized by a random bit $b \in \{0,1\}$ which is chosen at the beginning of the game (sometimes, we may flip the bit latter, when it is really needed). Depending on the value of the bit, two different views can be given to the adversary, whose goal is to guess the value of b. In this case, we are interested in the *advantage* of the adversary, which is the distance between its probability to win and $1/2$ (as mentioned previously, a common alternative definition is to say that the advantage is twice this value, and we may use both definitions interchangeably). For instance, the ZK-TCPA game is a decisional game, and we say that a threshold encryption scheme is ZK-TCPA secure if no adversary can win with a non-negligible advantage.

To prove that a game cannot be won with a non-negligible probability (or advantage), a common strategy is to use several game hops, where we replace a game by another which is simpler to analyze. To argue that this transformation is legitimate, we use a game reduction. To present this notion, we use the following notation: if $\mathbb{A}$ is an adversary (seen as a Turing machine), the adversary $\mathbb{B}^{\mathbb{A}}$ is an adversary which interacts with $\mathbb{A}$ by writing and reading on its tapes. Since we consider only polynomial adversaries, $\mathbb{B}^{\mathbb{A}}$ can only make up to a polynomial number of queries to $\mathbb{A}$, and each query must use inputs of polynomial size. Although $\mathbb{B}^{\mathbb{A}}$ may read all the tapes of $\mathbb{A}$, a generic Turing machine can be arbitrarily obfuscated. Therefore, there is no constructive way for $\mathbb{B}^{\mathbb{A}}$ to learn anything else than $\mathbb{A}$'s outputs (*e.g.*, the intermediate values of the variables handled by $\mathbb{A}$). For this reason, we use the same notation as for black-box oracle queries, *e.g.*, $\mathbb{B}^{\mathcal{O}_{\mathsf{Dec}}}$. This black-box access assumption means that, given another adversary $\mathbb{A}'$, $\mathbb{B}$ could similarly interact with $\mathbb{A}'$ so that $\mathbb{B}$ can be seen as an ITM which makes some queries to a Turing machine. Then we say that there is a polynomial reduction from G_1 to G_2 if there exists a PPT $\mathbb{B}$, such that, for all adversary $\mathbb{A}$ that wins G_1 with a non-negligible probability (or advantage), $\mathbb{B}^{\mathbb{A}}$ wins G_2 with a non-negligible probability (or advantage). For instance, we gave an explicit reduction from IND-CPA to DDH and the other way around in Section 2.2.2.

Now, consider two games G_1 and G_2. Suppose that there exist several intermediate games $H_0, \cdots, H_t$ such that, for all i, there is a polynomial reduction from H_i to H_{i+1}, with $H_0 = G_1$ and $H_t = G_2$, where $t \in \mathbb{N}$ is some fixed integer (that does not depend on the security parameter λ). Then, by transitivity, there is a polynomial reduction from G_1 to G_2. This is the principle of game hops: here, each H_i is a game hop, and is obtained from H_{i-1} by introducing a little tweak. To see how the notion of game hops can be used to prove Theorem 1, see the proof of the theorem in Appendix A. Alternatively, we also provide some examples of game hops in the following section.

Algorithm 23: $\mathrm{Exp}^{\mathsf{ind\text{-}pa0}}(\lambda, \mathbb{A})$

1 $\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;
2 $m_0, m_1 \longleftarrow \mathbb{A}(\mathsf{pk})$;
3 $b \xleftarrow{\$} \{0,1\}$;
4 $C \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(m_b)$;
5 $\boldsymbol{C} \longleftarrow \mathbb{A}(C)$;
6 $\boldsymbol{m} \longleftarrow (\mathrm{Dec}_{\mathsf{sk}}(y))_{y \in \boldsymbol{C} \setminus \{C\}}$;
7 $b' \longleftarrow \mathbb{A}(\boldsymbol{m})$;
8 **if** $b = b'$ **then return** 1 **else return** 0;

Algorithm 24: $\mathrm{Exp}^{\mathsf{n\text{-}ind\text{-}pa0}}(\lambda, \mathbb{A})$

1 $\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;
2 $(\nu_{0,k}, \nu_{1,k})_{k=1}^{n} \longleftarrow \mathbb{A}(\mathsf{pk})$;
3 $b \xleftarrow{\$} \{0,1\}$;
4 **for** $i = 1$ **to** n **do** $C_i \leftarrow \mathrm{Enc}_{\mathsf{pk}}(\nu_{b,i})$;
5 $\boldsymbol{C} \longleftarrow \mathbb{A}((C_i)_{i=1}^{n})$;
6 $M \longleftarrow \{\!\{(\mathrm{Dec}_{\mathsf{sk}}(C))_{C \in \boldsymbol{C} \setminus \{C_i, i \in [1,n]\}}\}\!\}$;
7 $b' \longleftarrow \mathbb{A}(M)$;
8 **if** $b = b'$ **then return** 1 **else return** 0;

3.1.2 The hybrid lemma

The polynomial reduction proof strategy was notably used in [BS99] to prove the equivalence between two notions of non-malleability, by giving a succession of polynomial reductions NM-CPA $\Longrightarrow$ SNM-CPA $\Longrightarrow$ IND-PA0 $\Longrightarrow$ NM-CPA, where SNM-CPA is some equivalent formulation of the NM-CPA security while IND-PA0 security is defined in Definition 14. This notion of security is extremely interesting in electronic voting because it is equivalent to a weak version of privacy, captured by Algorithm 24. While this game is not enough to define privacy, it is still interesting to consider as a minimal requirement. Now, it is notable that Algorithms 23 and 24 are similar: the main difference is that the adversary can choose an arbitrary number of pairs at line 2, instead of one in the IND-PA0 game. The equivalence between these two games is a good opportunity to introduce the hyrid lemma.

Definition 14 (IND-PA0 [BS99])**.** *An encryption scheme* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ *is IND-PA0 secure if, for all PPT* $\mathbb{A}$*,* $\mathbb{A}$ *wins the IND-PA0 game (defined in Algorithm 23) with a negligible advantage.*

Lemma 1 ([BS99])**.** *An encryption scheme is NM-CPA if and only if it is IND-PA0.*

To prove that IND-PA0 (and, therefore, NM-CPA security) is equivalent to n-IND-PA0, the natural strategy is to consider n game hops $H_0, \cdots, H_n$, where H_0 is the IND-PA0 game (and hence does not count as a game hop) while H_n is the n-IND-PA0 game. Game H_i is defined as in Algorithm 24, except that the adversary can choose up to $i+1$ pairs of plaintexts at line 2. Therefore, H_i can be seen as a *hybrid* between the IND-PA0 game (where the adversary can only choose one pair) and the n-IND-PA0 game (where the adversary can choose an arbitrary number of pairs). With such hybrids, it is possible to show that, for all i, if there is an adversary $\mathbb{A}$ which wins H_{i+1} with some non-negligible probability, then there exists an adversary $\mathbb{B}$ that wins H_i with a non-negligible probability (see Lemma 2).

Lemma 2. *For* $i \in \mathbb{N}$*, let* H_i *be the game defined in Algorithm 24, except that the adversary is restricted to* $n \leq i+1$ *at line 2. Suppose that* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ *is IND-PA0. Then, for* $i \in \mathbb{N}$*, for all adversary PPT* $\mathbb{A}_{i+1}$*, there exists a PPT* $\mathbb{A}_i$ *and a negligible function* μ_i *such that*

$$|\Pr(H_i(\lambda, \mathbb{A}_i) = 1) - \Pr(H_{i+1}(\lambda, \mathbb{A}_{i+1}) = 1)| \leq \mu_i.$$

Proof. Let $i \in \mathbb{N}$ and let $\mathbb{A}_{i+1}$ be some PPT adversary for H_{i+1}. We construct $\mathbb{A}_i$ as follows. First, $\mathbb{A}_i$ gets pk from H_i and forwards it to $\mathbb{A}_{i+1}$ which answers with $i+2$ pairs $(\nu_{0,1}, \nu_{1,1}), \cdots, (\nu_{0,i+2}, \nu_{1,i+2})$ (when $\mathbb{A}_{i+1}$ uses less pairs, $\mathbb{A}_i$ can use them in H_i and thus a

perfect simulation of H_{i+1} to $\mathbb{A}_{i+1}$; therefore we focus on the specific case where $\mathbb{A}_{i+1}$ uses exactly $i+2$ pairs). $\mathbb{A}_i$ can only play $i+1$ of them in H_i, therefore $\mathbb{A}_i$ does not play the last one. At this point, $\mathbb{A}_i$ gets $i+1$ encryptions from game H_i but has to show $i+2$ ciphertexts to $\mathbb{A}_{i+1}$. For this purpose, $\mathbb{A}_i$ completes $\mathbb{A}_i$'s view with a random ciphertext C_{i+2} of a known (random) plaintext ν. This way, $\mathbb{A}_i$ gets $\boldsymbol{C}$ from $\mathbb{A}_{i+1}$ and plays this in H_i. It gets M from H_i, removes ν from M and adds $\nu_{0,i+2}$ instead. Then, it sends the resulting multiset to $\mathbb{A}_{i+1}$. Finally, $\mathbb{A}_i$ returns $\mathbb{A}_{i+1}$'s output.

Reduction to IND-PA0. Now, we show that there exists a negligible function μ_i which suits the conclusion of the lemma. For this purpose, we consider an adversary $\mathbb{B}$ for the IND-PA0 game. This adversary will interact with $\mathbb{A}_{i+1}$ by simulating H_{i+1}. First, $\mathbb{B}$ gets pk from the IND-PA0 game and forwards this to $\mathbb{A}_{i+1}$ which answers with $(\nu_{0,j}, \nu_{1,j})_{j=1}^{i+2}$. To simulate H_{i+1}, $\mathbb{B}$ picks a random bit $b \in \{0,1\}$ and compute $C_j = \mathrm{Enc}_{\mathsf{pk}}(\nu_{b,j})$ for $j \in [1, i+1]$. However, for the last ciphertext, $\mathbb{B}$ picks a random plaintext ν and sends $(\nu, \nu_{b,i+2})$ in the IND-PA0 game which gives back the ciphertext C_{i+2}. Then $\mathbb{B}$ give those $i+2$ ciphertexts to $\mathbb{A}_{i+1}$ which answers with $\boldsymbol{C}$. Then $\mathbb{B}$ plays this in the IND-PA0 game which gives back the decryptions $\boldsymbol{m}$. Finally, $\mathbb{B}$ sends $\{\!\{\boldsymbol{m}\}\!\} \uplus \{\!\{(\nu_{0,j})_{j=1}^{i+2}\}\!\}$ to $\mathbb{A}_{i+1}$ which answers with b'. If $b' = b$, $\mathbb{B}$ returns 1 to claim that the IND-PA0 encrypted $\nu_{b,i+2}$; otherwise $\mathbb{B}$ returns 0 to claim that the IND-PA0 encrypted ν.

Now, remark that when the IND-PA0 game encrypts $\nu_{b,i+2}$, $\mathbb{B}$ plays a perfect simulation of H_{i+1} to $\mathbb{A}_{i+1}$, and therefore $\mathbb{B}$ outputs 1 with probability $p_{i+1} = \Pr(H_{i+1}(\lambda, \mathbb{A}_{i+1}) = 1)$. On the other hand, when the IND-PA0 game encrypts ν, $\mathbb{B}$ plays $\mathbb{A}_i$'s simulation of H_{i+1} to $\mathbb{A}_{i+1}$. However, it must outputs 0 to win the IND-PA0 game, and thus wins with probability $1-p_i$, where $p_i = \Pr(H_i(\lambda, \mathbb{A}_i) = 1)$. Therefore, $\mathbb{B}$'s advantage in the IND-PA0 game is $\mu_i = \frac{1}{2}|p_i - p_{i+1}|$. □

By the triangular inequality, it follows that for all PPT $\mathbb{A}$ against n-IND-PA0, there exists a PPT $\mathbb{B}$ and some negligible functions $\mu_1, \cdots, \mu_n$ such that

$$\left|\Pr(\mathrm{Exp}^{\mathsf{n\text{-}ind\text{-}pa0}}(\lambda, \mathbb{A}) = 1) - \frac{1}{2}\right| \leq \left|\Pr(\mathrm{Exp}^{\mathsf{ind\text{-}pa0}}(\lambda, \mathbb{B}) = 1) - \frac{1}{2}\right| + \sum_{i=1}^{n} \mu_i.$$

However, an upsetting fact is that a sum of polynomially many negligible functions is not always negligible. Of course, this is the case when there exists a negligible function μ such that $\mu_i \leq \mu$ for all $i \in \mathbb{N}$, and also if n is bounded by some constant. However, suppose that $n = \lambda$ and consider the example $\mu_i = 2^i/2^\lambda$ for all i. Clearly, for any $i \in \mathbb{N}$, μ_i is negligible; however, $\sum_{i=1}^{n} \mu_i$ is not negligible. Therefore, Lemma 2 by itself is not sufficient to conclude: for this purpose, we need the hybrid lemma. In this thesis, we use the version of [MF21, Theorem 3.17], that we restate into Theorem 2, which is better suited for game-based definitions. It is not clear whether our formulation is equivalent to or a consequence of [MF21, Theorem 3.17]. For this reason, we prove Theorem 2 in Appendix B.

In this theorem, we consider the practical use case of the hybrid argument when, given two games G_1 and G_2, we want to show that there exists a polynomial reduction from G_1 to G_2. The idea is that the adversary in G_1 can freely choose a (at most polynomial) parameter n while, in G_2, n is restricted to some small constant (*e.g.*, 1). Then the usual approach is to construct a succession of *hybrids* $(H_i)_{i\in\mathbb{N}}$ such that $H_0 = G_2$ (more precisely, see condition 1), and argue that H_n is indistinguishable from G_1, provided that n is large enough (condition 2). The hybrids must be pairwise similar, *i.e.* a simulation of H_{i+1} can be obtained from H_i in polynomial time (condition 3). In addition, we need to argue that the simulation is actually indistinguishable from the real H_{i+1}. For this purpose, we use a computational assumption, which is captured by the condition 4, where we suppose that there exists a decisional game G – in which the

adversary has to guess a bit $b \in \{0,1\}$ – that cannot be won with a non-negligible advantage. To prove that the simulation is indistinguishable from the real H_{i+1}, we exhibit a polynomial reduction to the computational assumption; however, for the conclusion to hold, we need this reduction to be "uniform": the same reduction is applied when proving the indistinguishability of any two hybrids. In addition, we also need the reduction to be "perfect": we construct an explicit adversary $\mathbb{B}$ for game G such that, given i, $\mathbb{B}$ plays a perfect simulation of H_{i+1} to $\mathbb{A}_{i+1}$ when $b = 1$ in G and $\mathbb{A}_i$'s simulation when $b = 0$ in G. These two additional requirements are expressed by condition 5. When all these conditions are gathered, the hybrid lemma gives the desired conclusion.

Theorem 2 (The hybrid lemma)**.** *Let G_1 and G_2 two games. We consider a sequence of games $(H_i)_{i\in\mathbb{N}}$ which are hybrids between G_1 and G_2. With these notations, assume that the following conditions are met:*

1. *For all PPT $\mathbb{A}$, for all security parameter λ, $\Pr(G_2(\lambda,\mathbb{A}) = 1) = \Pr(H_0(\lambda,\mathbb{A}) = 1)$.*
2. *For all PPT $\mathbb{A}$ for game G_1, there exists a polynomial $n_{\mathbb{A}}$ such that, for all $\lambda \in \mathbb{N}$, $\Pr(H_{n_{\mathbb{A}}(\lambda)}(\lambda,\mathbb{A}) = 1) = \Pr(G_1(\lambda,\mathbb{A}) = 1)$.*
3. *There exists a polynomial P and two transformation T and T' such that, given any PPT adversary $\mathbb{A}_{i+1}$ (resp. $\mathbb{A}_i$) for game H_{i+1} (resp. H_i), $\mathbb{A}_i = T(\mathbb{A}_{i+1})$ (resp. $\mathbb{A}_{i+1} = T'(\mathbb{A}_i)$) is an adversary for game H_i (resp. H_{i+1}) which makes at most $P(\lambda)$ additional transitions.*
4. *There exists a game G which depends on a parameter $b \in \{0,1\}$ such that, for all PPT adversary $\mathbb{B}$, $\varepsilon_{\mathbb{B}} = 2|\Pr(G(\lambda,\mathbb{B}) = 1) - 1/2|$ is negligible in λ.*
5. *There exists a PPT $\mathbb{B}$ such that, for all $i \in \mathbb{N}$ and all PPT $\mathbb{A}_{i+1}$ for game H_{i+1} (which in turns defines a PPT $\mathbb{A}_i$ for H_i), we have $\Pr(G(\lambda,\mathbb{B}^{\mathbb{A}_{i+1}}(i)) = 1 \mid b = 0) = \Pr(H_i(\lambda,A_i) = 1)$ and $\Pr(G(\lambda,\mathbb{B}^{\mathbb{A}_{i+1}}(i)) = 1 \mid b = 1) = \Pr(H_{i+1}(\lambda,A_{i+1}) = 1)$.*

Then, for all PPT $\mathbb{A}_1$, there exists a PPT $\mathbb{A}_2$ and a PPT $\mathbb{B}$ such that

$$|\Pr(G_1(\lambda,\mathbb{A}_1) = 1) - \Pr(G_2(\lambda,\mathbb{A}_2) = 1)| \leq n_{\mathbb{A}_1}\varepsilon_{\mathbb{B}}.$$

The hybrid lemma is a fundamental result in cryptography. In particular, it allows to prove Lemma 3, which is a key step in the proof of privacy for the Helios-like voting systems. For a complete proof of privacy of Helios, we refer to [BPW12, Theorem 3] or [BCG+15b, Theorem 2].

Lemma 3. *An encryption scheme is IND-PA0 if and only if, for all PPT adversary $\mathbb{A}$, the advantage $\left|\Pr(\mathrm{Exp}^{n\text{-}ind\text{-}pa0}(\lambda,\mathbb{A}) = 1) - 1/2\right|$ is negligible in λ.*

Proof. First, it is clear that n-IND-PA0 implies IND-PA0 since $\mathrm{Exp}^{\mathsf{ind\text{-}pa0}}$ is a special case of $\mathrm{Exp}^{\mathsf{n\text{-}ind\text{-}pa0}}$ with $n = 1$. Conversely, suppose that an encryption scheme $(\mathrm{Gen},\mathrm{Enc},\mathrm{Dec})$ is IND-PA0 secure. Then we consider the game hops $(H_i)_{i\in\mathbb{N}}$ defined as in Algorithm 24, except that n is restricted to be at most $i+1$ at line 2. Then, with $G_1 = \mathrm{Exp}^{\mathsf{n\text{-}ind\text{-}pa0}}$ and $G_2 = H_0$, we have:

1. For all $\mathbb{A},\lambda$, $\Pr(G_2(\lambda,\mathbb{A}) = 1) = \Pr(H_0(\lambda,\mathbb{A}) = 1)$.
2. For all PPT $\mathbb{A}$, there exists a polynomial $n_{\mathbb{A}}$ such that $\mathbb{A}$ can make at most $n_{\mathbb{A}}$ transitions. In particular, $\mathbb{A}$ cannot output more than $n_{\mathbb{A}}$ pairs at line 2 and therefore $\Pr(H_{n_{\mathbb{A}}(\lambda)}(\lambda,\mathbb{A}) = 1) = \Pr(G_1(\lambda,\mathbb{A}) = 1)$ for all $\lambda \in \mathbb{N}$.

3. Let $\mathbb{A}_{i+1}$ be an adversary for game H_{i+1}. In the proof of Lemma 2, we construct an adversary $\mathbb{A}_i$ for H_i which only computes an additional encryption. Conversely, given $\mathbb{A}_i$, and adversary for game H_i, $\mathbb{A}_{i+1} = \mathbb{A}_i$ is an adversary for game H_{i+1}.

4. By assumption, we can use $G = \mathrm{Exp}^{\mathsf{ind\text{-}pa0}}$.

5. We constructed such a $\mathbb{B}$ in the proof of Lemma 2.

By the hybrid lemma, for all PPT adversary $\mathbb{A}$ which wins $\mathrm{Exp}^{\mathsf{n\text{-}ind\text{-}pa0}}$ with a non-negligible advantage, there exists a PPT adversary $\mathbb{B}$ which wins H_0 with a non-negligible advantage. Since H_0 is clearly equivalent to the IND-PA0 game, this concludes the proof. □

3.2 Known results in the random oracle model

When designing a polynomial reduction in the random oracle model, we construct an adversary $\mathbb{B}$ that interacts with another adversary $\mathbb{A}$ by simulating a random oracle. This leads to surprisingly powerful results which can be used to prove various security properties.

3.2.1 Extracting a witness from a proof of knowledge

One of the main use cases of the ROM is to prove the security of standard ZKP, *i.e.* ZKP obtained from a disjunctive proof or a proof of knowledge of a preimage, using the Fiat-Shamir transformation. Indeed, we argued in Section 2.3.4 that the ROM allows to prove that standard ZKP are computationally sound while still revealing no useful information to a PPT adversary. However, the example of the Schnorr signature (see Algorithm 22) illustrates that the computational soundness is not sufficient in some cases. For signatures and authentication, we not only need to make sure that the statement is true, but also that the prover knows the corresponding witness. Fortunately, in the case of PoK of a preimage of z by a homomorphism φ, the ROM not only allows to prove that that $z \in \varphi(G)$ with overwhelming probability, but also that the prover "knows" a preimage of z. In other words, for all group homomorphism $\varphi : G \to H$, where the order of H does not have any divisor smaller than 2^λ, there exists a PPT extractor which, given $z \in H$ and a PPT $\mathbb{A}$ that outputs a valid PoK of a preimage of z with some non-negligible probability, is able to *extract* a preimage of z from $\mathbb{A}$.

The proof method to build this extractor uses the so-called *forking lemma* (or rewinding lemma), introduced in [PS96]. First, consider an adversary $\mathbb{A}$ in the ROM which outputs a valid proof $\pi = (c, a)$ with some non-negligible probability. Then $\mathbb{A}$ must have made a query to the random oracle with the input c (technically, $\mathrm{pre}||c$), otherwise the challenge d would be uniformly random and the verification equation $c = \varphi(a)z^{-d}$ would hold with a negligible probability. Then we can consider a PPT $\mathbb{B}$ which interacts with $\mathbb{A}$ by using the *rewinding paradigm*: if several independent copies of $\mathbb{A}$ are called several times with the same inputs and the same random tape, then they must return the same outputs. Hence, by using two copies of $\mathbb{A}$ and feeding them with exactly the same inputs, $\mathbb{B}$ can create a situation where both copies make an oracle query with the commitment c as an input. At this point, $\mathbb{B}$ gives a different challenge to both copies, in the hope that they would output a valid answer for the same commitment but two different challenges, allowing $\mathbb{B}$ to extract a preimage using the special soundness property. However, since $\mathbb{B}$ gave a different answer to the copies, they can actually *fork* at this point; hence the name of the lemma. Using this lemma in the specific case of the Schnorr signature, Pointcheval and Stern showed that it is indeed possible for $\mathbb{B}$ to *extract* a preimage of z in polynomial time.

This extractability result can be generalized as soon as the original Σ-protocol had the special soundness property (see, for instance, [BPW12, Theorem 1]).

Non-malleability in practice. Following the same proof strategy, it is possible to show that a simple transformation of the ElGamal encryption scheme is sufficient to provide NM-CPA security, as stated in Lemma 4. This is an interesting result in electronic voting, as we already stated that NM-CPA is a good notion of security that is necessary for privacy (see Lemma 3). Although there exists other strategies that do not rely on the ROM to create non-malleable variants of the ElGamal encryption (*e.g.*, [CS98]), the "Enc+PoK" paradigm is widely preferred because of its efficiency.

Lemma 4 ([BPW12])**.** *Let* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ *be an IND-CPA encryption scheme and, for all* pk*, let* $A_{\mathsf{pk}}(C, m, r)$ *a standard PoK algorithm that, given a ciphertext* C*, a plaintext* m *and a randomness* r *such that* $\mathrm{Enc}_{\mathsf{pk}}(m, r) = C$*, produces a PoK for the language* $\mathrm{Enc}_{\mathsf{pk}}(\mathscr{P} \times \mathscr{R})$*, where* $\mathscr{P}$ *is the plaintext space while* $\mathscr{R}$ *is the randomness space. Suppose that* A_{pk} *is obtained from a zero knowledge* Σ*-protocol that has special soundness, using the strong Fiat-Shamir transformation (i.e. the hash must at least contain the ciphertext* C*, the public key* pk *and the commitment* c*). Then, the encryption scheme* $(\mathrm{Gen}, \mathrm{Enc}', \mathrm{Dec}')$ *is NM-CPA secure, where* $\mathrm{Enc}'_{\mathsf{pk}}(m, r)$ *returns the pair* (C, π) *with* $C = \mathrm{Enc}_{\mathsf{pk}}(m, r)$ *and* $\pi = A_{\mathsf{pk}}(C, m, r)$ *while* $\mathrm{Dec}'_{\mathsf{sk}}(C, \pi)$ *returns* $\perp$ *if* π *is invalid and* $\mathrm{Dec}_{\mathsf{sk}}(C)$ *otherwise.*

3.2.2 Good practices for non-interactive proofs

Non-interactive proofs are ubiquitous in electronic voting, yet their security is not always understood properly. Indeed, recall that all the arguments of Section 2.3.4 (and also Section 3.2.1) assumed that both the NP language $\mathscr{L}$ and the element z (supposedly in $\mathscr{L}$) were fixed and could not be chosen by the adversary. However, this is not the case in electronic voting. For instance, consider the example of a voter who produces a ballot that contains an encryption C and a ZKP that C encrypts a valid voting option. Then the statement "C is valid" actually depends on C, which means that a malicious voter has an extra degree on liberty when creating the proof. Namely, suppose that we use a weak Fiat-Sharmir transformation where only the commitment is hashed to obtain the challenge. Then one could very well compute a random commitment c, deduce the challenge d, choose a random answer a and compute $z = (\varphi(a)/c)^{1/d}$. Then the "proof" $\pi = (c, a)$ will satisfy a verifier who would compute $d = \mathrm{hash}(c)$ then check that $c = \varphi(a)z^{-d}$; yet, z is uniformly random (since c is random) and thus not necessarily in $\varphi(G)$. This illustrates the difference between what a PPT adversary can do with a chosen (hence potentially trapdoored) z compared to with a given z (for which a cannot be extracted because of φ's one-wayness).

"Strong" Fiat-Shamir transformation. Interestingly, it was revealed in [BPW12] that the ZKP used in Helios were vulnerable to this attack. Worse, it was even revealed that a coalition of $t + 1$ talliers were able to create a ciphertext for any chosen plaintext, and forge a fake but valid-looking ZKP that this ciphertext encrypts a valid voting option. Although it is generally assumed that up to t talliers may be corrupted, this trust assumption is only made for privacy and not for verifiability. Yet, when a homomorphic tally is used, this allows to undetectably add (or remove) any arbitrary number of votes to any candidate, hence rigging the result. For this reason, the authors introduced the "strong" Fiat-Shamir transformation, in which the challenge is not only obtained from the commitment, but also from the ciphertext; *i.e.* $d = \mathrm{hash}(z||c)$ instead of $d = \mathrm{hash}(c)$. In what follows, we show that this is still not enough: while z is indeed fixed, this is not the case for φ.

How to forge a fake ZKP, again. The solution of [BPW12] is not sufficient in the context of electronic voting. More precisely, although the developers of Belenios [CGG19] used the "strong" Fiat-Shamir transformation as recommended, we found a way for the corrupted talliers to forge a fake but valid-looking ZKP for a false statement. This allows them to cast a ballot for an invalid voting option, which either breaks eligibility or prevents the result from being tallied. We presented this attack at E-Vote-ID as a short paper [CGY20]. To fix ideas, consider the case of an exponential ElGamal encryption in a group G of prime order q, with a base a. To simplify further, assume that a is an element of the public key; for instance, if $\mathsf{pk} = (g, h)$, suppose that $a = g$ (a similar attack exists when $a = h$ or when a can be chosen freely by the attacker; if a is fixed in advance or provably random, then a similar attack is possible by taking $\gamma = a$ at Step 1). With this in mind, recall that the encryption of a bit $b \in \{0, 1\}$ given a public key $\mathsf{pk} = (g, h)$ is obtained by choosing a random $r \in \mathbb{Z}_q$ and computing $x = g^r$ and $y = g^b h^r$. Now, to prove that (x, y) is an encryption of either 0 or 1, assume that we use the "strong" Fiat-Shamir transformation from [BPW12]: a valid proof π consists of a tuple $(c_{0,x}, c_{0,y}, c_{1,x}, c_{1,y}, d_0, d_1, a_0, a_1)$ such that, with $d = \mathrm{hash}(x||y||c_{0,x}||c_{0,y}||c_{1,x}||c_{1,y})$, we have $d_0 + d_1 = d$ while $c_{i,x} = g^{a_i} x^{-d_i}$ and $c_{i,y} = h^{a_i}(y/g^i)^{-d_i}$ for all $i \in \{0, 1\}$. Then, we explain how the attacker can create an encryption (x, y) of any chosen plaintext $m \in \mathbb{Z}_q$, and forge a fake but valid-looking ZKP π w.r.t. (x, y). To perform the attack, one must corrupt all the participant of the key generation protocol (*i.e.* the talliers), and proceed as follows:

1. Choose a random group generator $\gamma \in G \backslash \{1\}$.

2. Pick some random scalars $\ell_x, \ell_y, r_{0x}, r_{0y}, r_{1x}, r_{1y} \in \mathbb{Z}_q$ and compute $x = \gamma^{\ell_x}$, $y = \gamma^{\ell_y}$, $c_{0,x} = \gamma^{r_{0x}}$, $c_{0,y} = \gamma^{r_{0y}}$, $c_{1,x} = \gamma^{r_{1x}}$ and $c_{1,y} = \gamma^{r_{1y}}$.

3. Compute $d = \mathrm{hash}(x||y||c_{0,x}||c_{0,y}||c_{1,x}||c_{1,y})$.

At this point, the ciphertext (x, y) and the commitments are fixed, but not the public encryption key pk. The idea is to use it as some extra degree of liberty, by looking for g and h of the form $g = \gamma^{\ell_g}$ and $h = \gamma^{\mathsf{sk}\ell_g}$. However, for (x, y) to encrypt the desired plaintext $m \notin \{0, 1\}$, we need the secret key sk to respect the equation $m\ell_g = \ell_y - \mathsf{sk}\ell_x$. Finally, to forge our valid ZKP, we need to find d_0, d_1, a_0, a_1 that satisfy the verifying equations. By using the trapdoors set at Step 2, we can use the vector space structure of G and express all the desired equations in $\mathbb{Z}_q$, using the discrete logarithm in base γ when necessary. This leads to the following system, where the unknowns are in bold while the known values are in blue:

$$\begin{cases} m\boldsymbol{\ell_g} = \ell_y - \mathbf{sk}\ell_x \\ d = \boldsymbol{d_0} + \boldsymbol{d_1} \\ r_{0x} = \boldsymbol{a_0}\boldsymbol{\ell_g} - \boldsymbol{d_0}\ell_x \\ r_{0y} = \boldsymbol{a_0}\mathbf{sk}\boldsymbol{\ell_g} - \boldsymbol{d_0}\ell_y \\ r_{1x} = \boldsymbol{a_1}\boldsymbol{\ell_g} - \boldsymbol{d_1}\ell_x \\ r_{1y} = \boldsymbol{a_1}\mathbf{sk}\boldsymbol{\ell_g} - \boldsymbol{d_1}(\ell_y - \boldsymbol{\ell_g}) \end{cases} \iff \begin{cases} m\boldsymbol{\ell_g} = \ell_y - \mathbf{sk}\ell_x \\ d = \boldsymbol{d_0} + \boldsymbol{d_1} \\ r_{0x} = \boldsymbol{a_0}\boldsymbol{\ell_g} - \boldsymbol{d_0}\ell_x \\ r_{0y} - \mathbf{sk}r_{0x} = \boldsymbol{d_0}(\mathbf{sk}\ell_x - \ell_y) \\ r_{1x} = \boldsymbol{a_1}\boldsymbol{\ell_g} - \boldsymbol{d_1}\ell_x \\ r_{1y} - \mathbf{sk}r_{1x} = \boldsymbol{d_1}(\mathbf{sk}\ell_x - \ell_y + \boldsymbol{\ell_g}). \end{cases}$$

Since this system is quadratic, it is a priori not trivial to solve. However, assuming that the

denominators do not cancel (which happens with negligible probability), we have:

$$\begin{cases} \mathsf{sk} = \dfrac{\ell_y - m\boldsymbol{\ell_g}}{\ell_x} \\ d = \boldsymbol{d_0} + \boldsymbol{d_1} \\ \boldsymbol{a_0} = \dfrac{r_{0x} + \boldsymbol{d_0}\ell_x}{\boldsymbol{\ell_g}} \\ \boldsymbol{d_0} = \dfrac{r_{0y} - \mathsf{sk}r_{0x}}{\mathsf{sk}\ell_x - \ell_y} \\ \boldsymbol{a_1} = \dfrac{r_{1x} + \boldsymbol{d_1}\ell_x}{\boldsymbol{\ell_g}} \\ \boldsymbol{d_1} = \dfrac{r_{1y} - \mathsf{sk}r_{1x}}{\mathsf{sk}\ell_x - \ell_y + \boldsymbol{\ell_g}}. \end{cases}$$

Hence everything can be expressed as a function of ℓ_g, provided that $d = d_0 + d_1$. This last equation can then be rewritten as

$$d = \frac{r_{0y} - \mathsf{sk}r_{0x}}{\mathsf{sk}\ell_x - \ell_y} + \frac{r_{1y} - \mathsf{sk}r_{1x}}{\mathsf{sk}\ell_x - \ell_y + \ell_g} = \frac{r_{0y} - \frac{\ell_y - m\ell_g}{\ell_x}r_{0x}}{\ell_y - m\ell_g - \ell_y} + \frac{r_{1y} - \frac{\ell_y - m\ell_g}{\ell_x}r_{1x}}{\ell_y - m\ell_g - \ell_y + \ell_g}, \text{ hence}$$
$$\ell_g d = \frac{\frac{\ell_y - m\ell_g}{\ell_x}r_{0x} - r_{0y}}{m} + \frac{r_{1y} - \frac{\ell_y - m\ell_g}{\ell_x}r_{1x}}{1-m}.$$

Now that we finally have a linear equation, we can proceed as follows:

4. Compute $\ell_g = \left(\frac{r_{0x}\ell_y - r_{0y}\ell_x}{m\ell_x} + \frac{r_{1y}\ell_x - r_{1x}\ell_y}{(1-m)\ell_x}\right)\left(d + \frac{r_{0x}}{\ell_x} + \frac{mr_{1x}}{(m-1)\ell_x}\right)^{-1}$ and $g = \gamma^{\ell_g}$.

5. Compute $\mathsf{sk} = \frac{\ell_y - m\ell_g}{\ell_x}$ and $h = g^{\mathsf{sk}}$. Set $\mathsf{pk} = (g, h)$.

6. Deduce d_0, d_1 and then a_0 and a_1 from the above equations.

7. Return the ciphertext (x, y) and the ZKP $\pi = (c_{0x}, c_{0y}, c_{1x}, c_{1y}, d_0, d_1, a_0, a_1)$.

This attack allows to forge a single ballot that contains any desired number of voices for a specific candidate. For instance, if the talliers want Alice to win, they can give her any number of additional voices; if they want her not to win, they can give her a negative number of voices. Alternatively, they can also choose a large value of m, so that the tally would be impossible since it would require to solve the discrete logarithm problem. One would argue that those attacks require to freely choose the group generator g, while there is no reason to allow the talliers to do this. Since g must not have any specificity except being uniformly random, it can be obtained by a public coin protocol. In practice, g is actually fixed and determined by the group specification. When g is provably random or fixed, the authorities can no longer choose ℓ_g as in Step 4 and hence cannot forge a fake ZKP for the ciphertext (x, y). However, they are still able to choose sk, which allows to create an encryption of some random (not chosen) $m \in \mathbb{Z}_q$ and forge a fake ZKP that $m \in \{0, 1\}$. Therefore, the soundness of the ZKP is still lost and it is still possible to prevent the tally from being computed without being blamed.

Strong Fiat-Shamir transformation. To obtain a PoK, we recommend that the challenge be obtained from a binding description of the homomorphism $\varphi : G \to H$, the element z for which we want to prove the knowledge of a preimage and the commitment. When those three are fixed, the soundness of the Fiat-Shamir transformation is proven in the ROM; however, there

is no guarantee if, for instance, H or G are allowed to change. In the case of the ElGamal encryption scheme, it is therefore necessary to include the ciphertext and the public key in the hash (along with the commitment). If exponential ElGamal is used with some base $a \notin \{g, h\}$, then a should be included as well. Finally, one can even include a description of G just to be on the safe side (even if they cannot think of a clever way to use the same pk – seen as a bitstring – in another group).

To conclude this section, we give our own definition of the strong Fiat-Shamir transformation. The original "strong" transformation was too weak, as it allowed a PPT adversary to forge fake ZKP for false statements by using an attack scenario which was not anticipated. Therefore, it is possible that our solution may also be vulnerable to an attack in the future. For this reason, we give a clear context in which our transformation is secure, and which is suitable for electronic voting. First, to fix ideas, we give a definition of Σ-protocols in Definition 15.

Definition 15. *Let $\mathscr{W}, \mathscr{L}$ be two sets parametrized by a security parameter λ. (For the ease of the notations, we drop the dependency on λ.) A Σ-protocol for the relation $\mathcal{R} = \mathscr{W} \times \mathscr{L}$ is a tuple of PPT algorithms* $(\mathrm{Com}, \mathrm{Ans}, \mathrm{Ver})$.

It has ***correctness*** *if, for all $(w, z) \in \mathcal{R}$ (also denoted $w\mathcal{R}z$), for all $d \in [0, 2^\lambda - 1]$, $\mathrm{Ver}(z, c, d, a) = 1$, where $c, \rho = \mathrm{Com}(z)$ and $a = \mathrm{Ans}(w, \rho, z, d)$.*

It is ***zero knowledge*** *if there exists a PPT simulator* Sim *such that, for all $(w, z) \in \mathcal{R}$ and all $d \in [0, 2^\lambda - 1]$, $(\tilde{c}, \tilde{a}) = \mathrm{Sim}(z, d)$ follows the same distribution as (c, a), where $c, \rho = \mathrm{Com}(z)$ and $a = \mathrm{Ans}(w, \rho, z, d)$.*

It is ***computationally sound*** *if there exists a negligible function μ such that, for all PPT adversary $\mathbb{A}$ and all $z \notin \mathcal{L}$, we have:*

$$\Pr(c \leftarrow \mathbb{A}(\lambda, z); d \xleftarrow{\$} [0, 2^\lambda - 1]; a \leftarrow \mathbb{A}(d); \mathrm{Ver}(z, c, d, a) = 1) \leq \mu(\lambda).$$

Then, to introduce the notion of strong Fiat-Shamir transformation, we must capture the fact that the NP language may be chosen by the adversary. For this purpose, we consider in Definition 16 a family of Σ-protocols which are parametrized by a public key pk. In electronic voting, pk is the public encryption key (and, if necessary, the base of the exponentiation for exponential ElGamal). However, we cannot consider just *any* family since proving the security of the strong Fiat-Shamir transformation in a too generic setting would be hard. Therefore, we restrict ourselves to a *uniform* case, as it is the case in general (see Definition 16). One of the consequences is that, *technically*, since we ask for the relation $\mathcal{R}_{\mathsf{pk}}$ to be efficiently decidable given only pk and λ, the parameter pk must somehow include an encoding of the group, otherwise one would not know how to compute an exponentiation. By uniform, we mean that the same algorithm allows to compute $\mathrm{Com}_{\mathsf{pk}}$, $\mathrm{Ans}_{\mathsf{pk}}$, $\mathrm{Ver}_{\mathsf{pk}}$ and the simulator $\mathrm{Sim}_{\mathsf{pk}}$. In addition, we also demand that the computational soundness is *uniform* (see the UCS game in Algorithm 25). This way, we do not have to worry about the sum of polynomially many negligible functions being potentially non-negligible. For the non-interactive proof to be "zero-knowledge" in the ROM, we also ask that if q is polynomial, then q independent commitment from Com, even with potentially different pk, are pairwise distinct, except with a negligible probability. Intuitively, this means that the simulation of a non-interactive proof in the ROM will be perfectly indistinguishable from the real proof, except with a negligible probability. (Indeed, recall that to simulate a proof, the simulator first chooses a random challenge d and a random answer a, then computes $c = \varphi_{\mathsf{pk}}(a) z^{-d}$, which follows the same distribution as a honestly generated commitment when $z \in \varphi_{\mathsf{pk}}(G)$. To be able to forge the proof, the simulator "assigns" the value d to $\mathrm{hash}(\mathsf{pk}||z||c)$, which may not be possible if c was already used in another simulated ZKP.)

Algorithm 25: $\mathrm{Exp}^{\mathsf{UCS}}(\lambda, \mathbb{A})$

Requires: A family of Σ-protocols $(\mathrm{Com}, \mathrm{Ans}, \mathrm{Ver})$ w.r.t. the languages $(\mathscr{L})_{\mathsf{pk}}$

1 $\mathsf{pk}, z, c \longleftarrow \mathbb{A}(\lambda)$;

2 $d \xleftarrow{\$} [0, 2^\lambda - 1]$;

3 $a \longleftarrow \mathbb{A}(d)$;

4 **if** $\mathrm{Ver}_{\mathsf{pk}}(z, c, d, a) = 1$ **and** $z \notin \mathscr{L}_{\mathsf{pk}}$ **then return** 1 **else return** 0;

Definition 16 (Uniform Σ-protocol)**.** *Let $(\mathcal{R})_{\mathsf{pk}}$ be a family of relations parametrized by a public key pk and the security parameter λ. (For all pk, we denote $\mathscr{L}_{\mathsf{pk}} = \{z \mid \exists w, w\mathcal{R}z\}$.) Similarly, let $(\mathrm{Com}, \mathrm{Ans}, \mathrm{Ver})_{\mathsf{pk}}$ be a family of Σ-protocols for those relations. Suppose that for all pk, $(\mathrm{Com}, \mathrm{Ans}, \mathrm{Ver})_{\mathsf{pk}}$ has correctness, zero knowledge and computational soundness. We say that this family is* uniform *if the following conditions are met:*

- *There is a common PPT algorithm $\mathcal{R}$ such that, given a public key pk, a security parameter λ and $w, z \in \{0,1\}^*$, allows to decide if $w\mathcal{R}_{\mathsf{pk}}z$;*
- *There exists a common PPT algorithm* Com *(resp.* Ans *and* Ver*) which, given pk, λ and z (resp. w, ρ, z, d and z, c, d, a), computes $\mathrm{Com}_{\mathsf{pk}}(z)$ (resp. $\mathrm{Ans}_{\mathsf{pk}}(w, \rho, z, d)$ and $\mathrm{Ver}_{\mathsf{pk}}(z, c, d, a)$);*
- *There exists a common PPT simulator* Sim *which given pk, λ, $z \in \mathscr{L}_{\mathsf{pk}}$ and $d \in [0, 2^\lambda - 1]$, returns $\mathrm{Sim}_{\mathsf{pk}}(z, d)$, where $\mathrm{Sim}_{\mathsf{pk}}$ is the simulator of $(\mathrm{Com}_{\mathsf{pk}}, \mathrm{Ans}_{\mathsf{pk}}, \mathrm{Ver}_{\mathsf{pk}})$;*
- *There exists a negligible function μ such that, for all non-uniform polynomial $\mathbb{A}$, the probability that $\mathbb{A}$ wins the UCS game (defined in Algorithm 25) is at most $\mu(\lambda)$.*
- *For all polynomial q, q independent commitments from* Com *are pairwise distinct, except with probability at most $q\mu$.*

Now, we argue that the standard Σ-protocol are actually *uniform*. Indeed, consider a family $\varphi_{\mathsf{pk}} : G \to H$ which is parametrized by a parameter pk, but for which there is a common polynomial-time algorithm to compute every φ_{pk} given pk and $g \in G$. (For instance, φ can be the encryption algorithm.) By contrast with the usual, we suppose that the order of H does not have any non-trivial divisor smaller than $2^{2\lambda}$ (instead of 2^λ, which is the case in general since we want to protect ourselves against the birthday paradox. In addition, we also consider that for all pk, φ_{pk} is non-trivial, which means that we may rule out some specific values of pk. We consider $(\mathrm{Com}, \mathrm{Ans}, \mathrm{Ver})_{\mathsf{pk}}$, the standard Σ-protocol for proving the knowledge of a preimage from ϕ_{pk} (defined in Section 2.3.2). Then $(\mathrm{Com}_{\mathsf{pk}}, \mathrm{Ans}_{\mathsf{pk}}, \mathrm{Ver}_{\mathsf{pk}})$ clearly comes from a common polynomial-time algorithm, the same goes for $\mathrm{Sim}_{\mathsf{pk}}$. In addition, recall that a standard Σ-protocol has the special soundness property. Therefore, whenever the adversary outputs pk, z, c such that $z \notin \varphi_{\mathsf{pk}}(G)$, then there exists at most one value $d \in [0, 2^\lambda - 1]$ for which a valid answer a can be found. Hence, the adversary wins the UCS game with probability at most $\mu(\lambda) \leq 2^{-\lambda}$, which is indeed independent from pk. Finally, since we assumed that for all pk, φ_{pk} is non-trivial, its image is a non-trivial sub-group of H, and therefore has a cardinality of at least $2^{2\lambda}$ (indeed, recall that the cardinality of a sub-group is a divisor of the order of the group, and that there is no non-trivial divisor smaller than $2^{2\lambda}$). Yet, the commitment algorithm consists of choosing a random $\alpha \in G$ and returning $c = \varphi_{\mathsf{pk}}(\alpha)$, so that c is a random element of $\varphi_{\mathsf{pk}}(G)$, chosen among at least $2^{2\lambda}$ possibilities. Therefore, q independent commitments can collide with

probability at most $1 - \prod_{i=0}^{q-1}(1 - i2^{-2\lambda})$. When q is at most polynomial, this is approximately $\frac{q(q-1)}{2^{2\lambda+1}} \leq q^2 2^{-2\lambda} \leq q2^{-\lambda} = q\mu$.

Now that we have fixed the context, we give Definition 17 which explains how to securely turn an interactive Σ-protocol into a non-interactive ZKP. Note that the security of this transformation is already included in the definition of a uniform Σ-protocol, for instance in the UCS game.

Definition 17 (Strong Fiat-Shamir transformation)**.** *Consider a uniform Σ-protocol* (Com, Ans, Ver) *for the family $\mathcal{R}_{\mathsf{pk}}$, and* $\text{hash} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ *a hash function. The strong Fiat-Shamir transformation of this protocol consists of the two following algorithms:*

- $\mathbf{Prove_{pk}(w, z)}$ **:** *Compute $c, \rho = \text{Com}_{\mathsf{pk}}(z)$, compute $d = \text{hash}(\mathsf{pk}||z||c)$ (modulo 2^λ) and return (z, c, a), where $a = \text{Ans}_{\mathsf{pk}}(w, \rho, z, d)$.*
- $\mathbf{Verif_{pk}(z, c, a)}$ **:** *Compute $d = \text{hash}(\mathsf{pk}, z, c)$ (modulo 2^λ) and return $\text{Ver}_{\mathsf{pk}}(z, c, d, a)$.*

Note that pk *must contains all the necessary informations to compute $\mathcal{R}_{\mathsf{pk}}$, Com_{pk}, Ans_{pk}, Ver_{pk} and Sim_{pk}. This typically includes the group generator and a specification of a group.*

3.3 Universally composable security

In electronic voting, we often consider complex protocols that are divided into several phases, whose security is assessed independently. However, recall how the provably secure Fiat-Shamir transformation (that required $\mathscr{L}$ and z to be fixed) became unsecure as soon as the adversary was able to choose $\mathscr{L}$ and z. This example shows that a protocol as a whole may be unsecure even when its components are individually secure. To address this, we use an universally composable framework in which the security of a protocol can be deduced from the security of its components.

3.3.1 Presentation of the framework

We use the SUC framework [CCL15], which is a simpler version of the universally composable framework of Canetti [Can01] (hence the acronym SUC). In this framework, a protocol is a fixed set of participants that are modeled as probabilistic polynomial ITM which interact with each other using common input / output tapes (*i.e.* communication tapes). Executing all the participants concurrently leads to an execution of the protocol, which is called a *process*. In turn, a process can invoke sub-processes, but with the same participants that must allocate a part of their memory and computation time for this. This way, several sub-processes can be run in parallel. However, we consider that they are all uniquely identified, which means that a message designed for a specific instance of a sub-protocol cannot be (re)used in another instance of the same sub-protocol.

The above restriction is a part of the *communication model*, which also contains the following assumptions. First, we consider that the communication channels provide integrity, and are ideally authenticated: when Alice sends a message to Bob, this message cannot be modified, it cannot be diverted as a message sent to Charlie and Bob will know that Alice was the one who sent it. In addition, a message cannot be replayed: after a message was delivered to a participant, it cannot be delivered again (except if the same message was sent several times). However, the channels are not considered secure: the adversary can read any message between two participants, block the messages and decide at which moment they might be delivered (if at all). In particular, the adversary can freely invert the order in which several messages may be delivered, which means that we consider an asynchronous model of communication. The asynchronicity of the model

also means that, at a given time, exactly one entity may be activated. We suppose that an entity's activation is interrupted if and only if the entity sends a message, writes something in its output tape or uses the specific instruction wait (in which case the adversary is activated next). However, any participant can send several messages simultaneously, in which case they may be delivered independently.

Another way to present the model of communication, as done in [CCL15], is to consider an incorruptible entity, known as the *router*. When a participant i sends a message m to another participant j, this is modeled as i writing a send query to the router, using a dedicated communication tape. The router then sends (i, j, m) to the adversary which is activated next. Then, at any moment, the adversary can decide to deliver a message that is stored inside the router, in which case the router deletes this message and writes (i, j, m) in j's communication tape. Therefore, we can consider that each (honest) participant has only two communication tapes: one for sending and the other for receiving, and that both tapes are connected to the router. Although the router is always honest, it cannot do anything else than forwarding the messages to the adversary and delivering them when instructed.

In this thesis, we consider an adversary which can non-adaptively corrupt some participants. The corrupted participants may be fully impersonated, which means that the adversary's program is executed instead. When representing the participants as ITM, one can consider that the communication tapes of the corrupted participants are connected to the adversary. In particular, since the router needs to identify which tape belongs to which participant in order to deliver the messages, we consider that the adversary's communication tapes are clearly identified, which means that the corrupted participants can be deduced from the tapes of the adversary. Note that it might be desirable to let the adversary choose the participants to corrupt at the very beginning of the experiment. In this case, we can still assume that the adversary has a dedicated tape for the corrupted participants and that those tapes allow it to "control" them.

In addition to the adversary $\mathbb{A}$, there is an additional polynomial ITM $\mathcal{Z}$ which represents the *environment*. It can arbitrarily write in the input tape of the participants (which activates them) and is activated when they write in their output tape (at which point it can directly read the output). Recall that we consider that several instances of the same sub-protocol are independent. Hence, if the environment writes several times in the input tape of the same participant, the latter will start a fresh, independent session of the protocol. For simplicity, we suppose that, in a given session, a honest participant may only write a single output, after which the session is killed and every local variable is erased. Apart from that, $\mathcal{Z}$ can only interact with $\mathbb{A}$, for instance by giving it some specific instructions and reading its feedback. (For simplicity, we consider that $\mathcal{Z}$ can only read the outputs of the honest participants since it can instruct $\mathbb{A}$ to directly send the corrupted participants' outputs.) Note that the interaction between $\mathcal{Z}$ and $\mathbb{A}$ is direct and does not use the router. Now, the security of a protocol is assessed by comparing the real protocol with an ideal one in which the computations are handled by some trusted party (which is not necessarily polynomially bounded).

In the ideal protocol, the honest participants handle their inputs to the trusted party which honestly follows a specific algorithm and is expected to give them an answer. However, while the adversary can block or delay the communications between a participant and a trusted party, it cannot read the corresponding messages: the content of the input and that of the trusted party's answer remain private. (Nevertheless, we suppose that the adversary can read a public part of the message, such as its length or the session identifier.) Note that the corrupted parties, who are under the control of the adversary, may send anything to the trusted party and not necessarily the input given by the environment. At some point, the trusted party can send a message to any participant. When such a message is delivered, the honest participant outputs

its content. Finally, the adversary may directly interact with the trusted party, without using a corrupted participant or the router as an intermediate. For instance, since the adversary is able to block all the communications, it is possible to consider that it can force the protocol to abort by instructing the trusted party to send $\perp$ to all the participants, which in turn will output $\perp$.

Intuitively, the protocol is SUC-secure if, for all adversary $\mathbb{A}$ in the real process, there exists a PPT simulator $\mathcal{S}$ in the ideal process such that no PPT $\mathcal{Z}$ can tell whether it is interacting with $\mathbb{A}$ in the real process or with $\mathcal{S}$ in the ideal process.

The idea is that $\mathcal{S}$ can interact with $\mathcal{Z}$ by simulating the real process, *i.e.* the honest participants, the router and the interaction with $\mathbb{A}$. For this purpose, $\mathcal{S}$ can act as the adversary of the ideal process in which $\mathcal{Z}$ is active. In this process, it makes some queries to the trusted party, impersonates the corrupted participants and delays the communications of the honest participants with the trusted party, as explained above. However, $\mathcal{S}$ cannot rewind $\mathcal{Z}$, and therefore cannot rewind $\mathbb{A}$ either since it may use $\mathcal{Z}$'s instructions to prevent this. With this restricted course of action, $\mathcal{S}$ must simulate the messages of the honest participants, but also make sure that its simulation is consistent with the outputs given by the trusted party (recall that the environment can read the outputs of the honest participants). For instance, in a protocol that aims at generating a common random string, the random string generated in the simulation must be the same as the one given by the trusted party in the ideal process.

Since the goal of the environment is to distinguish the simulation from the real process, we consider that it can only output 0 or 1. Also, to capture the fact that $\mathcal{Z}$ can be *any* environment, we consider that $\mathcal{Z}$ can have an arbitrary auxiliary input $z \in \{0,1\}^*$ of polynomial size, which represents some precomputed data. This input can depend on the protocol, on the adversary $\mathbb{A}$, and even on the simulator. Technically speaking, z can also depend on the security parameter; therefore, $\mathcal{Z}$ can be seen as a non-uniform adversary rather than a Turing machine.

Since $\mathcal{Z}$ has access to an unlimited precomputation power, SUC-security is a very strong notion of security. It means that whatever computation that $\mathbb{A}$ was able to perform (including something which would rig the distribution of the outputs), $\mathcal{S}$ is able to do the exact same thing in the ideal process, where there is no communications between the participants. Nonetheless, SUC-secure does not necessarily mean secure; rather, it means that the protocol is *as secure* as the ideal one, whose security is easy to assess.

3.3.2 The composition theorem

A composable framework is interesting because the security of a protocol can be deduced from the security of its sub-protocols, which gives a comprehensive way to build a security proof. For instance, suppose that we consider that a functionality $\mathcal{F}$ (*e.g.*, generating a common random group element) is taken for granted. Then we can imagine a protocol P in which the participants are able to make arbitrary queries to $\mathcal{F}$ in order to get an answer: this is the $\mathcal{F}$-hybrid model, depicted in Fig. 12. In [CCL15], the hybrid process is similar to the real process, except that the participants can make queries to some trusted parties listed in the family $\mathcal{F}$. By contrast with the ideal process, a query to a trusted party does not necessarily contain the participant's inputs; rather, it can be anything defined by the protocol's specifications. Just as in the ideal process, the content of the query and that of the trusted party's answer remain private, but the adversary can learn some information such as the length or the nature of the message (and use this to decide when to deliver the message, if at all). Finally, another difference is that the trusted party's answer to the participant is not output by the participant but instead processed depending on the specification of the protocol. (Note that the adversary can still directly interact with the ideal functionality, as in the ideal model.) In this thesis, we denote $\mathrm{Real}^{\mathcal{F}}_{P,\mathbb{A},\mathcal{Z}}(\lambda, z)$ (resp.

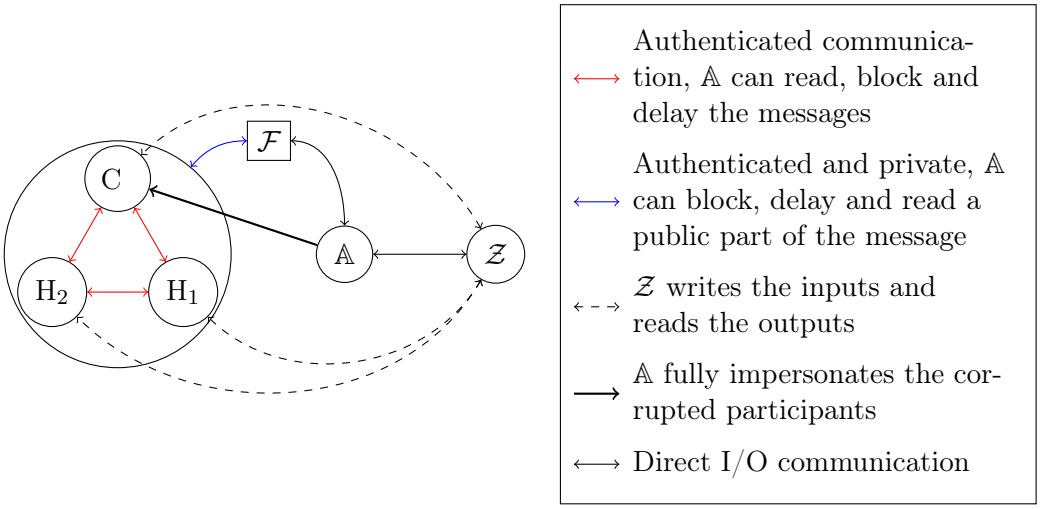


Figure 12: Illustration of the $\mathcal{F}$-hybrid process, with one corrupted participant C and two honest participants H_1 and H_2

$\text{Ideal}_{\mathcal{G},\mathcal{S},\mathcal{Z}}(\lambda, z)$) the probability that the environment outputs 1 while interacting with $\mathbb{A}$ (resp. $\mathcal{S}$) in the $\mathcal{F}$-hybrid process (resp. the ideal process were $\mathcal{G}$ is the trusted party). With these notations, SUC-security is given by Definition 18. Note that in the ideal model, the adversary has no access to $\mathcal{F}$; hence, the simulator must also simulate the interactions with $\mathcal{F}$ in addition to the messages of the participants and the outputs of $\mathcal{G}$.

Definition 18 ([CCL15])**.** *Let P be a protocol and $\mathcal{F}, \mathcal{G}$ two families of trusted parties. We say that P SUC-securely computes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model if, for all PPT $\mathbb{A}$, there exists a PPT $\mathcal{S}$ such that for all PPT $\mathcal{Z}$ and every $k \in \mathbb{N}$, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and $z \in \{0,1\}^{\lambda^k}$,*

$$\left|\text{Real}^{\mathcal{F}}_{P,\mathbb{A},\mathcal{Z}}(\lambda, z) - \text{Ideal}_{\mathcal{G},\mathcal{S},\mathcal{Z}}(\lambda, z)\right| \leq \mu(\lambda).$$

Now, an interesting result in the SUC framework is the universal composability of the SUC-secure property. To better understand this result, consider a protocol P in the $\mathcal{F}$-hybrid model. Also, consider that there is a protocol Q which SUC-securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model. Then we can consider the protocol P^Q in the $\mathcal{G}$-hybrid model, which is the protocol P except that every query to a trusted party in $\mathcal{F}$ is replaced by an invocation of Q as a sub-protocol, where the content of the query is the initial input of each participant. Since we considered that several copies of the same sub-protocol must be independent, it means that each invocation of Q will use its own independent copy of $\mathcal{G}$. The composition theorem [CCL15, Theorem 2.3], reproduced in Lemma 5, states that if Q and P are SUC-secure, then P^Q is SUC-secure as well.

Lemma 5. *Let P be a protocol in the $\mathcal{F}$-hybrid model and Q be a protocol that SUC-securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid process. Then, for all PPT $\mathbb{A}$, there exists a PPT $\mathcal{S}$ such that for all PPT $\mathcal{Z}$ and all $k \in \mathbb{N}$, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and $z \in \{0,1\}^{\lambda^k}$,*

$$\left|\text{Real}^{\mathcal{G}}_{P^Q,\mathcal{S},\mathcal{Z}}(\lambda, z) - \text{Real}^{\mathcal{F}}_{P,\mathbb{A},\mathcal{Z}}(\lambda, z)\right| \leq \mu(\lambda).$$

Thanks to this theorem, we can use the usual strategy of game hoping, and modify the protocol step by step so that there is less and less cryptography involved. However, it requires

to prove that some sub-protocols are SUC-secure to begin with, which can be scary if we look at the five consecutive quantifiers involved. For this reason, we also mention two fundamental results from the UC framework, which are also valid in the SUC framework (In fact, it is proven that a SUC-secure protocol is also UC-secure, see [CCL15, Theorem 4.13].)

Dummy adversary. The first result is that the first quantification on the PPT $\mathbb{A}$ can be dropped. More precisely, it is known that the hardest adversary to simulate is the *dummy* adversary, which forwards every message to the environment. In addition, the dummy adversary accepts queries of the form $\mathsf{send}(i, j, m)$ (resp. $\mathsf{deliver}(i, j, m)$) from the environment, where i is a corrupted (resp. any) participant, j is a participant and m a message. Upon receiving such a query, the adversary writes (i, j, m) in i's outward communication tape (resp. forwards the query to the router). (See for instance [Can00, Claim 11], which states that if the dummy adversary can be simulated, then any adversary can be simulated.)

Restricted environment. The above result means that we do not have to look for a process which, given an adversary, constructs a simulator; rather, we can directly exhibit an explicit, universal simulator. This, indeed, is reassuring and allows to better understand the notion of SUC-security. Nevertheless, the latter is still too generic as the environment is able to choose *any* input for the participants, including inputs which are not consistent with each other. For instance, consider a threshold decryption protocol. Then, if the participants do not agree with the public encryption key, the messages exchanged during the real protocol will most likely allow the environment to distinguish this situation with that of an ideal protocol. For this reason, it is important to restrict the inputs that the environment can give to the participants. In this thesis, we consider *restricted* environments, which are only allowed to choose the inputs in some (not necessarily efficiently) decidable language $\mathscr{L} \subset \{0,1\}^*$. In our example of a threshold decryption, the environment must give the same $\mathsf{pk}, (h_i)_{i=1}^{n_T}$ to every participant, and the secret shares of the participants must be consistent with the public key. In [Nie03, Theorem 3.5], it is shown that the composition theorem still holds if we restrict the environment to choosing such inputs.

3.3.3 Programmable random oracle model

We already mentioned that we use the ROM, which can be incorporated into the SUC framework as the ideal functionality $\mathcal{F}_{RO}$ defined in Algorithm 26 (we use the notation $x \in H$ to denote that x is a key of H). We acknowledge that using the ROM in a UC framework is not that common. Usually, using the common reference string assumption is preferred since the random oracle cannot be instantiated. We mention that there exists universally composable commitment schemes (see for instance [Lin11, FLM11]) that can be used instead of hash-based commitments. Outside of the ROM, a hash-based commitment is not equivocable nor extractable, and hence not SUC-secure. However, computing a universally composable commitment is by several orders of magnitude less efficient than computing a hash. Consequently, in electronic voting, it is unlikely that UC commitments will be used instead of hash functions in the near future. The same remark applies for ZKP (see for instance [GOS06] for a construction of universally composable ZKP).

3.3.4 An illustrative example: synchronous broadcast

Now that we introduced the SUC framework, the natural next step is to explain how we can prove that a protocol is SUC-secure. For this purpose, we give our first proof of SUC-security. On this occasion, we identify two sub-protocols that are featured in most asynchronous protocols, which are the broadcast and the synchronous broadcast. In order to simplify the presentation of the protocols in the remaining of this thesis, we will therefore define the corresponding ideal

Algorithm 26: $\mathcal{F}_{RO}$

Requires: The security parameter λ
Variables: A hashmap H (initially empty)
On *message* m *from participant* i**:**
  if $m \in H$ **then** Answer to i with $m, H(m)$;
  else
    $s \xleftarrow{\$} \{0,1\}^{2\lambda}$;
    Add the key m in H with the value s;
    Answer to i with m, s;

functionalities $\mathcal{F}_B$ and $\mathcal{F}_{SB}$, and use the $\mathcal{F}_B, \mathcal{F}_{SB}$-hybrid model.

Synchronous broadcast. In the SUC framework, the model of communication is atomic, which means that there is no native way for secure broadcasting. Typically, if a participant is supposed to broadcast a commitment, it can actually send a different commitment to the every other participant, which is certainly undesirable. In general, broadcasting in an unsecure communication network is a fundamental problem related to the byzantine agreement problem, which requires a majority of honest participants. However, it is common to consider that there is an ideal broadcast channel, which allows to abstract away this difficulty as an independent problem. Therefore, we use the $\mathcal{F}_B$-hybrid model, where $\mathcal{F}_B$ is an ideal functionality for broadcasting (see Algorithm 29). For simplicity, we consider that the messages sent by the ideal broadcast functionality arrive "simultaneously", which is technically a breach in the SUC framework since the adversary is supposed to be able to block and delay every message. However, we consider that properly modeling the broadcast ideal functionality in the SUC framework is out of scope for this thesis.

Broadcasting can be used for various reasons, but is sometimes insufficient. Indeed, it is often desirable that the values are not only broadcast (*i.e.* the same view is given to every participant), but also that they are broadcast *simultaneously*. In other words, in a situation where several participants must independently broadcast a value, we want to make sure that the adversary cannot cheat and choose the broadcast value depending on the values broadcast by the others. To prevent this, we use a round of synchronization which consists of the following steps. First, broadcast a commitment on the value to broadcast; second, once every commitment has been received, broadcast an opening of the commitment; third, verify the opening of all the other participants. (For a concrete example, see for instance Fig. 7, where this strategy is used to choose a common random implicit polynomial.) In order to simplify the description of our different protocols in the SUC framework, we introduce in Algorithm 28 an ideal functionality $\mathcal{F}_{SB}$ (for synchronous broadcast) which allows to factor this sub-protocol into a single query to a trusted party.

Note that $\mathcal{F}_{SB}$ is "consumed" when called once, which means that every instance of a synchronous broadcast protocol should call an independent copy of $\mathcal{F}_{SB}$. In practice, it means that at each activation, a honest participant creates a new, fresh session of the protocol with a different identifier, so that all the sessions are independent (this supposes that the session identifiers scheduling is publicly shared). To justify the use of the $\mathcal{F}_{SB}$-hybrid model, we also give in Algorithm 27 the protocol SB which allows to SUC-realize $\mathcal{F}_{SB}$ in the $\mathcal{F}_{RO}, \mathcal{F}_B$-hybrid model, as claimed in Lemma 6.

Syntax of a protocol in the SUC framework. In the SUC framework, a participant

is modeled as an ITM, and can be activated by two means: on receiving a message and when the environment writes a new input on its input tape. Depending on how a participant was activated and at which state it is in the procedure, it may have a different behavior, depending on the protocol. To model this, we use a specific syntax which specifies the exact behavior of the participant, see Algorithm 27 for an example. Note that the different actions in this algorithm are not necessarily sequential, as they depend on the state the participant is in and on the nature of the message / input that activated it. Also, recall that the execution is interrupted whenever a participant sends a message (that technically includes any query to an ideal functionality, but, for the sake of simplicity, we may write the procedure as if the following actions were continuous), writes in its output tape (in which case all the local variables are erased and the session is killed) or uses the specific command wait. Finally, remark that we give in Algorithm 27 the behavior for a single honest participant, which has the index i. This is because the same algorithm is used for all the honest participants, as it is always the case in this thesis.

A first proof in the SUC framework. We now give Lemma 6, which states that the SB protocol SUC-securely computes the corresponding ideal functionality.

Lemma 6. *The SB protocol described by Algorithm 27 SUC-securely computes $\mathcal{F}_{SB}$ in the $\mathcal{F}_{RO}, \mathcal{F}_B$-hybrid model.*

Proof. **The simulator.** We construct an explicit simulator $\mathcal{S}$ that simulates the entire hybrid process, including the router, the honest participants, the random oracle and $\mathcal{F}_B$; however, it does not simulate the corrupted participants who are controlled by the dummy adversary (the simulator can only control the corrupted participants of the ideal process). Unless stated otherwise, all of the following actions take place in the simulated real process.

First, the simulator gets the length of the input from the public part of the queries to $\mathcal{F}_{SB}$ in the ideal process. With this information, whenever a honest participant is activated in the initial state, the simulator simulates a communication from i to $\mathcal{F}_{RO}$ with the corresponding length, and change i's state to the commit state.

Then, whenever i is activated in the commit state, it checks that the activation is due to a message from the (simulated) $\mathcal{F}_{RO}$ and changes i's state to the opening state. To simulate i's broadcast, it generates a random a of 2λ bits and broadcasts it.

Then the simulator can run a perfect simulation up until a honest participant i has to reveal $\alpha||m_i$. This can only happen if this participant received a commitment c_k from all the other participants, including the corrupted ones. At this moment, since the simulator actually played the role of the random oracle, it can look for a preimage $\alpha_j||m_k$ for all c_k such that k is corrupted. If no preimage is found (*i.e.* no query was answered by c_k for a given k), the simulator chooses m_k at random. Now that the simulator has a value m_k for all the corrupted participant, it can use them to query the $\mathcal{F}_{SB}$ ideal functionality in the ideal process, using the corresponding value for each corrupted participant. However, it blocks $\mathcal{F}_{SB}$'s answers except for itself. This allows the simulator to learn the input of all the honest participants, and thus to broadcast $\alpha||m_i$ in the simulation as required (for this purpose, the simulator generates a random $\alpha \in \{0,1\}^\lambda$. Afterward, the corresponding honest participant switches to the verify state.

Now, the simulator can run a perfect simulation of the verify state, since it has access to all the information. However, since the simulator does not control the honest participants of the ideal process, it remains to explain how it can have them output the same values as in the simulated hybrid process. Clearly, if a honest participant i outputs $\mathsf{blame}(j)$ in the simulated hybrid process, it means that j was corrupted. Then the simulator can have the corrupted participant j send a message to $\mathcal{F}_{SB}$ in the ideal process, so that the ideal functionality will send $\mathsf{blame}(j)$ to all participants. At this point in the ideal process, the simulator blocks every such

Algorithm 27: SB

Requires: The security parameter λ
n: number of participants
Inputs: A message m_i

On *input* m_i**:**
  Start a new independent session
  in the Commit state;
  $\alpha \xleftarrow{\$} \{0,1\}^\lambda$;
  Query $\mathcal{F}_{RO}$ with $\alpha||m_i$;
State *Commit*:
  On *answer* q, a *from* $\mathcal{F}_{RO}$**:**
    Change state to Open;
    Query $\mathcal{F}_B$ with a;
State *Open*:
  On *answer* j, m *from* $\mathcal{F}_B$**:**
    if $c_j = \perp$ **then** $c_j \longleftarrow m$;
    if $c_k \neq \perp$ *for all* k **then**
      Change state to Verify;
      Query $\mathcal{F}_B$ with $\alpha||m_i$;
    else wait;
State *Verify*:
  On *answer* j, m *from* $\mathcal{F}_B$**:**
    if $m_j = \perp$ **and** $|m| \geq \lambda$ **then**
      Parse m as $\alpha_j||m_j$;
      Query $\mathcal{F}_{RO}$ with m;
    else wait;
  On *answer* q, a *from* $\mathcal{F}_{RO}$**:**
    for *all* k *s.t.* $\alpha_k||m_k = q$ **do**
      if $c_k = a$ **then** $v_k \longleftarrow 1$;
      else Output blame(k);
    if $v_j = 1$ *for all* j **then**
      Output $m_1||\cdots||m_n$;
    else wait;

Algorithm 28: $\mathcal{F}_{SB}$

Requires: n: number of participants
States: Initial state q_0
Final state q_f
(if several SB are required, the participants call several independent copies of $\mathcal{F}_{SB}$)

State q_0**:**
  On *message* m *from participant* i**:**
    if $s_i \neq \perp$ **then**
      Change state to q_f;
      Send blame(i) to all j;
    $s_i \longleftarrow m$;
    if $s_j \neq \perp$ *for all* j **then**
      Change state to q_f;
      Send $s_1||\cdots||s_n$ to all j and $\mathcal{S}$;
    else wait;
State q_f**:**
  On *message from* i**:**
    Send blame(i) to all j;

Algorithm 29: $\mathcal{F}_B$

On *message* m *from participant* i**:**
  Send i, m to all $j \neq i$;
  (no other message can be delivered between two of those messages, but the order in which they are delivered is still up to the adversary)

message except for i, which will cause the latter to output $\mathsf{blame}(j)$ as in the simulation. Finally, if a honest participant i outputs $m'_1||\cdots||m'_n$ in the simulation, the simulator uses the router of the ideal process to deliver to i the initial message $m_1||\cdots||m_k$ of $\mathcal{F}_{SB}$, which causes i to output $m_1||\cdots||m_k$.

Indistinguishability. Once the simulator is defined, it remains to explain that the simulation is computationally indistinguishable from the real process. For this purpose, we identify the only two elements which might differ in the simulation. First, when a honest participant broadcast a commitment, it broadcasts $\mathcal{O}_{\mathsf{RO}}(\alpha||m_i)$ for some random λ-bits α; however, in the simulation, the simulators does not know m_i yet and therefore broadcasts a random 2λ-bit elements. Clearly, except if two participants choose the same α (which happens with a negligible probability), the simulated commitments are perfectly indistinguishable from the real ones.

Second, in the ideal process, all the honest participants that do not output a blame have the same output $m_1||\cdots||m_n$; however, in the hybrid process, each said participant may output a different $m'_1||\cdots||m'_n$. However, if a honest participant outputs $m'_1||\cdots||m'_n$, it means that they received a message of the form $j, \alpha'_j||m'_j$ such that $\mathcal{O}_{\mathsf{RO}}(\alpha'_j||m'_j) = c_k$. Yet, since c_k was broadcast using $\mathcal{F}_B$, all the honest participants agree on this value, therefore they must all output the same $m'_1||\cdots||m'_n$ (unless a collision occurred, which happens with a negligible probability). Finally, we also have that $m'_j = m_j$ for all j, except if the adversary managed to find a (first or second) preimage of c_k or a collision (see Section 2.1.3 for a rigorous analysis of the security of a hash-based commitment in the ROM). □

Part II

Secure Tally-Hiding

As seen in Section 1.1.1, an electronic voting protocol is usually divided into several phases, such as the set up, the registration, the voting phase and the tally phase. For the latter, there are two main strategies: homomorphic tally and mixnets. The first strategy relies on the homomorphic property of the encryption scheme to decrypt the "sum" of the ballots sent by the voters, without decrypting the ballots individually. It is convenient, efficient and arguably ideal in many cases; however, it is not suitable for just any counting function. In particular, some very popular counting functions such as single transferable vote (STV) cannot be readily tallied using this strategy. For a more generic counting function, the solution by default is to rely on a decryption mixnet, which reveals all the choices made by the voters, while still concealing the link between any given voter and any given choice. Once the choices are known in the clear, the desired counting function can be publicly computed. Nevertheless, the main problem with this solution is that it reveals too much information compared to the result of the counting function. For instance, in STV voting, a voter can choose any permutation of the candidates, and there can be several hundreds of candidates. Consequently, there are often more voting options available than there are voters; therefore a voter can "sign" their ballot by using a specific and unlikely permutation. This leads to the so-called *Italian attacks*, where a coercer asks a voter to first choose the instructed candidate and then to choose a specific permutation. This way, the coercer can efficiently coerce a large number of voters simultaneously, and detect which voter obeys and which one disobeys. To address such a situation, we explore the possibility of using a fully *tally-hiding* scheme, which only reveals the result of the election. This part presents the results of [CGY22a], which is the conference version of [CGY21].

Tally-hiding is possible thanks to multi-party computation (MPC) techniques, which allow to evaluate any function on the private inputs of the participants, without revealing anything else than the output. For this reason, we first introduce some generic MPC protocols that illustrate the usual solutions that exist in the literature. Unfortunately, we will see that they are not always applicable in the context of electronic voting. Afterwards, we present the main primitive that we choose, and explain the motivations behind this choice. From this protocol, it is possible to derive other protocols that securely compute several arithmetic primitives such as additions and comparisons; and eventually to design an entire protocol for secure tally-hiding. For the security proofs, we used some involved arguments in the SUC framework.

Chapter 4

Multi-party computation for electronic voting

In multi-party computation, we consider several participants that each possess a secret input $x_1, \cdots, x_n$ and want to collectively compute $f(x_1, \cdots, x_n)$ without revealing anything else about their individual inputs. In general, f can output a different value to each participant but, for simplicity, we do not consider this possibility. For instance, a voting system can be seen as an instance of an MPC protocol, where the input of a voter is the chosen voting option, f is the counting function, and some additional participants such as the talliers are here to help performing the computations. The usual solution in MPC is to give a representation of f as a boolean circuit that consists of binary gates, to interpret the inputs as the sources of the circuit and to read the outputs from its leaves. In this context, f is considered as a function from $(\{0,1\}^N)^n$ to $\{0,1\}^m$, for some fixed N and m: since we do not want to reveal *anything* on the secret inputs, we do not want to reveal their length either, so that they might be padded in order to always have the maximum possible length N. Once f is represented as a circuit, it remains to explain how to securely evaluate each gate. In general, a boolean circuit can be evaluated thanks to an arithmetic circuit in any field $\mathbb{Z}_q$. Indeed, the field elements 0 and 1 would represent the corresponding boolean values; the multiplication corresponds to the logical and; and turning x into $1 - x$ corresponds to the logical negation.

A wide variety of protocols can be used to evaluate an arithmetic circuit, and we present some of them in Section 4.1. However, they are not necessarily suitable for electronic voting. Indeed, we do not expect the voters to engage into a complex MPC protocol; rather, the ideal is when they can *vote and go*. Hence, the usual strategy in electronic voting is that the talliers first generate a public encryption key, the voters encrypt their vote and then the talliers compute the result from the encrypted ballots. When transposing this in the MPC setting, it means that the participants of the MPC protocol would be the talliers and that their inputs would be the secret shares. As for the encrypted ballots, they can be a parameter of the function f, or some additional common inputs. In the literature, the Paillier encryption scheme emerged as the solution for MPC on encrypted data. In particular, we introduce the ABB framework in Section 4.2, which is suitable for electronic voting. However, there are many reasons to prefer the ElGamal encryption scheme from that of Paillier. Consequently, we explored the possibility of an MPC protocol based on the ElGamal encryption scheme. In Section 4.3, we present the conditional gate protocol, which is the main MPC building block that we use in this thesis.

4.1 Three popular approaches for multi-party computation

First, we mention three popular approaches for generic MPC.

4.1.1 Garbled circuits

A classical MPC strategy is to use the so-called *garbled circuits*, introduced by Yao in [Yao86]. In a two-parties setting, we have a *garbler* Ginny and an *evaluator* Evan. As in the general setting, they all have a private input that they represent as two bitstrings and a function f that they want to evaluate on their inputs, and that they represent as a boolean circuit. The idea is that Ginny will *garble* the circuit f: for each gate i, she creates four random labels, say $G_i^0, G_i^1, E_i^0, E_i^1$, as well as two random labels O_i^0, O_i^1. Intuitively, the G_i's represent the two possibilities for the first input while the E_i's represent the two possibilities for the second. Thanks to the truth table of the gate, Ginny can create the mapping $g : \{G_i^0, G_i^1\} \times \{E_i^0, E_i^1\} \to \{O_i^0, O_i^1\}$, that she uses to encrypt the gate. For this purpose, she uses a key derivation mechanism as well as a symmetric encryption function to compute $\mathrm{Enc}_{\mathrm{KDF}(G_i^x, E_i^y)}(g(G_i^x, E_i^y))$ for all $x, y \in \{0, 1\}$, that she communicates with Evan.

Now, so that Evan can evaluate the circuit, Ginny sends him all the labels that correspond to the bitwise representation of her inputs, in the correct order. In addition, she also sends him the labels that correspond to the bitwise representation of Evan' inputs, in the correct order. For this purpose, Ginny and Evan use an oblivious transfer protocol, which allows Evan to get the desired label without revealing the value of any bit of his input (see for instance [LP11] for a construction). Given the labels of the inputs of a gate, Evan derives the corresponding key and uses it to decrypt the output of the circuit. Note that Evan has four values to decrypt, while only one of them is the correct one. Therefore, it is necessary to impose that the decryption fails if an incorrect key is used. Once Evan has evaluated the circuit, he has the labels which correspond to the outputs of the circuit. Depending on whether we want to guarantee that Ginny or Evan gets the result, we can either ask Evan to send Ginny the label (in which case she learns the result and can share it with Evan if she wants), or ask Ginny to use plaintexts instead of random labels for the leaves of the circuit (in which case Evan learns the result and can share it with Ginny if he wants). In any case, it is difficult to guarantee that they both simultaneously get the result, since one can always decide to leave the protocol once they have learned the result.

Garbled circuit are still popular nowadays and benefited from many improvements over the years (*e.g.*, [ZRE15, BMR16]). However, it is not clear whether they can be applied in the context of electronic voting. A major difficulty is related to universal verifiability. Indeed, to prevent Ginny from encrypting an incorrect circuit that has the same number of gates, the participants use a *cut-and-choose* strategy: instead of garbling a single circuit, Ginny produces, say, k garbled circuits and has to open $k-1$ of them, chosen randomly by Evan. This way, if Ginny tries to cheat, this is detected with probability at least $(k-1)/k$. Nevertheless, it means that the garbler still has a non-negligible probability to cheat without being caught, which leads to an uncomfortable situation in electronic voting. In addition to not providing computational soundness, this paradigm also assumes that Evan is honest. If this is not the case, then Ginny and Evan can agree on one random circuit to rig: Evan opens the others, which are indeed valid, but evaluates the fake one. Unfortunately, it is difficult to fix this using the usual Fiat-Shamir transformation: if the circuits to open were determined from a hash of the encrypted circuits, Ginny could generate $k-1$ valid circuits and rig one at random, compute the ones that she has to open and start over again until she is successful. Therefore, using garbled circuits imposes an additional trust assumption on the participants, which is not ideal.

4.1.2 Linear secret sharing schemes

Another popular strategy is based on linear secret sharing schemes, as introduced in [BGW88, CCD88]. For simplicity, we consider Shamir's secret sharing scheme, where n participants share a secret x with a threshold t, using a random (implicit) polynomial P of degree t such that $P(0) = x$. In this setting, recall that the share of participant i is $x_i = P(i)$. Now, remark that if two secrets x and y have been shared with the (implicit) polynomials P and Q, then the participants can compute the sum $x_i + y_i = (P+Q)(i)$, which is a share of $x+y = (P+Q)(0)$. In other words, it is possible to add – without interaction – two shared secrets; hence the adjective *linear*. Now, by computing $x_i y_i$, the participants can similarly create the shares of the value xy; however, the polynomial PQ has a degree $2t$ and it would be preferable to share xy with a polynomial of degree t instead. Indeed, if the participants want to evaluate a complex arithmetic circuit on their shared secrets, they cannot afford to let the degree of the polynomial increase after each multiplication gate; otherwise, it would not be possible to recover the final output.

To circumvent this difficulty, assume that $2t < n$, so that it is still possible for the participants to recover the value xy from their shares $x_i y_i$. In what follows, we denote $[\![x]\!]_t^i$ the share of a secret x for the participant i, using a degree t polynomial. So that they can each obtain $[\![xy]\!]_t^i$ from $[\![x]\!]_t^i$ and $[\![y]\!]_t^i$, the participants can proceed as follows:

- Collectively generate $[\![m]\!]_t^i$ and $[\![m]\!]_{2t}^i$, for some random and unknown mask $m \in \mathbb{Z}_q$. Locally compute the shares $[\![z]\!]_{2t}^i$ of $z = xy - m$, as $[\![z]\!]_{2t}^i = [\![x]\!]_t^i [\![y]\!]_t^i - [\![m]\!]_{2t}^i$.
- Each participant broadcast their share $[\![z]\!]_{2t}^i$ of z, allowing them to deduce the value of z using Lagrange interpolation. Indeed, $[\![z]\!]_{2t}^i$ is the value obtained when evaluating in i a polynomial of degree $2t$. Therefore, by collecting $2t+1$ or more of them, one can deduce z, the value of the polynomial when evaluated in 0.
- Locally deduce $[\![xy]\!]_t^i$ as $z + [\![m]\!]_t^i$. Indeed, z can be considered as a trivial share of itself, using a constant polynomial. Therefore, since the secret sharing scheme is linear, $z + [\![m]\!]_t^i$ is a share of $z + m = xy$, as desired.

With the above protocol, it remains to explain how to actually generate $[\![m]\!]_t$ and $[\![m]\!]_{2t}$. A possible solution would be to use Pedersen's verifiable secret sharing scheme [Ped91b]: each participant i can choose two polynomials P and Q of degree t and $2t$, broadcast the corresponding commitments $(c_{i,k})_{k=0}^{t}$ and $(c'_{i,k})_{k=0}^{2t}$ with $c_{i,0} = c'_{i,0}$ (this condition means that both shared secrets are the same) and secretly send their shares $P(j)$ and $Q(j)$ to the other participants. Just as in Pedersen's DKG depicted in Fig. 7, the latter can check that their shares are consistent with the commitments, therefore the overall protocol would be secure, provided that we have a way to make sure that the commitments are actually *broadcast*, *i.e.* that the view of the commitments is the same for all the participants. After that each participant has broadcast $[\![m_i]\!]_t$ and $[\![m_i]\!]_{2t}$, everyone can locally compute the sum of all their shares. More involved solutions allow to decrease the overall complexity, especially the communication cost; see for instance [GSZ20].

In the context of electronic voting, using linear secret sharing schemes is more suitable than garbled circuits. First, the trust assumption where we require a honest majority is closer to what we usually assume for the talliers: although it is preferable to consider *any* threshold so that we can decrease their number, asking for $t < n/2$ can be considered acceptable. Second, it is no longer possible for the adversary to cheat without being caught with an overwhelming probability, therefore the verifiability of the corresponding protocols is closer to the usual notion of verifiability in electronic voting. In addition, we can think of various ways to augment those protocols with public commitments and/or ZKP to obtain universal verifiability, which means

that the trust assumption on the talliers would only be required for privacy, as this is usually the case in electronic voting. Finally, we already use secret sharing and a DKG protocol, therefore the MPC techniques are not *that far* from what we are used to.

However, in electronic voting, the talliers have the shares of the secret decryption key, and not of the voting options chosen by the voters. Hence, to actually use MPC based on secret sharing, we have but two solutions: either have the voters privately send a share of the chosen voting option to each tallier, or have the talliers evaluate a rather complex circuit which depends on the public ballots and takes the secret shares as inputs, instead of the chosen voting options. Since each multiplication gate requires to interactively precompute some $[\![m]\!]_t, [\![m]\!]_{2t}$ and to perform an additional interactive protocol once the ballots are known, the second solution may be too expensive. As for the first one, it would be preferable to let the voters produce a single encrypted ballot as usual, rather than asking them to compute a share for each tallier. Indeed, although computing a share seems like an easy task compared to an encryption and, say, a ZKP, the voters have to send their shares *privately*, which means that an additional (asymmetric) encryption must be computed for each tallier. In addition, it is not clear how the voters can *prove* that the shares they send are actually consistent with their ballot. A related question was addressed in Kryvos [HKK+22], where the voters send a commitment to the board and the share of their voting option to the talliers. However, the proposed solution requires the voters to compute an expensive SNARK (see Section 2.3.5), which takes up to several minutes according to their benchmark.

4.1.3 Fully homomorphic encryption

The linear secret sharing schemes offer an efficient and simple solution for generic MPC; however, it seems that they are not the perfect tool that we need in electronic voting. Ideally, we would like the talliers, who hold the shares of the secret key, to compute the result of the election from the encrypted ballots sent by the voters. Therefore, what we want is actually to perform some operations on encrypted, unknown data, rather than on known private inputs. At this point, a prominent question is "why don't you use fully homomorphic encryption?". Indeed, performing operations on encrypted data is a typical goal in cloud computing, where the proposed solution is to use a fully homomorphic encryption scheme (FHE). FHE recently emerged as a solution for generic MPC [KLO+19], and benefited from many improvements since the first practical proposals [Gen09, vDGHV10] (see for instance [CGGI16b, CLOT21]). Today, it is efficient enough to be used in practice. In addition, non-interactive ZKP for FHE schemes can be derived from lattice-based ZKP, which was the subject of active research recently [BCK+14, DFMS19, ESLL19, LNP22]. Finally, a distributed protocol to collectively evaluate an arithmetic circuit when the secret key has been shared between several participants can be found in [BGG+18]. Despite this, it seems that there is no well-established distributed key generation protocol for FHE schemes: the most relevant contributions that we could find are [KLO+19, AMM22]. Currently, other strategies are used to distribute the trust between the talliers, such as multi-key encryption [CCS19]; see for instance [dPLNS17] for an academic proposition of a post-quantum electronic voting system, and [CGGI16a] for another proposal based on FHE. With these materials, asking whether FHE may be used is a legitimate question that requires a thorough analysis. Designing a voting system is a tricky task that includes many pitfalls: the confidence that we have in the current solutions such as Helios is only possible thanks to years of studies.

4.2 The arithmetic blackbox for Paillier encrypted integers

A suitable framework for electronic voting is the ABB framework [Nie03, Part III], which allows to securely evaluate any function on encrypted data, provided that the secret key has been shared between several participants. This framework has been notably used in the independent work of Ordinos [KLM+20], that gives a contribution which is similar to our tally-hiding toolbox. To explain how Ordinos compares to our contribution, we provide several comparisons at different levels. First, we compare the primitives in Section 4.3.3; then we give a comparison at the protocol level in Section 5.3.1; finally, we compare the resulting voting systems in Chapter 6. In this section, we give all the necessary materials to fully understand the mechanism of Ordinos.

4.2.1 MPC from threshold homomorphic encryption

The main primitive of the ABB framework is a multiplication protocol which relies on the homomorphic property of the Paillier encryption scheme, introduced in Section 2.2.4.

First, recall that in the Paillier cryptosystem, the public encryption key is a stong RSA modulus n, such that $n = pq$ with two safe prime numbers p and q. Such a modulus is necessary coprime with its Euler totient $\phi(n)$, and the secret key is defined as an integer which is congruent to 0 modulo $\phi(n)$ and to 1 modulo n. To encrypt a message $m \in \mathbb{Z}_n$, one picks a random $r \in \mathbb{Z}_{n^2}^\times$ and computes $C = (1+n)^m r^n \in \mathbb{Z}_{n^2}^\times$. For $m \in \mathbb{Z}_n$, we denote $E_n = (1+n)^m$, the trivial encryption of m with the randomness 1. To decrypt C with the secret key sk, one computes $C^{\mathsf{sk}} - 1$ modulo n^2, interpret this as an integer $u \in [0, n^2 - 1]$ and returns $m = u/n$. Also, we recall that the Paillier encryption scheme is additively homomorphic: given two $m_1, m_2 \in \mathbb{Z}_n$ and $r_1, r_2 \in \mathbb{Z}_{n^2}^\times$, we have $\mathrm{Enc}_{\mathsf{pk}}(m_1, r_1)\mathrm{Enc}_{\mathsf{pk}}(m_2, r_2) = \mathrm{Enc}_{\mathsf{pk}}(m_1 + m_2, r_1 r_2)$. Finally, recall that there are DKG protocols and threshold decryption protocols available for the Paillier encryption scheme.

With all the above remarks, the only remaining ingredient that we need for secure computations on encrypted data is to be able to *multiply* two Paillier-encrypted messages. For this purpose, one can use the protocol of [DN03], which is an adaptation of the protocol from [CDN01]. We reproduce this protocol in Algorithm 30.

Algorithm 30: Mul

Requires: A threshold decryption setup for the Paillier cryptosystem, with the public key n
$A, B \in \mathbb{Z}_{n^2}^\times$, two encryptions of (unknown) $a, b \in \mathbb{Z}_n$

Outputs: C, a random encryption of ab

1 Each party i chooses a random $d_i \in \mathbb{Z}_n$, computes $M_i = B^{d_i}$, a random encryption D_i of d_i and a PoK π_i that M_i and D_i are well-formed. Finally, i broadcasts M_i, D_i, π_i;

2 Let S be the subset of the parties that gave a valid PoK. Each party compute $A \prod_{i \in S} D_i$. Then this value is decrypted into $x = a + \sum_{i \in S} d_i$ using threshold decryption;

3 The parties output $C = B^x / \prod_{i \in S} M_i$;

This protocol use the same *mask-reveal* paradigm as in Section 4.1.2: we first *mask* the value a by adding a random (unknown) value d and reveal $x = a + d$. Then, using the homomorphic property of the Paillier encryption scheme, we compute B^x, which is an encryption of $b(a+d)$. Afterwards, the value bd needs to be removed. For this purpose, we need an encryption M of bd, so that we can compute $C = B^x / M$. The resulting protocol is arguably simple: only two broadcasts are required per participant, and the size of the exchanged messages remain

reasonable. The security of this protocol has been assessed in a UC framework, see [Nie03, Part III].

4.2.2 Known MPC protocols in the ABB framework

It is remarkable that Algorithm 30 realizes the same functionality (the multiplication) as the protocol presented in Section 4.1.2; the main difference is that it operates on encrypted data instead of data that are shared between the participants. This similarity, in conjunction with the additively homomorphic property of the Paillier encryption scheme, is interesting because it means that most of the known protocols from the MPC community can be used in the ABB framework. For instance, although it is not explicitly stated, Ordinos makes an intensive use of the following classical MPC protocols:

- **RandBit** is a protocol which allows to collectively generate an encryption B of a random bit $b \in \{0, 1\}$. See Algorithm 32 for a possible realization.
- **RandBits(ℓ)** is a protocol which allows to collectively generate an encryption R of a random (quasi-uniform) $r \in [0, n-1]$, along with the encryptions $R_0, \cdots, R_{\ell-1}$ of its ℓ least significant bits. See Algorithm 31, which is adapted from [DFK+06]. In this algorithm, RangeProof allows to produce a zero knowledge proof that some element $r_{*,i}$ is in a specific range. To produce such a proof, a common reference string of the form N, g, h is necessary, where N is a strong RSA modulus and g, h two independent generators of the subgroup of the invertible squares modulo N. See Section 4.2.3 for more details.
- **RandInv** is a protocol which allows to collectively generate two encryptions R, R' of two $r, r' \in \mathbb{Z}_n$ such that $rr' = 1$. See Algorithm 34, which is adapted from [BB89].
- **Prefix** is a protocol which takes as input some encryptions $M_1, \cdots, M_k$ of (invertible and unknown) plaintexts $(m_1, \cdots, m_k)$ and outputs some encryptions $Z_1, \cdots, Z_k$ such that, for all i, Z_i is an encryption of the product $m_1 \cdots m_i$. See Algorithm 34, which is adapted from [BB89].

The main interest of those protocols is that they are highly parallelizable and can be done using a constant number of rounds. Indeed, they mostly consists of the generation of many independent random numbers. In addition, those generations can be *precomputed*, which is very interesting in electronic voting since this allows to allocate more time to compute the tally.

4.2.3 Range proofs for Paillier-encrypted integers

A non-interactive zero-knowledge **range proof** allows a prover to prove that a value is in a given range. To prove that $v \in [a, b]$, the main strategy is to prove that $(v - a)(b - v) \geq 0$, therefore a range proof can be obtained from a proof that a value is non-negative. In the context of a Paillier encrypted values, a classical method to prove that $x \geq 0$ (see for instance [Lip03]) consists of making an integer commitment c of the plaintext x, to prove that c is consistent with the Paillier ciphertext X, then to prove that x is the sum of four squares. This proof method has been slightly optimized in [Gro05, Section 5], where it is remarked that it is sufficient to prove that $4x + 1$ is the sum of three squares. To compute such squares, one can first choose a random even integer $x_1 \in \mathbb{N}$ of a carefully calibrated size, so that $p = 4x + 1 - x_1^2$ is prime with some non-negligible probability (otherwise, pick another x_1). Since p is congruent to 1 modulo 4, it can be written as the sum of two squares.

Algorithm 31: RandomBits

Requires: A CRS $\sigma = (N, g, h)$
n, a Paillier encryption key
ℓ, the number of bits
Outputs: $R, (R_0, \cdots, R_{\ell-1})$ s.t.
R is an encryption of
$r \in [0, n-1]$ and
$(R_0, \cdots, R_{\ell-1})$ are
encryptions of the ℓ least
significant bits of r

1 **for** $i = 0$ to $\ell - 1$ **do** $R_i \leftarrow$ RandBit();
2 **for** $i = 1$ *to* n_T*, participant* i **do**
3 $\quad B \longleftarrow 2^{\ell+\lambda-1} - 1$;
4 $\quad r_{*,i} \xleftarrow{\$} [0, B]$;
5 $\quad R_{*,i}, \pi_i \leftarrow \text{RangeProof}(\sigma, n, B, r_{*,i})$;
6 $R \longleftarrow \prod_{i=1}^{n_T} R_{*,i}$;
7 **for** $i = m - 1$ *down to* 0 **do**
8 $\quad R \longleftarrow R^2$;
9 $\quad R = RR_i$;
10 **return** R, $(R_0, \cdots, R_{m-1})$;

Algorithm 32: RandBit

Requires: pk, an exponential ElGamal or Paillier encryption key
E_1, an encryption of 1
Outputs: Z, an encryption of a random $b \in \{0, 1\}$

1 $Z_0 \longleftarrow E_1$;
2 **for** $i = 1$ *to* n_T*, participant* i **do**
3 $\quad s_i \xleftarrow{\$} \{-1, 1\}$; $r \xleftarrow{\$} \mathscr{R}$;
4 $\quad Z_i \longleftarrow Z_{i-1}^{s_i} \text{Enc}_{\mathsf{pk}}(0, r)$;
5 $\quad$ Produce a PoK π_i of well-formedness;
6 $\quad$ Send (Z_i, π_i) to the other participants;
7 The participants verify the PoK of the others;
8 **return** $(E_1 Z_{n_T})^{\frac{1}{2}}$;

Algorithm 33: Prefix

Requires: $M_1, \cdots, M_N$ encryptions of $m_1, \cdots, m_N \in \mathbb{Z}_n^\times$
Outputs: $Z_1, \cdots, Z_N$, encryptions of $m_1, m_1 m_2, \cdots, \prod_{i=1}^{N} m_i$

1 $R_0 \longleftarrow 1$;
2 **for** $i = 1$ *to* N **do**
3 $\quad R_i, R_i' \longleftarrow$RandInv();
4 $\quad S_i \longleftarrow \text{Mul}(R_{i-1}, M_i)$;
5 $\quad S_i \longleftarrow \text{Mul}(S_i, R_i')$;
6 $\quad$ Decrypt S_i into s_i.
7 $Z_1 \longleftarrow M_1$;
8 **for** $i = 2$ *to* N **do**
9 $\quad a_i \longleftarrow \prod_{j=1}^{i} s_j$;
10 $\quad Z_i \longleftarrow R_i^{a_i}$;
11 **return** $Z_1, \cdots, Z_N$;

Algorithm 34: Randinv

Outputs: R, R', encryptions of $r \in_r \mathbb{Z}_n^x$ and r^{-1}

1 The participants simultaneously broadcast some random $A_i, B_i \in \mathbb{Z}_{n^2}$;
2 $A \longleftarrow \prod_{i=1}^{n_T} A_i$; $B \longleftarrow \prod_{i=1}^{n_T} B_i$;
3 $C \longleftarrow \text{Mul}(A, B)$;
4 The participants decrypt C into c;
5 $R \longleftarrow A$; $R' = B^{c^{-1}}$;
6 **return** R, R';

Table 4: Estimated complexity of the range proof, assuming that $|n| = |N| = 3072$ and that $\lambda = 128$; the computational complexity is expressed as an approximative equivalent in the number of exponentiations modulo n^2, with a 3072 bits exponent

Prover (# exp.)	Verifier (# exp.)	Proof size
6	4	$\sim 20\|n\|$

For completeness, we give a possible construction for a range proof in the Paillier setting in Algorithm 35. In this algorithm, we use the integer commitment scheme of [DF02], as well as the corresponding ZKP that allows to prove that a given commitment c and Paillier ciphertext X opens to the same integer x. The algorithm can be seen as a combination of the two proofs, that we reproduced from [CPP17, Section 3.2] and [CPP17, Fig. 2]. See also [DLP22] for another proof system.

For the complexity analysis, our main metric is the number of exponentiations modulo n^2, with an exponent n. This corresponds to the cost of computing a Paillier encryption of 0. Since the exponentiations do not have the same size, we counted the number of modular multiplications instead, then converted this number to a number of exponentiation by dividing by $\log n$. However, a multiplication modulo N does not cost the same as modulo n^2. Since there are twice as many bits to compute, the latter costs approximately four times as much as the former. Hence, we count the number of modular multiplications modulo N, add to this four time the number of modular multiplications modulo n^2. Then, considering that n and N have approximately the same size, we divide by $4 \log n$ to *convert* this number to a number of exponentiations modulo n^2. To simplify, we considered that the number ℓ of bits to generate in the RandBits process (which is the only protocol where the range proof is needed) is small compared to the other parameters. The resulting complexity estimate is given in Table 4, assuming that N and n are 3072-bits integers and that the security parameter is $\lambda = 128$.

4.2.4 Comparing two Paillier encrypted integers

We can now present the main primitive of Ordinos, which is the equality test from [LT13]. To test whether two Paillier encrypted ℓ-bits integers x and y are equal, one can use the additively homomorphic property of the Paillier encryption scheme to obtain an encryption of $x - y$, and test whether $x - y = 0$. To test whether an encrypted x is equal to 0, Lipmaa and Toft suggest to create the unique polynomial P_ℓ such that $P_\ell(1) = 1$ and $P_\ell(k) = 0$ for $k \in \{2, \cdots, \ell + 1\}$. Their strategy is to first compute the Hamming weight h of x, then to evaluate P_ℓ on $1 + h$, from which they derive the result. The corresponding protocol is given in Algorithm 37.

The advantage of this approach is that most of the procedure can be precomputed so that only a small part has to be done *online*, after the operands are known.

From this protocol, one can construct a recursive comparison protocol, as explained in [LT13]. To compute the greater-than comparison on two encrypted x and y, the idea is to test the equality of the most significant halves of x and y. If they are equal, we compare the least significant halves recursively; otherwise, we compare the most significant halves recursively. This gives Algorithm 38, which is also used in Ordinos. In this algorithm, we took the liberty to denote $R_\top, R_\bot$ the result of RandBits, while RandBits returns some encryptions of the form $R, (R_0, \cdots, R_{l-1})$. We can derive $R_\bot$ as $\prod_{i<\ell/2}(R_i)^{2^i}$ and $R_\top$ in a similar manner.

Algorithm 35: Range proof

Requires: A strong RSA modulus N
Two squares $(g, h) \in \mathbb{Z}_N^\times$
A Paillier encryption key n
A bound B, and $x \in [0, B]$
Outputs: X, a Paillier encryption of x
and a ZKP π that $x \in [0, B]$

$s \xleftarrow{\$} \mathbb{Z}_{n^2}^\times$; $X \longleftarrow \mathrm{Enc}_n(x, s)$;
$r \xleftarrow{\$} [0, N-1]$; $c \longleftarrow g^x h^r$;
(* PoK that c and X are consistent: *);
$\alpha \xleftarrow{\$} [0, 2^{2\lambda} n - 1]$; $\rho \xleftarrow{\$} \mathbb{Z}_{n^2}^\times$;
$\beta \xleftarrow{\$} [0, 2^{2\lambda} n - 1]$;
$c_{\mathrm{Enc}} \longleftarrow \mathrm{Enc}_n(\alpha, \rho)$;
$c_{\mathrm{Com}} \longleftarrow g^\alpha h^\beta$;
$d_{PoK} \longleftarrow \mathrm{hash}(n||N||g||h||c_{\mathrm{Enc}}||c_{\mathrm{Com}})$;
$a_x \longleftarrow \alpha + d_{PoK} x$;
$a_s \longleftarrow \rho s^d$;
$a_r \longleftarrow \beta + dr$;
$\pi_{PoK} \longleftarrow (c_{\mathrm{Enc}}, c_{\mathrm{Com}}, a_x, a_s, a_r)$;
(* Beginning of the range proof *);
Compute $x_1, x_2, x_3 \in \mathbb{Z}$ s.t.
$4(B-x)x + 1 = \sum_{i=1}^3 x_i^2$;
$r_1, r_2, r_3 \xleftarrow{\$} [0, N]$;
for $i = 1$ to 3 **do** $c_i \leftarrow g^{x_i} h^{r_i}$;
$x_0 \longleftarrow B - x$; $r_0 \longleftarrow r$;
$m_0, \cdots, m_3 \xleftarrow{\$} [0, 2^{2\lambda} B]$;
$s_0, \cdots, s_3 \xleftarrow{\$} [0, 2^{2\lambda} N]$;
$\sigma \xleftarrow{\$} [0, 2^{2\lambda} BN]$;
for $i = 1$ to 3 **do** $e_i \longleftarrow g^{m_i} h^{s_i}$;
$e_4 \longleftarrow h^\sigma c^{4m_0} \prod_{i=1}^3 c_i^{-m_i}$;
$d \longleftarrow$
$\mathrm{hash}(n||N||g||h||B||c||(c_i)_{i=1}^3||(e_i)_{i=1}^4)$;
for $i = 0$ *to* 3 **do**
  $z_i \longleftarrow dx_i + m_i$;
  $t_i \longleftarrow dr_i + s_i$;
$\tau \longleftarrow \sigma + d(x_0 r_0 - \sum_{i=1}^3 x_i r_i)$;
$\Pi \longleftarrow$
$(c, \pi_{PoK}, (c_i)_{i=1}^3, (e_i)_{i=1}^4, (z_i, t_i)_{i=0}^3, \tau)$;
return X, Π;

Algorithm 36: Verification algorithm

Requires: A strong RSA modulus N
Two squares $(g, h) \in \mathbb{Z}_N^\times$
A Paillier encryption key n
A bound B
A ciphertext $X \in \mathbb{Z}_{n^2}$
Inputs: $c_{\mathrm{Enc}}, a_s \in \mathbb{Z}_{n^2}^\times, c_{\mathrm{Com}} \in \mathbb{Z}_N^\times$
$a_x, a_r \in \mathbb{Z}$
$c, (c_i)_{i=1}^3, (e_i)_{i=1}^4 \in \mathbb{Z}_N^\times$
$(z_i, t_i)_{i=0}^3, \tau \in \mathbb{Z}$

(* Verification of the PoK*);
$d_{PoK} \longleftarrow \mathrm{hash}(n||N||g||h||c_{\mathrm{Enc}}||c_{\mathrm{Com}})$;
if $c_{\mathrm{Enc}} \neq \mathrm{Enc}_n(a_x, a_s) X^{-d_{PoK}}$ **or**
$c_{\mathrm{Com}} \neq g^{a_x} h^{a_r} c^{-d_{PoK}}$ **then return** 0;
(* Verification of the range proof*)
$d \longleftarrow$
$\mathrm{hash}(n||N||g||h||B||c||(c_i)_{i=1}^3||(e_i)_{i=1}^4)$;
for $i = 0$ *to* 3 **do**
  if $g^{z_i} h^{t_i} c_i^{-d} \neq e_i$ **then return** 0;
if $e_4 \neq h^\tau g^d c^{4z_0} \prod_{i=1}^3 c_i^{-z_i}$
then return 0;
return 1;

Algorithm 37: EQH

Requires: An encryption X of a ℓ-bit integer x
P_ℓ, the unique polynomial of degree ℓ such that $P_\ell(1) = 1$ and $P_\ell(k) = 0$ for $k \in \{2, \cdots, \ell + 1\}$

Outputs: Z, an encryption of 1 if $x = 0 \bmod 2^\ell$, of 0 otherwise

$R, R_{\ell-1}, \cdots, R_0 \longleftarrow \text{RandBits}(\ell)$;
$M, M' \longleftarrow \text{RandInv}()$;
$M_1, \cdots, M_\ell \longleftarrow \text{Prefix}(M, \cdots, M)$;
$A \longleftarrow X/R$;
The participants collectively decrypt A into a;
Let $a_0, \cdots, a_{\ell-1}$ be the bit representation of $a - n$ modulo 2^ℓ;
$H \longleftarrow E_1 \prod_{i=0}^{\ell-1} E_{a_i} R_i^{1-2a_i}$ (* $h = 1 + \sum_{i=0}^{\ell-1} a_i \oplus r_i$ *);
$M_H \longleftarrow \text{Mul}(M', H)$;
The participants decrypts M_H into m_H;
for $i = 0$ *to* ℓ **do**
  $H_i \longleftarrow M_i^{(m_H)^i}$
Return $Z = \prod_{i=0}^{\ell} H_i^{\alpha_i}$ (* where the α_i's are the coefficients of P_ℓ *);

Algorithm 38: GTH

Requires: X, Y, ℓ, two encryptions of two ℓ-bit integers x and y

Outputs: Z, an encryption of 1 if $x \geq y$, of 0 otherwise

if $\ell = 1$ **then**
  return $E_1/Y\text{Mul}(X, Y)$;
$R, R_\perp, R_\top \longleftarrow \text{RandBits}(\ell)$;
$W \longleftarrow E_{2^\ell} X/Y$;
$M \longleftarrow WR$;
Decrypt M into m;
$m_\perp \longleftarrow m \bmod 2^{\ell/2}$; $m_\top \longleftarrow \lfloor m/2^{\ell/2} \rfloor \bmod 2^{\ell/2}$;
$B \longleftarrow \text{EQH}(E_{m_\top}, R_\top)$ (* $x_\top = y_\top$ *);
$C \longleftarrow B^{m_\perp - m_\top} E_{m_\top}$ (* $m_\perp$ if $b = 1$, $m_\top$ otherwise *);
$D \longleftarrow \text{Mul}(B, R_\perp/R_\top) R_\top$ (* $r_\perp$ if $b = 1$, $r_\top$ otherwise *);
$F \longleftarrow E_1/\text{GTH}(C, D)$;
$W' \longleftarrow F^{2^\ell} E_{m \bmod 2^\ell}/(R_\top^{2^{\ell/2}} R_\perp)$ (* $w \bmod 2^\ell$ *);
Return $Z = (W/W')^{1/2^\ell}$;

4.3 The conditional gate protocol in the ElGamal setting

Until now, we presented existing MPC protocols, and gave some details about the protocols used in Ordinos. We now present the protocol that we used in this thesis, and explain the motivations behind this choice.

4.3.1 Presentation of the protocol

Suppose that the secret key sk of an exponential ElGamal public key has been shared between several participants. For instance, $\mathsf{pk} = (g, h)$ and, to encrypt $m \in \mathbb{Z}_q$, one picks $r \in \mathbb{Z}_q$ at random and computes $(g^r, g^m h^r)$. For $m \in \mathbb{Z}_q$, we denote $E_x = (1_G, g^m)$, the trivial exponential ElGamal encryption of m. In this setting, we can readily compute the logical negation of two encrypted bits: if X is an encryption of b, then E_1/X is an encryption of $1-b$. To compute the logical and, we use the *conditional gate* protocol from [ST04]. However, we adapted the original protocol into Algorithm 39, for the sake of SUC-security. In this algorithm, we denote $\mathrm{Renc}_{\mathsf{pk}}(X, r)$ the algorithm that takes a ciphertext X, a random $r \in \mathbb{Z}_q$, and returns $X\mathrm{Enc}_{\mathsf{pk}}(0, r)$. Compared to the original protocol, the adaptation is less efficient but can be proven secure in the SUC framework, as shown in Theorem 3.

Although the original protocol in [ST04] is called the *conditional gate* in [ST04], we denoted Algorithm 39 CSZ, which stands for "conditionally set to zero". Indeed, given an encryption X of any $x \in \mathbb{Z}_q$ and Y of $y \in \{0, 1\}$, $\mathrm{CSZ}(X, Y)$ is a random reencryption of X if $y = 1$ and a random encryption of 0 otherwise. Therefore, this algorithm is slightly more useful than a logical and gates which would require both inputs to be in a binary domain.

Algorithm 39: CSZ

Requires: G, a group of prime order q and public coin generator g
pk of the form (g, h), an exponential ElGamal public key,
whose shares are distributed among the n_T participants
$\tilde{h} \in G$, a public element independent from pk
X, an encryption of some $x \in \mathbb{Z}_q$
Y, an encryption of $y \in \{0, 1\}$

Outputs: Z, a random encryption of xy

1 $Y_0 \longleftarrow E_{-1}Y^2$; $X_0 \longleftarrow X$;
2 **for** $i = 1$ *to* n_T*, for the authority* i*,* **do**
3 $(u, v) \longleftarrow X_{i-1}$;
4 $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q$; $s \xleftarrow{\$} \{-1, 1\}$;
5 $X_i \longleftarrow (u^s g^{r_1}, v^s h^{r_1})$; $e \longleftarrow \tilde{h}^{r_1}$;
6 $Y_i \longleftarrow \mathrm{Renc}_{\mathsf{pk}}(Y_{i-1}^s, r_2)$;
7 Broadcast X_i, e, Y_i and a ZKP π_i that they are well formed (see Algorithm 41) ;
8 Each authority verifies the proof of the other authorities (see Algorithm 42) ;
9 They collectively rerandomize X_{n_T} and Y_{n_T} into X' and Y' (see Algorithm 40) ;
10 They collectively decrypt Y' into y_{n_T} (see Algorithm 18);
11 They output $Z = (XX'^{y_{n_T}})^{\frac{1}{2}}$;

Algorithm 39 consists of three interactive steps. To begin with, as Y is supposed to be an encryption of $y \in \{0, 1\}$, we use the homomorphic property to turn it into an encryption Y_0 of $2y - 1 \in \{-1, 1\}$ at line 1. This operation is essentially *free*, and does not require any

interaction. Then, the first real step is a round of communications (lines 1 to 8). During this step, the authorities collectively generate a random and implicit $s \in \{-1, 1\}$ and compute a reencryption X_{n_T} (resp. Y_{n_T}) of X^s (resp. Y_0^s). This way, Y_{n_T} is an encryption of a random $y' = s(2y-1) \in \{-1, 1\}$, so that decrypting it does not reveal anything about the initial value of y. The second step is a rerandomization phase (line 9) which we added to obtain SUC-security. Finally, the last step is a threshold decryption protocol (line 10), during which the authorities decrypt Y' to obtain y'. Then, by computing $X'^{y'}$, they can form an encryption of $x(2y-1)$ (indeed, X' is an encryption of sx so that the sign s is simplified in the exponent). To derive the desired encryption of xy, they can locally multiply by X (to obtain an encryption of $2xy$) then raise to the power of $(q+1)/2$, which cancels the factor 2 in the exponent. In what follows, we comment on the modifications that we made compared to the original version of [ST04].

Public coin g. For a technical reason, we require that g is obtained with a public coin protocol. For this purpose, we consider that g is derived from a hash of "Conditional gate". This is useful for an explicit reduction to DDH, since it prevents the environment of the SUC framework to choose g freely. Note that there exists other versions of the DDH game, where the adversary is allowed to choose the generator g. If such a computational assumption is made, we no longer need g to be public coin.

Round of communications. Compared to the original conditional gate protocol, we added the following modifications:

1. We use an ElGamal commitment $(u^s g^{r_1}, \tilde{h}^{r_1})$ instead of a Pedersen commitment $g^s \tilde{h}^r$;
2. In addition, we require that the ElGamal commitment and the reencryption use the same randomness r_1;
3. Finally, we also demand that the participants prove that $s \in \{-1, 1\}$, while it was originally only required to prove the knowledge of some $s \in \mathbb{Z}_q$.

The two first modifications were made to obtain the extractability of u^s *à la Shoup* (without rewinding). Combined with the third modification, they allow the simulator to extract the value $s_1 \in \{-1, 1\}$ used by the adversary, which is required in the proof of SUC-security. (Since u may be chosen by the adversary, this also requires to check that $u \neq 1$.) To prevent the adversary from exploiting the trapdoor $\log_{\tilde{h}}(g)$ (which allows to extract u^s) not the trapdoor $\log_{\tilde{h}}(h)$ (which allows to extract v^s), we can derive $\tilde{h}$ from a hash of pk. In addition to provide extractability, the third modification prevents the adversary from choosing $s_1 \notin \{-1, 1\}$. This means that, after the round of communications, Y_{n_T} is an encryption of $y' = s_1 s_2 (2y-1) \in \{-1, 1\}$, where $s_2 \in \{-1, 1\}$ is a random element determined by the choices of the honest participants. Hence, by computing $X_{n_T}^{y'}$, the sign $s_1 s_2$ is simplified in the exponent, as expected. To obtain the PoK required at line 7, one can use a standard disjunctive PoK (see Algorithm 41).

Rerandomization. We also added a second step, which is a reencryption phase (see line 9). This is necessary for the SUC framework; indeed, consider an attacker that corrupts the last participant. Then it can choose many random s, r_1, r_2 until X_{n_T} meets a particular pattern that occurs with non-negligible probability (for instance, the 7 first bits in its bitwise representation are 0). The consequence of such an "attack" is that Z will not be an *uniformly* random encryption of xy as desired, so that SUC-security would be lost. To perform the rerandomization, we can use a synchronous broadcast of random encryptions of 0, along with the corresponding ZKP. (On this occasion, we recall that synchronous broadcast is discussed in Section 3.3.4.) For the rerandomization phase, we consider the protocol described in Algorithm 40.

Algorithm 40: Rerandomization

Requires: G, a group of prime order q
pk, an ElGamal public key
Inputs: X, a ciphertext
Outputs: X', a rerandomization of X

1 **for** $i = 1$ *to* n_T, *participant* i **do**
2 $r_i \xleftarrow{\$} \mathbb{Z}_q$;
3 $A_i \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(0, r_i)$;
4 $\pi_i \longleftarrow \mathrm{PoK}^0(\mathsf{pk}, A_i, r_i)$;
5 (* Can be obtained Algorithm 15, with $(g_1, g_2) = \mathsf{pk}$ and $(g_3, g_4) = A_i$*)
6 $c_i \longleftarrow \mathrm{hash}(A_i, \pi_i)$;
7 Broadcast the commitment c_i;
8 Once a commitment has been received from all the other authorities, broadcast A_i, π_i;
9 Verify that the broadcast A_j, π_j are consistent with the corresponding commitments;
10 **return** $X \prod_{i=1}^{n_T} A_i$;

Computing the PoK. At line 7, we need a PoK that X_i, c_s, Y_i is well formed. This proof guarantees that there exists $s \in \{-1, 1\}$ and $r_1, r_2 \in \mathbb{Z}_q$ such that $X_i = \mathrm{Renc}_{\mathsf{pk}}(X_{i-1}, r_1)$, $e = \tilde{h}^{r_1}$ and $Y_i = \mathrm{Renc}_{\mathsf{pk}}(Y_{i-1}^s, r_2)$. We use the following standard disjunctive proof:

$$X_i = \mathrm{Renc}_{\mathsf{pk}}(X_{i-1}, r_1) \textbf{ and } Y_i = \mathrm{Renc}_{\mathsf{pk}}(Y_{i-1}, r_2) \textbf{ and } e = \tilde{h}^{r_1}$$
$$\textbf{or}$$
$$X_i = \mathrm{Renc}_{\mathsf{pk}}(X_{i-1}^{-1}, r_1) \textbf{ and } Y_i = \mathrm{Renc}_{\mathsf{pk}}(Y_{i-1}^{-1}, r_2) \textbf{ and } e = \tilde{h}^{r_1}.$$

Algorithm 41: PoK-CSZ

Requires: A group G of prime order q
An exponential ElGamal public key pk
Some ciphertexts $X_i, Y_i, X_{i-1}, Y_{i-1}$ and $e \in G$
$r_1, r_2 \in \mathbb{Z}_q$ and $s \in \{-1, 1\}$ such that
$X_i = \mathrm{Renc}_{\mathsf{pk}}(X_{i-1}^s, r_1)$, $Y_i = \mathrm{Renc}_{\mathsf{pk}}(Y_{i-1}^s, r_2)$ and $e = \tilde{h}^{r_1}$

1 $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$;
2 $c_{s,X} \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(0, \alpha)$; $c_{s,Y} \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(0, \beta)$; $c_{s,e} \longleftarrow \tilde{h}^{\alpha}$;
3 $d_{-s}, a_{-s,X}, a_{-s,Y} \xleftarrow{\$} \mathbb{Z}_q$;
4 $c_{-s,X} \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(0, a_{-s,X})(X_i X_{i-1}^s)^{-d_{-s}}$;
5 $c_{-s,Y} \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(0, a_{-s,Y})(Y_i Y_{i-1}^s)^{-d_{-s}}$;
6 $c_{-s,e} \longleftarrow \tilde{h}^{a_{-s,X}} e^{-d_{-s}}$;
7 $d \longleftarrow \mathrm{hash}(\mathsf{pk}||X_{i-1}||Y_{i-1}||X_i||Y_i||c_{1,X}||c_{1,Y}||c_{-1,X}||c_{-1,Y}||c_{1,e}||c_{-1,e})$;
8 $d_s \longleftarrow d - d_{-s}$;
9 $a_{s,X} \longleftarrow \alpha + r_1 d_s$; $a_{s,Y} \longleftarrow \beta + r_2 d_s$;
10 Return $(c_{1,X}, c_{1,Y}, c_{-1,X}, c_{-1,Y}, c_{1,e}, c_{-1,e}, d_1, d_{-1}, a_{1,X}, a_{1,Y}, a_{-1,X}, a_{-1,Y})$;

To verify the proof, one can use Algorithm 42.

Algorithm 42: Ver-CSZ

Requires: A group G of prime order q
An exponential ElGamal public key pk
Some ciphertexts $X_i, Y_i, X_{i-1}, Y_{i-1}$ and $e \in G$
$\pi = (c_{1,X}, c_{1,Y}, c_{-1,X}, c_{-1,Y}, c_{1,e}, c_{-1,e}, d_1, d_{-1}, a_{1,X}, a_{1,Y}, a_{-1,X}, a_{-1,Y})$

1 **if** X_i is of the form $(1_g, *)$ **then return** 0;
2 $d \longleftarrow \text{hash}(\mathsf{pk}||X_{i-1}||Y_{i-1}||X_i||Y_i||c_{1,X}||c_{1,Y}||c_{-1,X}||c_{-1,Y}||c_{1,e}||c_{-1,e})$;
3 Check that the following equalities hold:
4 $\quad d_1 + d_{-1} \stackrel{?}{=} d$;
5 $\quad \text{Enc}(0, a_{1,X})(X_i/X_{i-1})^{-d_1} \stackrel{?}{=} c_{1,X}$;
6 $\quad \text{Enc}(0, a_{1,Y})(Y_i/Y_{i-1})^{-d_1} \stackrel{?}{=} c_{1,Y}$;
7 $\quad \tilde{h}^{a_{1,X}} e^{-d_1} \stackrel{?}{=} c_{1,e}$;
8 $\quad \tilde{h}^{a_{-1,X}} e^{-d_{-1}} \stackrel{?}{=} c_{-1,e}$;
9 $\quad \text{Enc}(0, a_{-1,X})(X_i X_{i-1})^{-d_{-1}} \stackrel{?}{=} c_{-1,X}$;
10 $\quad \text{Enc}(0, a_{-1,Y})(Y_i Y_{i-1})^{-d_{-1}} \stackrel{?}{=} c_{-1,Y}$;
11 **if** so **then return** 1 **else return** 0;

4.3.2 Universal verifiability

An interesting property of the CSZ protocol is its universal verifiability. Indeed, each step of the protocol produces a transcript and some ZKP that allow any external auditor to check that the correct operations were performed. For instance, the round of communications (line 3 to 6) produce the transcript $\pi^{RC} = (X_i, e_i, Y_i, \pi_i)_{i=1}^{n_T}$, where π_i is a ZKP that there exists X_i, e_i, Y_i are well-formed. Afterwards, the rerandomization produces the transcript $\pi^{RR}(U_i, V_i, \pi_i^0)_{i=1}^{n_T}$ where U_i, V_i are some ciphertexts and π_i^0 is a ZKP proof that they are encryption of 0. By verifying all these proofs and computing $X' = X_i \prod_{i=1}^{n_T} U_i$ and $Y' = \prod_{i=1}^{n_T} V_i$, the verifier can deduce the result of the rerandomization phase and is guaranteed that X' and Y' are well-formed. Finally, the threshold decryption produces the transcript $\pi^{TD} = (w_i, \pi_i^{\text{Dec}})_{i=1}^{n_T}$, where w_i is i's partial decryption of Y' and π_i^{Dec} is a ZKP that it is well-formed. By verifying those proofs and combining the partial decryptions, the verifier can deduce the value Z of the output and be assured that Z is indeed an encryption of xy as desired. The final transcript of CSZ is $\pi^{\text{CSZ}} = \pi^{RC}||\pi^{RR}||\pi^{TD}$.

4.3.3 Comparison with the multiplication protocol

At a first glance, the conditional gate seems less useful than the multiplication protocol presented in Section 4.2. Indeed, to compute a multiplication on encrypted data with the conditional gate, we need to evaluate an already complex boolean circuit with many conditional gate sub-protocols; by contrast, this requires a single call of Mul. In this section, we give our main motivations for choosing the conditional gate anyway. Before that, however, we give a few metrics that are going to be useful in the remaining of this thesis.

Some metrics to evaluate the complexity of an MPC protocol. A first metric is the computational complexity. In asymmetric cryptography, the most prominent operation is the exponentiation, therefore we evaluate the complexity by counting (approximately) the number of exponentiations. Now, another important thing to assess in an interactive protocol is the

Table 5: Complexity estimate for the CSZ in a group G of order q and the Mul protocol with the public key n, where n_T is the number of participants.

	# exponentiations per participant	# synchronization steps	transcript size (# bits)
Mul	$9n_T + 4$	2	$17n_T\|n\|$
CSZ	$33n_T$	$n_T + 2$	$34n_T\|q\|$

communication complexity. This is more difficult because several factors can be relevant, such as the communication delay, the uploading / downloading speed and even the available memory. A first indicator is the total size of the messages exchanged. To evaluate this in the context of electronic voting, we give the size of the verification transcript, which is a collection of scalars and group elements that allow anyone to verify that the protocol was executed correctly, and therefore that its output is valid. This transcript gives a fair estimate of the complexity of the communications, as it is closely related to the total size of the messages exchanged. However, another important metric to consider is the number of synchronization steps. Indeed, in a purely asynchronous protocol, everyone can perform their computation locally and send and receive whenever it is needed. By contrast, if a participant must wait for the other participants to send some contribution before they can continue to operate, then the protocol becomes less efficient.

With the above metrics, the conditional gate protocol seems less efficient than the Mul protocol (see Tab. 5). The main difference comes from the fact that CSZ requires a round of communication when then authorities must in turn perform some computations, while Mul only consists of a few broadcasts. In MPC, reducing the number of communications is crucial; however, we do not have that many talliers: in general, $n_T \leq 5$. Therefore, provided that several CSZ protocols are computed in parallel, the efficiency loss can be amortized.

Comparison of the Paillier and ElGamal cryptosystems. Although the conditional gate protocol seems less interesting than the multiplication protocol in all aspects, we recall that the CSZ can be used with the exponential ElGamal encryption, while the Mul protocol requires the Paillier encryption scheme. This was our main motivation when choosing the CSZ protocol, and there are many reasons for this.

The computational assumption. First, the Paillier encryption scheme is based on DCRA, while the ElGamal encryption scheme is based on the DDH assumption. The DCRA is not as old as the DDH assumption; in addition, it is only used for Paillier encryption, while there are a lot of cryptographic schemes that rely on the DDH assumption. For this reason, although it is currently believed that attacking DCRA requires to factor n, it is possible that it might actually be easier. In addition, there are sub-exponential algorithms for factorization, while this is not the case for DDH on well-chosen elliptic curves. The consequence is that, for the same level of security, it is recommended to choose a 3072-bits n, while the size of the group in an ElGamal setting can be a 256-bits prime number q. Clearly, computing an exponentiation with a 3072-bits exponent modulo a number of 6144 bits is not the same as computing an exponentiation with a 256-bits exponent in an elliptic curve over $\mathbb{Z}_p$, where p is another 256-bits prime number. In Table 6, we provide estimates based on a medium level of optimization, for a native implementation on a modern processor (based on OpenSSL and using RSA for Paillier emulation), and for a JavaScript implementation running in a modern web browser (based on libsodium.js and JS BigInt). Since an ElGamal encryption requires two exponentiations, a raw estimate is that a Paillier encryption is about 1 250 times more expensive than an ElGamal encryption on the voter side.

Available libraries. Another reason to choose ElGamal over Paillier is the availability of widely used libraries that support elliptic curve cryptography, such as Libsodium, OpenSSL or

Table 6: Estimated number of exponentiation per second in the Paillier and ElGamal setting

	Paillier	Elliptic curve ElGamal
Server-side	200	10 000
Voter-side	2	5 000

Crypto++. By contrast, although some libraries that support the Paillier encryption scheme can be found, they are not used by a large community that would have detected most of the vulnerabilities. In Ordinos [Ord], they implemented the Paillier cryptosystem from scratch, based on the gmp library. In electronic voting, the need for a well-studied library is all the more so important because we usually need one on the voter side, and one on the server side.

Distributed key generation. A related difficulty is that the DKG in the Paillier setting is really complex and difficult to implement; by contrast, Pedersen's DKG is more simple and is often used in electronic voting. If we want to use an electronic voting system based on the Paillier encryption scheme in the future, it means that we would need to produce a safe implementation of the DKG, which is not done in the implementation of Ordinos.

Conclusion. In this thesis, we demonstrate that it is possible to make a reasonable use of the ElGamal encryption scheme in MPC. Switching to ElGamal is a win some, lose some decision: the communication complexity becomes larger and the dependency on the size of the inputs is more important. However, it can be greatly beneficial in some situations, which include electronic voting protocols.

4.4 Security of the conditional gate in the SUC framework

The conditional gate is our main building block for our toolbox. In order to build confidence on the resulting protocols, we use an universally composable security framework, introduced in Section 3.3. In this section, we prove the SUC-security of the conditional gate protocol, which is stated in Theorem 3. To make the proof as easy to follow as possible, we use a comprehensible proof strategy and use the composition theorem.

4.4.1 Proof strategy for the conditional gate

To assess the security of any protocol in the SUC framework, a natural strategy is to use the following steps.

1 Definition of the ideal functionality. First, we give $\mathcal{F}_{CSZ}$, the description of the trusted party that realizes the *conditional set to zero* functionality in the ideal process. As explained in Section 3.3, a SUC-secure protocol is not necessarily secure; rather, it is *as secure* as the ideal protocol. For this reason, it is important to provide an easy to analyze ideal functionality. We give Algorithm 43, in which the command abort causes the ideal functionality to erase any local data and send $\perp$ to all the participants as well as the adversary. This ideal functionality works as closely as possible as a trusted party: it collects the inputs of the participants, check their consistency and return the desired output. However, pk is supposed to be the public key, $(h_i)_{i=1}^{n_T}$ the public commitments of the shares of the participants and X and Y the two ciphertexts to operate on. Therefore, whenever a participant communicates with $\mathcal{F}_{CSZ}$, we consider that this part of the message can be read by the adversary (recall that the adversary can read a public part of the message when a participant communicates with the ideal functionality, but not the totality of the message). Remark that the ideal functionality can abort even if there is a majority

of honest participants, which means that we do not guarantee fairness. In addition, the abortion message $\perp$ does not allow to blame anyone, which means that we do not provide accountability.

Algorithm 43: $\mathcal{F}_{CSZ}$

Requires: G, a group of prime order q

1 **On** *message* $(g,h),(h_j)_{j=1}^{n_T},s,X,Y$ *from participant* i**:**
2 Send $(g,h),(h_j)_{j=1}^{n_T},X,Y$ to $\mathcal{S}$;
3 **if** $g^s \neq h_i$ **then** abort;
4 $X_i \longleftarrow X$; $Y_i \longleftarrow Y$; $s_i \longleftarrow s$;
5 **if** $X_j \neq \perp$ *for all* j **then**
6 Check that the received $(g,h),(h_j)_{j=1}^{n_T}$ are all the same (if not, abort);
7 **if** there exists j_1, j_2 s.t. $X_{j_1} \neq X_{j_2}$ or $Y_{j_1} \neq Y_{j_2}$ **then** abort;
8 Using the shares, decrypt X_1 and Y_1 into x and y;
9 $r \xleftarrow{\$} \mathbb{Z}_q$; $Z \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(xy,r)$;
10 Send Z to all participants and $\mathcal{S}$;
11 **else** wait;

2 Definition of the hybrid process. The second step is to define the hybrid process; which allows to model the protocol in the SUC-framework. For this purpose, we need to define all the ideal functionalities that we are going to use: they define the main abstractions of the proof. In our case, we use the $\mathcal{F}_{RO}$-hybrid model, as already discussed in 3.3.3, as well as the $\mathcal{F}_{SB}$-hybrid model, which we discussed in Section 3.3.4, where we showed that it can be realized from the ideal broadcast functionality $\mathcal{F}_B$ and $\mathcal{F}_{RO}$.

In general, describing the hybrid process also requires to give the exact algorithm of the honest participants in the hybrid model. Since the conditional gate protocol is rather complex, we are going to decompose it into several sub-protocols and use the composition theorem.

3 Decomposition into several sub-protocols. As explained in Section 4.3, the conditional gate protocol is divided into three parts: the round of communications (lines 1 to 8), the rerandomization (line 9) and the threshold decryption (line 10). Then, a natural way to decompose the protocol is to analyze the three parts separately, which will be done in the remaining on this chapter. Each part has its dedicated sub-section but, for the purpose of the proof, we do not treat them in the chronological order.

4 Restrictions on the environment. Finally, as we mentioned in Section 3.3.2, it is sometimes necessary to impose a restriction on the environment. For the conditional gate, the condition is that $y \in \{0,1\}$. Therefore, we demand that the input of the participants is of the form $(g,h),(h_j)_{j=1}^{n_T},s_i,X,Y$ such that Y is an exponential ElGamal encryption of 0 or 1 obtained with the public key (g,h) (*i.e.* an ElGamal encryption of either 1_G or g). In addition, we require that $(g,h),(h_j)_{j=1}^{n_T},X,Y$, which is supposed to be a public input, is the same for all the participants. To simplify the presentation, we also demand that $(g,h),(h_j)_{j=1}^{n_T},s_i$ is the output of a DKG, *i.e.* that there exists a polynomial f of degree t such that $g^{f(j)} = h_j$ for all j, with $g^{f(0)} = h_0$ and $f(i) = s_i$. This additional condition is not only decidable, but also efficiently so; therefore the participants can check it themselves and abort if it is not met. However, those additional checks may distract the reader from the important ones.

Algorithm 44: RR (algorithm of i)

Requires: G, a group of prime order q
Inputs: pk, an exp. ElGamal key
A ciphertext X

1 **On** *input* $(g, h), X$**:**
2 Start a new independent session;
3 $r_1 \xleftarrow{\$} \mathbb{Z}_q$; $\alpha \xleftarrow{\$} \mathbb{Z}_q$;
4 $U_i \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(0, r_1)$;
5 $c_u \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(0, \alpha)$;
6 Query $\mathcal{F}_{RO}$ with $(\mathsf{pk}||X||U_i||c_u)$;
7 Wait for the answer d;
8 $a_u \longleftarrow \alpha + r_1 d$;
9 Query $\mathcal{F}_{SB}$ with U_i, c_u, a_u;
10 **On** *message* $(U_j, \pi_j)_{j=1}^{n_T}$ *from* $\mathcal{F}_{SB}$**:**
11 **for** $j = 1$ *to* n_T **do**
12 $c_{uj}, a_{uj} \longleftarrow \pi_j$;
13 Verify the PoK:
14 Query $\mathcal{F}_{RO}$ with $(\mathsf{pk}||X||U_j||c_{uj})$;
15 Wait for the answer d;
16 **if** $c_{uj} \neq \mathrm{Enc}_{\mathsf{pk}}(0, a_{uj}) U_j^{-d}$
17 **then** Output $\perp$;
18 Output $X \prod_{i=1}^{n_T} U_i$;

Algorithm 45: $\mathcal{F}_{rerand}$

Requires: G, a group of prime order q
1 **On** *message* pk, X *from* i**:**
2 $\mathsf{pk}_i \longleftarrow \mathsf{pk}$; $X_i \longleftarrow X$;
3 **if** $X_j \neq \perp$ *for all* j **then**
4 **if** $X_j = X_1$ **and** $\mathsf{pk}_j = \mathsf{pk}_1$ $\forall j$ **then**
5 $\alpha \xleftarrow{\$} \mathbb{Z}_q$;
6 $X' \longleftarrow \mathrm{Renc}_{\mathsf{pk}_1}(X_1, \alpha)$;
7 Send X' to all j and to $\mathcal{S}$;
8 **else** abort;
9 **else** wait;
10 **On** *message* i *from* $\mathcal{S}$**:**
11 Send pk_i, X_i to $\mathcal{S}$;

4.4.2 The rerandomization

We start with the easiest phase, which is the rerandomization phase. We show in Lemma 7 that it SUC-securely computes the $\mathcal{F}_{rerand}$ ideal functionality, defined in Algorithm 45. This ideal functionality outputs $\perp$ if the participants do not agree on a common public key pk and a common ciphertexts X (since the participants need to rerandomize two ciphertexts, they will need to call the ideal functionality twice). If they do, it outputs a random rerandomization X' of X. Also, since the inputs of the participants are supposed to be a common public pk, X, this ideal functionality allows the adversary to learn the input of each participant.

Lemma 7. *Assuming that there is at least one honest participant, the rerandomization sub-protocol described in Algorithm 44 SUC-securely computes the $\mathcal{F}_{rerand}$ ideal functionality (given in Algorithm 45) in the $\mathcal{F}_{RO}, \mathcal{F}_{SB}$-hybrid model, where $\mathcal{F}_{SB}$ is defined in Algorithm 28.*

Proof. We construct the simulator $\mathcal{S}$ which interacts with the environment in the ideal process, and simulates the hybrid process.

First, the simulator acts in the ideal process and forwards the messages of all the honest participants, which allows it to learn their inputs from $\mathcal{F}_{rerand}$. With this knowledge, it runs a perfect simulation of the RR protocol, up until when it has to reveal the answer of $\mathcal{F}_{SB}$ to a corrupted participant at line 10. At this moment, the simulator checks that the honest participants all had the same input pk, X. To begin with, suppose that this is not the case, which is **Case 1**. Then the simulator continues the perfect simulation and, whenever a simulated

honest participant outputs something in the simulated hybrid process, $\mathcal{S}$ forwards the answer of $\mathcal{F}_{rerand}$ (which is necessarily $\perp$) to the same participant in the ideal process. This way, the said participant outputs $\perp$ in the ideal process. Since the simulator runs a perfect simulation of the hybrid process, it remains to show that when two participants do not have the same input in the hybrid process, then the output of any honest participant (if any) is $\perp$ with overwhelming probability.

Case 1: no consensus. If two honest participants, say i and j, have two different inputs pk_i, X_i and pk_j, X_j then, for all honest participant k, either the PoK π_i or the PoK π_j will appear invalid (except with negligible probability). Indeed, pk_k, X_k cannot be simultaneously equal to pk_i, X_i and pk_j, X_j. Without a loss of generality, assume that $(\mathsf{pk}_k, X_k) \neq (\mathsf{pk}_i, X_i)$. Then $\mathcal{F}_{RO}$, when queried with $(\mathsf{pk}_k||X_k||U_i||c_{ui})$ outputs a different answer than when queried with $(\mathsf{pk}_i||X_i||U_i||c_{ui})$, except with a negligible probability. Let d_k and d_i be the two different answers. Since the proofs are generated honestly, we have $c_{ui} = \mathrm{Enc}_{\mathsf{pk}_i}(0, a_{ui})U_i^{-d_i}$. Except with negligible probability, this is different from $\mathrm{Enc}_{\mathsf{pk}_k}(0, a_{ui})U_i^{-d_k}$, therefore k rejects the proof as invalid and outputs $\perp$, except with a negligible probability.

Case 2. Now, suppose that all the honest participants have the same input pk, X. Then, for all corrupted participant j, the simulator looks for a query to $\mathcal{F}_{RO}$ of the form $(\mathsf{pk}||X||U_j||c_{uj})$, which was answered by some d_j such that $c_{uj} = \mathrm{Enc}_{\mathsf{pk}}(0, a_{uj})U_j^{-d_j}$ (*i.e.* the PoK π_j is valid). If there is no such query for some j, then the corresponding proof will look invalid to all the honest participants, except with negligible probability. In this case, all the honest participants will output $\perp$ in the hybrid process. To have the same output in the ideal process, the simulator makes a query from all the corrupted participants, but with an input $(\mathsf{pk}', X') \neq (\mathsf{pk}, X)$. This way, the ideal process answers $\perp$ to all the participants as desired.

If there is such a query for all j, the proof will appear valid to all the honest participants, which will therefore output $X \prod_{i=1}^{n_T} U_i$ in the hybrid process. To have this match the output of the ideal process, the simulator first sends the query (pk, X) to $\mathcal{F}_{rerand}$ with all the corrupted participants, so that $\mathcal{F}_{rerand}$ answers with some X'. However, the simulator blocks all the answers towards a honest participant: it will deliver them one by one, when it will need a honest participant to output X' in the ideal process. Then, the simulator changes the contribution of a single honest participant i in the simulation, and sets $U_i = X'/(X \prod_{j \neq i} U_i)$. Also, using the control over the random oracle, it simulates the PoK π_i so that it appears valid to the adversary. For this purpose, it chooses a challenge d at random and the answer $a \in \mathbb{Z}_q$ at random as well. Then, it computes $c_u = \mathrm{Enc}_{\mathsf{pk}}(0, a)U_i^{-d}$. Since d was chosen at random, then (except with a negligible probability) no query was made to $\mathcal{F}_{RO}$ with the input $\mathsf{pk}||X||U_i||c_u$, so that the simulator can answer every subsequent such query with d. The simulated proof is then $\pi = (c_u, a)$. Remark that since π_j is valid for all j, then, by the computational soundness of the PoK, U_j is encryptions of 0 for all j, except with a negligible probability. Consequently, $\prod_{j \neq i} U_j$ is an encryption of 0. Also, since X' is a random reencryption of X, X'/X is a random encryption of 0. Therefore, U_i is also a random encryption of 0. Hence, by the zero knowledge property of the PoK, U_i, π follows the same distribution as in the real hybrid process (except with a negligible probability).

Conclusion. The above simulator gives a perfect simulation of the hybrid process, except with a negligible probability. In addition, the outputs of the honest participants are the same in the simulated hybrid and in the ideal process. Therefore, the view of the environment is the same in both the hybrid and the ideal processes, except with a negligible probability. □

4.4.3 The threshold decryption

We now address the threshold decryption part, whose goal is to evaluate the ideal functionality $\mathcal{F}_{\text{Dec}}$ given in Algorithm 47. Compared to the "ideal" ideal functionality, this one lives in a setting where each participant i has the result $\mathsf{pk}, (h_j)_{j=1}^{n_T}, s_i$ of a DKG as an input, as well as a ciphertext Y to decrypt. In this input, only the secret share s_i is private so that the ideal functionality allows the adversary to learn the remaining (public) part. Apart from that, $\mathcal{F}_{\text{Dec}}$ is similar to $\mathcal{F}_{\text{CSZ}}$: it collects the inputs of the participants, checks their consistency and returns the desired output, which is the decryption of the common ciphertext Y.

A subtle difficulty is that the threshold decryption protocol in the ElGamal setting is not universally composable. Indeed, although we exhibited a simulator in Theorem 1, the latter suffers from two major flaws which are detrimental for the sake of SUC-security. First, the simulated partial decryptions do not follow the same distribution as the real partial decryptions. Yet, in a UC framework, the environment chooses the inputs of the participants (and therefore their shares), which allows it to distinguish the simulated partial decryption from the real ones. A similar difficulty is that the simulator also chooses the ciphertext to decrypt, while Theorem 1 only gave a notion of security against chosen *plaintext* attacks – and not chosen ciphertexts. Therefore, the threshold decryption protocol is only SUC-secure if the adversary corrupts *exactly* t participants: if less participants are corrupted, the simulator does not have access to enough secret share to perform the polynomial interpolation in the exponent.

This is when our rerandomization phase comes to the rescue. In Lemma 8, we show that if the decryption protocol is preceded by a (perfect) rerandomization phase, then it achieves SUC-security.

Lemma 8. *The threshold decryption protocol described in Algorithm 46 SUC-securely computes $\mathcal{F}_{\text{Dec}}$ (defined in Algorithm 47) in the $\mathcal{F}_{rerand}, \mathcal{F}_{RO}$-hybrid model.*

Proof. We construct a simulator $\mathcal{S}$ which interacts with the environment in the ideal process and simulates the hybrid process by simulating the honest participants and the $\mathcal{F}_{rerand}, \mathcal{F}_{RO}$ ideal functionalities. First, the simulator acts in the ideal process and forwards all the messages of the honest participants to $\mathcal{F}_{\text{Dec}}$ in order to get $(g, h), (h_j)_{j=1}^{n_T}, Y$. If the data of the honest participants are not consistent, the simulators can run a perfect simulation of the hybrid process, since $\mathcal{F}_{rerand}$ will output $\perp$ which will cause all the honest participants to output $\perp$ as in the ideal process. Consequently, we suppose that all the honest participants have the same $(g, h), (h_j)_{j=1}^{n_T}, Y$. Then the simulator uses the corrupted participants of the ideal process and forwards their inputs to the ideal functionality, which causes it to send the plaintext y to everyone. However, $\mathcal{S}$ blocks this answer to everyone, except for itself: it will deliver the answers one by one, when it will need a honest participant to output y in the ideal process.

Now that $\mathcal{S}$ knows the plaintext y that corresponds to Y, it picks $r \in \mathbb{Z}_q$ at random and compute $u = g^r$ as well as $v = yh^r$, so that $Y' = (u, v)$ is a random reencryption of Y. Using this Y', $\mathcal{S}$ can run a perfect simulation of $\mathcal{F}_{rerand}$.

After the rerandomization phase, $\mathcal{S}$ has to simulate the actual threshold decryption protocol, except that it does not know the secret share of the honest participants. Let i be a honest participant. When i receives Y' from $\mathcal{F}_{reerand}$ in the simulated hybrid process, $\mathcal{S}$ computes $w_i = h_i^r$, chooses $a \in \mathbb{Z}_q$ at random as well as the challenge d. Then, $\mathcal{S}$ computes $c_g = g^a h_i^{-d}$ and $c_u = u^a w_i^{-d}$. Since those two are random, no query to $\mathcal{F}_{RO}$ was made with the input $(g, h)||Y'||w_i||c_g||c_u$ (except with a negligible probability) so that the simulator can answer all subsequent such queries with d. Now, since (g, u, h_i, w_i) is a DDH tuple, (c_g, c_u, d, a) follows the

Algorithm 46: TD (algorithm of i)

Requires: G, a group of prime order q
Inputs: (g,h), an ElGamal public key
$(h_j)_{j=1}^{n_T}$, the commitments on the shares of the participants
s_i, the secret share of participant i
Y, a ciphertext

On *input*:
  Start a new independent session;
  Send $(g,h), Y$ to $\mathcal{F}_{rerand}$;
On $\perp$ from $\mathcal{F}_{rerand}$: Output $\perp$;
On *message* Y' *from* $\mathcal{F}_{rerand}$:
  Parse Y' as (u,v);
  $w_i \longleftarrow u^{s_i}$;
  Compute the PoK:
  $\alpha \xleftarrow{\$} \mathbb{Z}_q$;
  $c_g \longleftarrow g^\alpha$; $c_u \longleftarrow u^\alpha$;
  Query $\mathcal{F}_{RO}$ with $((g,h)||Y'||w_i||c_g||c_u)$;
  Wait for the answer d;
  $a \longleftarrow \alpha + ds_i$;
  Send (w_i, c_g, c_u, a) to all j;
On *message* (w, c_g, c_u, a) *from* j:
  Query $\mathcal{F}_{RO}$ with $((g,h)||Y'||w||c_g||c_u)$;
  Wait for the answer d;
  if $c_g \neq g^a h_j^{-d}$ **or** $c_u \neq u^a w^{-d}$
  then Output $\perp$ **else** $w_j \longleftarrow w$;
  if $\exists S \subset [1, n_T]$ *s.t.* $|S| = t+1$ ***and*** $\forall j \in S, w_j \neq \perp$ **then**
    for $j \in S$ **do** $\Lambda_j \longleftarrow \prod_{k \in S \setminus \{j\}} \frac{k}{j-k}$;
    $y \longleftarrow v \prod_{j \in S} w_j^{\Lambda_j}$;
    Output y;
  else wait;

Algorithm 47: $\mathcal{F}_{\text{Dec}}$

Requires: G, a group of prime order q
On $(g,h), (h_j)_{j=1}^{n_T}, s, Y$ *from* i:
  Send $(g,h), (h_j)_{j=1}^{n_T}, Y$ to $\mathcal{S}$;
  if $g^s \neq h_i$ **then** abort;
  $Y_i \longleftarrow Y$; $s_i \longleftarrow s$;
  if $Y_j \neq \perp$ *for all* j **then**
    Check that the received (g,h), $(h_j)_{j=1}^{n_T}$ are all the same
    if not **then** abort;
    if there exists j_1, j_2 s.t. $Y_{j_1} \neq Y_{j_2}$ **then** abort;
    Decrypt Y_1 into y;
    Send y to all participants and $\mathcal{S}$;
  else wait;

same distribution as in the real hybrid process (this is the zero knowledge property of the ZKP), therefore the simulation is perfectly indistinguishable from the real process.

Finally, when a (simulated) honest participant i receives (w, c_g, c_u, a) from some j, the simulator runs the algorithm of the participant to decide whether it should output $\perp$, output some value y' computed from the received shares or wait. If the participant has to output $\perp$, it means that j was corrupted. Then $\mathcal{S}$ uses j in the ideal process to send a query to $\mathcal{F}_{\text{Dec}}$, but with an inconsistent s_j. This way $\mathcal{F}_{\text{Dec}}$ sends $\perp$ to all participants and $\mathcal{S}$ can block every answer, except for i which will therefore output $\perp$ in the ideal process. If i has to wait, then $\mathcal{S}$ makes it wait. However, if i has to output something, it outputs $y' = v \prod_{j \in S} w_j^{\Lambda_j}$ while it can only output y in the ideal process. Fortunately, for all j in S, the PoK of correct partial decryption is valid. Therefore, by the soundness of the ZKP, (except with a negligible probability) there exists $s_j \in \mathbb{Z}_q$ such that $g^{s_j} = h_i$ and $u^{s_j} = w_j$. Hence, except with a negligible probability, $y' = y$ (this comes from the Lagrange interpolation of $f(0)$). □

4.4.4 The round of communications

The final part is the round of communication. Since we could not find a smart ideal functionality that is realized by this part, we conclude the proof by giving Lemma 9, which states the SUC-security of Algorithm 48, which is the conditional gate protocol in the $\mathcal{F}_{RO}, \mathcal{F}_{\text{Dec}}$-hybrid process. In this Algorithm, Rnd can be derived from Algorithm 39 (lines 3 to 6) and Algorithm 41, and Ver-CSZ can be derived from Algorithm 42. Rnd allows a participant to produce X_i, Y_i, e and to prove that they are well-formed; Ver-CSZ allows to verify the ZKP.

Compared to the protocol presented in Algorithm 39, we can see that the participants broadcast $(X_{i-1}, Y_{i-1}, X_i, Y_i, e, \pi)$ instead of just (X_i, Y_i, e, π). This allows them to synchronize their view "on the fly", without adding too many synchronization steps at each broadcast. The price to cost is that at the end of the round of communications, all the participants may not agree on the same X_{n_T}, Y_{n_T}.

Another difference is that in Algorithm 39, the participants simultaneously rerandomize X_{n_T} and Y_{n_T} into X' and Y', while the two rerandomization got somehow separated in Algorithm 48: one is done right away and the other one is *consumed* by $\mathcal{F}_{\text{Dec}}$ (see Section 4.4.3). This is purely for the sake of the presentation: since the two rerandomizations are independent, they can actually be done simultaneously.

Finally, in the SUC framework, the environment is allowed to choose freely the inputs of the participants which, for convenience, include g. Yet, recall that we said in Section 4.3 that g must be public coin (otherwise we would need another version of DDH, which would also be acceptable). Therefore, at the beginning of the protocol, the participants get g from the random oracle and check that it is consistent with their input. Note that, to be able to write g in the input of the participants, the environment must first query it to the random oracle, using the adversary or a corrupted participant.

Lemma 9. *Assuming that there is at least one honest participant, and under the DDH assumption, the protocol depicted in Algorithm 48 SUC-securely computes $\mathcal{F}_{\text{CSZ}}$ (defined in Algorithm 43) in the $\mathcal{F}_{RO}, \mathcal{F}_{rerand}, \mathcal{F}_{\text{Dec}}$-hybrid model.*

Proof. We construct a simulator $\mathcal{S}$ which interacts with the environment in the ideal process and simulates the hybrid process by simulating the honest participants and the $\mathcal{F}_{RO}, \mathcal{F}_{\text{Dec}}$ ideal functionalities. First, the simulator chooses a random $g \in G$ and, whenever $\mathcal{F}_{RO}$ is queried with "Conditional Gate", the simulator answers with g. Also, whenever $\mathcal{F}_{RO}$ is queried with a new input of the form $(g||h)$, $\mathcal{S}$ chooses a random trapdoor τ, computes $\tilde{h} = g^{1/\tau}$ and answers with

Algorithm 48: CSZ (algorithm of participant i)

Requires: G, a group of prime order q
Inputs: $(g,h),(h_j)_{j=1}^{n_T},s_i,X,Y$
Variables: Two ciphertexts pr_x^j and pr_y^j for all $j\neq i$ (initially, $\perp$)

On *input*:
  Start a new independent session;
  Query $\mathcal{F}_{RO}$ with "Conditional Gate";
  Check that the answer is g
  (otherwise, Output $\perp$);
  $E_{-1}\longleftarrow(1_G,g^{-1})$;
  $X_0\longleftarrow X$; $Y_0\longleftarrow E_{-1}Y^2$;
  Query $\mathcal{F}_{RO}$ with $(g||h)$;
  Wait for the answer $\tilde{h}$;
  if $i>1$ **then** change to Waiting 1, wait;
  else
    $X_1,e,Y_1,\pi_1\longleftarrow\mathrm{Rnd}(X_0,Y_0,\tilde{h})$;
    Change state to Waiting 2;
    Send $(X_0,Y_0,X_1,e,Y_1,\pi_1)$ to all j;

State *Waiting* 1:
  On (A,B,C,e,D,π) *from* $j<i$:
    if $X_j=\perp$ **then**
      $X_j\leftarrow C$, $e_j\leftarrow e$;
      $Y_j\leftarrow D$; $\pi_j\leftarrow\pi$;
      $\mathrm{pr}_x^j\leftarrow A$; $\mathrm{pr}_y^j\leftarrow B$;
      Ignore all future messages from j;
    if $X_k\neq\perp$ *for all* $k<i$ **then**
      for $j=1$ *to* $i-1$ **do**
        if $X_{j-1}\neq\mathrm{pr}_x^j$ **or** $Y_{j-1}\neq\mathrm{pr}_y^j$ **then** Output $\perp$;
      $X_i,e,Y_i,\pi\leftarrow\mathrm{Rnd}(X_{i-1},Y_{i-1},\tilde{h})$;
      Change state to Waiting 2;
      Send $(X_{i-1},Y_{i-1},X_i,e,Y_i,\pi)$ to all;
    else wait;

State *Waiting* 2:
  On $A,B,C,e,D,\pi)$ *from* $j>i$:
    if $X_j=\perp$ **then**
      $X_j\leftarrow C$, $e_j\leftarrow e$;
      $Y_j\leftarrow D$; $\pi_j\leftarrow\pi$;
      $\mathrm{pr}_x^j\leftarrow A$; $\mathrm{pr}_y^j\leftarrow B$;
      Ignore all future messages from j;
    if $X_k\neq\perp$ *for all* $k>i$ **then**
      for $j=i+1$ *to* n_T **do**
        if $X_{j-1}\neq\mathrm{pr}_x^j$ **or** $Y_{j-1}\neq\mathrm{pr}_y^j$ **then** Output $\perp$;
      Check all the PoK:
       for $j=1$ *to* n_T **do**
        if Ver-CSZ$(\mathsf{pk},X_{i-1},Y_{i-1},X_i,Y_i,e,\pi_i)=0$ **then** Output $\perp$;
      Change state to Decrypt;
      Send $(g,h),X$ to $\mathcal{F}_{rerand}$;
    else wait;

State *Decrypt*:
  On $\perp$ from $\mathcal{F}_{rerand}$ Output $\perp$;
  On *message* X' *from* $\mathcal{F}_{rerand}$:
    Send $(g,h),(h_k)_k,s_i,Y_{n_T}$ to $\mathcal{F}_{\mathrm{Dec}}$;
  On $\perp$ from $\mathcal{F}_{\mathrm{Dec}}$ Output $\perp$;
  On *message* g^y *from* $\mathcal{F}_{\mathrm{Dec}}$:
    Output $(XX'^y)^{1/2}$;

$\tilde{h}$. This way the simulation is perfectly indistinguishable from the real hybrid (if $\tau = 0$, $\mathcal{S}$ sets $\tilde{h}$ to 1_G). At some point, the environment must activate a honest participant by writing on its input tape, which fixes $(g, h), (h_i)_{i=1}^{n_T}, X, Y$ for the session. (If the same participant is activated several times, the simulator runs several independent sessions. This assumes, for instance, that a different prefix is used for querying $\mathcal{F}_{RO}$ in each session.) Now that the protocol has really began, we explain how to simulate the different states.

Simulation until Waiting 2. Let i be the last honest participant (*i.e.* for all $j > i$, participant j is corrupted). The simulator runs a perfect simulation of the round of communications, up until when i has to change to the state "Waiting 2". This can happen at line 13 or line 27. In any case, for all $j < i$, participant j revealed its contribution X_j, e_j, Y_j, π_j. Before revealing the contribution of i, the simulator checks all the ZKP. If one is invalid, then all the honest participants will output $\perp$ at line 25, 40 or 42, therefore the simulator will not have to simulate the decryption. Hence, the best course of action is to continue the perfect simulation without cheating, until every honest participant outputs $\perp$.

If all the proof are valid then the computational soundness guarantees that, except with a negligible probability, there exists $r_1, r_2 \in \mathbb{Z}_q$ and $s \in \{-1, 1\}$ such that $X_{i-1} = \text{Renc}_{\mathsf{pk}}(X_0^s, r_1)$ and $Y_{i-1} = \text{Renc}_{\mathsf{pk}}(Y_0^s, r_2)$. The simulator first acts in the ideal process and forwards all the messages of the honest participants to the ideal functionality $\mathcal{F}_{\text{CSZ}}$. Also, it instructs the corrupted participants to send their inputs to $\mathcal{F}_{\text{CSZ}}$ as well, so that $\mathcal{F}_{CSZ}$ answers with some ciphertext Z_f. Note that due to the restrictions on the environment, $\mathcal{F}_{\text{CSZ}}$ does not abort. As usual, the simulator blocks the answer towards all the participant except itself: it will deliver them when it will need a honest participant to output Z_f in the ideal process.

Since Z_f is the output of the ideal process, the couple $Z_f, (1_G, g)$ is such that Z_f is a reencryption of X_{i-1}^y and $(1_G, g)$ is a reencryption of Y_{i-1}^y, where $y = \text{Dec}_{\mathsf{sk}}(Y_{i-1})$ (except with a negligible probability since this comes from the soundness of the ZKP). However, this couple is not random enough and the environment might notice that a trivial encryption of 1 is used. Therefore the simulator rerandomizes it by choosing a random $s' \in \{-1, 1\}$, two random $\alpha, \beta \in \mathbb{Z}_q$ and computing $X_i = \text{Renc}_{\mathsf{pk}}(Z_f^{s'}, \alpha)$ and $Y_i = \text{Renc}_{\mathsf{pk}}((1_G, g^{s'}), \beta)$. This way, X_i and Y_i becomes independent from Z_f and y, and follow the correct distribution. We denote $X_i = (u_{x,i}, v_{x,i})$ and $X_{i-1} = (v_{x,i-1}, v_{x,i-1})$.

At this point, there is a single value of e_i for which X_i, e_i, Y_i is well-formed, but this value depends on y: $e_i = (u_{x,i}/u_{x,i-1}^{ys'})^\tau$. However, $\mathcal{S}$ has no way to know y. Therefore, it cannot produce a perfect simulation and will pick e as a uniformly random element instead.

Now, $\mathcal{S}$ has to forge a fake ZKP π_i, which is possible thanks to the control over $\mathcal{F}_{RO}$. However, since the statement to prove is most likely false, the forged ZKP does not follow the same distribution as the real one. Since the view of the environment is not the same as in the real hybrid process, we will need to prove that the simulated view is indistinguishable from the fake one.

Remark that the simulator created a situation where Y_i is an encryption of a known plaintext s', which will be useful in the remaining of the proof.

Simulation of Waiting 2. Since there are no honest participant left to simulate, the simulator can perform a perfect simulation of *Waiting 2*. Nevertheless, each time a participant $j > i$ sends a valid $X_{j-1}, Y_{j-1}, X_j, e_j, Y_j, \pi_j$, then the soundness of the ZKP assures the existence of $r_1, r_2 \in \mathbb{Z}_q$ and $s \in \{-1, 1\}$ such that $(u_{x,j}, v_{x,j}) = X_j = (g^{r_1} u_{x,j-1}^s, h^{r_1} v_{x,j-1}^s)$, $Y_j = \text{Renc}_{\mathsf{pk}}(Y_{j-1}^s, r_2)$ and $e_j = \tilde{h}^{r_1}$, where $(u_{x,j-1}, v_{x,j-1}) = X_{j-1}$. Hence, by computing $u_{x,j} e_j^{-\tau}$, $\mathcal{S}$ recovers either $u_{x,j-1}$ or $u_{x,j-1}^{-1}$ depending on s, which enables it to deduce the value of s used by j (recall that if the proof π_{j-1} is valid, then $u_{x,j-1} \neq 1$; $j - 1 > 0$ since $j > i \geq 1$).

To avoid the confusion with j's secret share, we denote it σ_j.

Simulation of the rerandomization of X. When a honest participant reaches the *Rerandomize* state, the simulator knows the value $y = s' \prod_{j>i} \sigma_j$ which is encrypted into Y_{n_T}. At this point, except with a negligible probability (if the adversary managed to forge a fake ZKP), the ciphertext $X' = (Z_f^2/X)^y$ is a "random" reencryption of X_{n_T}. (Indeed, the environment had no information about Z_f yet, therefore X' follows the correct distribution and is independent from the remaining of its view.) Hence the simulator can use his value as the output of $\mathcal{F}_{rerand}$ instead of a honestly generated reencryption.

Simulation of the decryption. To simulate the decryption, the simulator uses the real plaintext y. This way the output $(XX'^y)^{1/2}$ is indeed equal to Z_f.

Indistinguishability. We now prove that the simulation is indistinguishable from the real hybrid game. Before giving the reduction to DDH, we propose to dream up a bit and construct an imaginary simulator $\mathcal{S}_i$, which can compute a discrete logarithm. This simulator uses the same simulation as $\mathcal{S}$, except that for the last honest participant i, e_i is not chosen as a random group element. Indeed, since $\mathcal{S}_i$ can decrypt Y_{i-1}, it can use the "correct" value of e_i for which X_i, e_i, Y_i is well-formed. In turn, by the zero knowledge property of the ZKP, the simulated proof π_i will be perfectly indistinguishable from the real one. In fact, the tuple X_i, e_i, Y_i, π_i computed by $\mathcal{S}_i$ follows the same distribution as in the real hybrid process. Since the remaining of the simulation is perfect (except with a negligible probability), $\mathcal{S}_i$ creates a perfect simulation of the real hybrid process (except with a negligible probability). Hence, the environment can distinguish $\mathcal{S}$'s simulation from the real hybrid process if and only if it can distinguish the simulation from $\mathcal{S}$'s from $\mathcal{S}_i$'s.

Now, let $\mathcal{Z}$ be an environment and $\mathbb{A}$ be an adversary for DDH. (Recall that the "adversary" in the SUC framework is just the dummy adversary, so that only the environment is relevant.) We denote p and p_i the probability that $\mathcal{Z}$ outputs 1 when interacting with $\mathcal{S}$ and $\mathcal{S}_i$. The adversary $\mathbb{A}$ receives a challenge tuple g_1, g_2, g_3, g_4 in the DDH game. To decide whether it is a DDH tuple or not, it interacts with $\mathcal{Z}$ by simulation $\mathcal{S}$ as well as the corrupted participants. However, when $\mathcal{Z}$ queries $\mathcal{F}_{RO}$ with "Conditional Gate", $\mathbb{A}$ answers with $g = g_1$ (if $\mathcal{Z}$ creates several independent sessions, $\mathbb{A}$ can use a random $\alpha \in \mathbb{Z}_q$ and answer with $g = g_1^\alpha$ instead; in this case, it will also use g_3^α instead of g_3). In addition, whenever the environment makes a new query of the form $(g||h)$, $\mathbb{A}$ chooses a random $\tau \in \mathbb{Z}_q$ and computes $\tilde{h} = g_2^\tau$. This way, except if $g_2 = 1_G$ or $g_1 = 1_G$ (in which case the DDH challenge is trivial), $g, h, \tilde{h}$ follows the exact same distribution as in $\mathcal{S}$'s simulation.

At some point, the environment must write on the input tape of a participant, which fixes $(g, h), (h_j)_{j=1}^{n_T}$ for the session. Due to the restrictions on the environment, $\mathcal{Z}$ must write an input of the form $(g, h), (h_j)_{j=1}^{n_T}, s_k$ in the input tape of all the participants, which allows $\mathbb{A}$ to learn $\mathsf{sk} = \log_g(h)$ by combining all the secret shares.

Afterwards, $\mathbb{A}$ continues the simulation until it must reveal the contribution (X_i, e_i, Y_i) of the last honest participant. For this purpose, $\mathbb{A}$ parses X_{i-1} as $(u_{x,i-1}, v_{x,i-1})$, chooses a random $s \in \{-1, 1\}$ and computes $u_{x,i} = u_{x,i-1}^s g_3$ as well as $v_{x,i} = v_{x,i-1}^s g_3^{\mathsf{sk}}$, which defines $X_i = (u_{x,i}, v_{x,i})$. As for Y_i, $\mathbb{A}$ chooses r_2 at random and compute $Y_i = \mathrm{Renc}_{\mathsf{pk}}(Y_{i-1}^s, r_2)$. Finally, it sets e_i as g_4^τ, so that X_i, e_i, Y_i is well-formed if and only if g_1, g_2, g_3, g_4 is a DDH tuple. Then $\mathbb{A}$ continues the simulation normally, except that it cannot use τ to extract s_j for $j > i$, since $\tilde{h}^\tau \neq g$. However, it can extract s_j by decrypting Y_j and Y_{j-1} using sk: if the plaintexts are equal, $s_j = 1$; otherwise, $s_j = -1$.

At the end of the simulation, the environment outputs a bit b. If $b = 1$, $\mathbb{A}$ states that g_1, g_2, g_3, g_4 was a DDH tuple; otherwise, it states that the challenge tuple was a random tuple. Remark that when the challenge is a DDH tuple, $\mathbb{A}$ runs the same simulation as $\mathcal{S}_i$ and hence

wins with probability p_i; on the other hand, when the challenge is a random tuple, $\mathbb{A}$ runs $\mathcal{S}$'s simulation but must output 0 to win, therefore it wins with probability $1-p$. Hence $\mathbb{A}$'s probability to win the DDH game is $\frac{1}{2}(p'+1-p)$, so that $\mathbb{A}$'s advantage is $\frac{1}{2}|p'-p|$. Under the DDH assumption, $\mathbb{A}$'s advantage is negligible, therefore $|p'-p|$ is negligible, which concludes the proof. □

4.4.5 The conditional gate protocol is SUC-secure

Now that we proved that all the components of the conditional gate protocol are SUC-secure, the SUC-security of the protocol is a direct consequence of the composition theorem, given in Lemma 5. Indeed, by Lemma 9, we have the SUC-security provided that the threshold decryption protocol and the rerandomization are SUC-secure. In Lemma 8, we showed that the SUC-security of the threshold decryption can be derived from that of the rerandomization. Also, in Lemma 7, we showed that the SUC-security of the rerandomization is a consequence of that of the synchronous broadcast, whose security comes from $\mathcal{F}_B$ and $\mathcal{F}_{RO}$ by Lemma 6. When we compile all those results together, this gives Theorem 3, which is the desired result.

Theorem 3. *Under the DDH assumption, and if at least one participant is honest, the conditional gate protocol given in Algorithm 39 SUC-securely computes the $\mathcal{F}_{\text{CSZ}}$ ideal functionality given in Algorithm 43, in the $\mathcal{F}_{RO}, \mathcal{F}_B$-hybrid model, where $\mathcal{F}_{RO}$ is the programmable random oracle ideal functionality (see Algorithm 26) and $\mathcal{F}_B$ is the broadcast ideal functionality (see Algorithm 29).*

Proof. This is a direct consequence of Lemma 9, Lemma 8, Lemma 7, Lemma 6 and Lemma 5. □

Chapter 5

A toolbox for verifiable tally-hiding

The first contribution of this thesis is to provide a toolbox for verifiable tally-hiding in the ElGamal setting. This toolbox is built upon the *conditional gate* protocol from [ST04] which allows to securely realize the *conditionally set to zero* functionality on bitwise encrypted data (see Section 4.3). It consists of various MPC protocols which allow to realize several useful functionalities. For some functionalities, we propose several computation/communication trade-offs, which can mitigate the potentially expensive communication cost. Most of the considered protocols, as well as their respective complexity are displayed in Table 8. Since we proved the security of this building block in the SUC framework, it means that the security of the whole toolbox is guaranteed in the SUC-framework as well; in other words, we can prove that the resulting protocols are as secure as if they were executed by some trusted party. Similarly, the universal verifiability of the resulting protocols can be deduced from that of CSZ.

Contents

5.1 The basic primitives of the MPC toolbox 109
5.1.1 Logical operations on encrypted data 109
5.1.2 Application to elementary arithmetic 112
5.1.3 Comparisons and tie breaking 112
5.2 Advanced algorithms 114
5.2.1 Multiplication and division 114
5.2.2 Solving ordering related problems 115
5.2.3 Aggregation of several encrypted binary values 118
5.2.4 Different communication/computation trade-offs 119
5.3 Comparison with other approaches 122
5.3.1 Comparison with Ordinos 123
5.3.2 Public tally hiding 124

5.1 The basic primitives of the MPC toolbox

5.1.1 Logical operations on encrypted data

Thanks to the conditional gate protocol and the homomorphic property of the exponential ElGamal encryption scheme, it is possible to derive a protocol for the most common logical operations. For convenience, we drop the other inputs (such as $(g, h, (h_i)_{i=1}^{n_T})$ and use the notation $\mathrm{CSZ}(X, Y)$

to denote the output of the conditional gate protocol, when applied to the ciphertexts X and Y. By abuse of notation, we consider that this output is always a well-formed ciphertext Z and not $\perp$ (*i.e.* that the protocol does not abort), so that $\mathrm{CSZ}(X, Y)$ can be used as an input in another protocol. The same goes for the other protocols that we build upon the conditional gate.

Basic boolean operations. Recall that the logical negation can be evaluated "for free" thanks to the homomorphic property: $\mathrm{Not}(B) = E_1/B$. In addition, remark that the CSZ protocol readily allows to compute the And algorithm, which is a specific case where X is also supposed to be an encryption of $x \in \{0, 1\}$. Thanks to the homomorphic property, it is easy to derive a protocol to evaluate the logical or and the logical xor.

Algorithm 49: Xor

Requires: X, Y, encryptions of $x, y \in \{0, 1\}$
Outputs: Z, an encryption of $x \oplus y$

1 **return** $XY/\mathrm{CSZ}(X, Y)^2$;

Algorithm 50: Or

Requires: X, Y, encryptions of $x, y \in \{0, 1\}$
Outputs: Z, an encryption of $x \vee y$

1 **return** $XY/\mathrm{CSZ}(X, Y)$;

Since the basic binary boolean operations (*i.e.* and, xor, or) are associative, it is possible to compute $\mathrm{And}(X_0, \cdots, X_{m-1})$ (resp. $\mathrm{Or}(X_0, \cdots, X_{m-1})$ and $\mathrm{Xor}(X_0, \cdots, X_{m-1})$) using a logarithmic number of synchronization steps, thanks to a boolean circuit that has a tree structure. See for instance Algorithm 51 for the case of the logical and.

Algorithm 51: And

Requires: $(X_1, \cdots, X_N)$, encryptions of $x_1, \cdots, x_N \in \{0, 1\}$
Outputs: Z, an encryption of $x_1 \wedge \cdots \wedge x_N$

1 $m \longleftarrow \lceil \log N \rceil$;
2 **for** $j = 0$ to $N - 1$ **do** $X_{1,j} \longleftarrow X_{j+1}$;
3 **for** $i = 1$ *to* m **do**
4 **for** $j = 0$ *to* $\lfloor N/2 \rfloor - 1$ *(in parallel)* **do**
5 $X_{i+1,j} \longleftarrow \mathrm{And}(X_{i,2j}, X_{i,2j+1})$;
6 **if** N is odd **then** $X_{i+1,\lfloor N/2 \rfloor} \longleftarrow X_{i,N-1}$;
7 $N \longleftarrow \lceil N/2 \rceil$;
8 **return** $X_{m+1,0}$;

Conditional branching. In addition to providing a way to realize the basic boolean operations, the conditionally set to zero functionality can also be used to evaluate a branching condition. In generic MPC, we want to avoid branching as much as possible since we do not want to reveal which branch is being evaluated: this could constitute a side-channel information. Therefore, the main strategy is to evaluate both branches and use a protocol to (obliviously) keep the relevant one. A classical solution is to use the ternary operator If, which takes as input a boolean b, two expressions x and y and returns either x when $b = 1$ or y when $b = 0$. This operator can be evaluated with a single call to CSZ; the same goes for the conditional swap. Note that in some cases, such as in Algorithm 51 (line 6), the branching condition depends on a public parameter, so that there is no need to hide which branch is computed.

Note that those operators can be used for many bits in parallel. For instance, assume that $\boldsymbol{X} = X_0, \cdots, X_{\ell-1}$, that $\boldsymbol{Y} = X_0, \cdots, Y_{\ell-1}$, and that B is an encryption of a bit $b \in \{0, 1\}$. Then we can define $\mathrm{If}(B, \boldsymbol{X}, \boldsymbol{Y})$ as $\mathrm{If}(B, X_0, Y_0), \cdots, \mathrm{If}(B, X_{\ell-1}, Y_{\ell-1})$. Similarly, if $(X'_i, Y'_i) = \mathrm{Swap}(B, X_i, Y_i)$ for all i, $\mathrm{Swap}(B, \boldsymbol{X}, \boldsymbol{Y})$ can also be defined as $(X'_0, \cdots, X'_{\ell-1}), (Y'_0, \cdots, Y'_{\ell-1})$.

Algorithm 52: Swap

Requires: B, a cipher of $b \in \{0,1\}$
X, Y, encryptions of x, y
Outputs: X', Y', s.t. X' (resp. Y') is a reenc. of Y (X) if $b = 1$, of X (resp. Y) otherwise

1 $Z \longleftarrow \mathrm{If}(B, Y, X)$;
2 **return** $Z, XY/Z$;

Algorithm 53: If

Requires: B, a cipher of $b \in \{0,1\}$
X, Y, encryptions of x, y
Outputs: Z, an encryption of x if $b = 1$, of y otherwise

1 **return** $Y\mathrm{CSZ}(X/Y, B)$;

Selection of an element in a list. The CSZ protocol can also be used to *select* an element inside a list. For this purpose, we suppose that $[X_i]$ (resp. $[\boldsymbol{X_i}]$) is a list of m ciphertexts (resp. of m lists of ℓ encryptions of bits) and that $[B_i]$ is a list of m encryptions of the bits b_i, such that one of them is 1 while the others are 0. Then we can recover a reencryption of X_i (resp. ℓ reencryptions of $\boldsymbol{X_i}$) where i is the index such that $b_i = 1$. By abuse of notation, we denote this procedure Select in both cases.

Algorithm 54: Select

Requires: $[(X_{i,0}, \cdots, X_{i,\ell-1})], [B_i]$
Outputs: $\boldsymbol{Z}$, a rencryption of $\boldsymbol{X_i}$ s.t. B_i is an encryption of 1

1 **for** all i, j **do** $A_{i,j} \longleftarrow \mathrm{CSZ}(X_{i,j}, B_i)$;
2 **for** all j **do** $Z_j \longleftarrow \prod_i A_{i,j}$;
3 **return** $(Z_0, \cdots, Z_{\ell-1})$

Algorithm 55: Select

Requires: $[X_i], [B_i]$
Outputs: Z, a rencryption of X_i s.t. B_i is an encryption of 1

1 **return** $\prod_i \mathrm{CSZ}(X_i, B_i)$;

Integer shift. Finally, consider an integer x and its binary representation $x_0, \cdots, x_{\ell-1}$, such that $x = \sum_{i=0}^{\ell-1} x_i 2^i$. A common operation is to *shift* the binary representation: the right shift corresponds to $0, x_0, \cdots, x_{\ell-2}$ and the left shift corresponds to $x_1, \cdots, x_{\ell-1}, 0$. In an MPC setting where x is encrypted bit-by-bit, we can perform the shift operations on the encrypted data for free, by using a trivial encryption E_0 of 0. We denoted the corresponding processes ShiftR and ShiftL. However, it may be useful to perform those operations *conditionally* to an encrypted boolean b. For this purpose, we can use the If protocol in parallel, which gives the conditional left shift and conditional right shift protocols.

Algorithm 56: CLS

Requires: $(V_0, \cdots, V_{\ell-1})$, ciphertexts
B, an encryption of 0 or 1
Outputs: $\boldsymbol{V'}$, a reencrypted left shift of $\boldsymbol{V}$ if $b = 1$, a reencryption of $\boldsymbol{V}$ otherwise.

1 $V_\ell \longleftarrow E_0$;
2 **for** $j = 0$ *to* $\ell - 1$ *(in parallel)* **do**
3 $\quad V'_j \longleftarrow \mathrm{If}(B, V_{j+1}, V_j)$;
4 Return $\boldsymbol{V'}$;

Algorithm 57: CRS

Requires: $(V_0, \cdots, V_{\ell-1})$, ciphertexts
B, an encryption of 0 or 1
Outputs: $\boldsymbol{V'}$, a reencrypted right shift of $\boldsymbol{V}$ if $b = 1$, a reencryption of $\boldsymbol{V}$ otherwise.

1 $V_{-1} \longleftarrow E_0$;
2 **for** $j = 0$ *to* $\ell - 1$ *(in parallel)* **do**
3 $\quad V'_j \longleftarrow \mathrm{If}(B, V_{j-1}, V_j)$;
4 Return $\boldsymbol{V'}$;

5.1.2 Application to elementary arithmetic

With the boolean operations, we can readily use the schoolbook boolean circuits to evaluate the elementary arithmetic operations. We give Algorithm 58, which comes from the schoolbook algorithm for the addition. At line 6, the new value of the carry bit is deduced by polynomial interpolation as a function of the other parameters. In the resulting formula, we recall that $1/2$ corresponds to $(q+1)/2$ modulo q. On this occasion, we remark that there is a possible optimization when one of the two operands is known, which allows to save half of the computations. We explicit the corresponding optimization in Algorithm 59, where E_{y_i} denotes a trivial encryption of y_i. In addition, note that the Add protocol returns a ℓ-bits results when given two $(\ell-1)$-bits inputs. Sometimes, it may be preferable to keep operands of the same size and drop the last carry bit, in which case the operation would be performed modulo 2^ℓ. In this case, we use the notation $\text{Add}^{[\ell]}$.

Algorithm 58: Add

Requires: $(X_0, \cdots, X_{\ell-1}), (Y_0, \cdots, Y_{\ell-1})$, bit-wise encryptions of x, y
Outputs: $(Z_0, \cdots, Z_\ell)$, bit-wise encryption of $x+y$

1 $R \longleftarrow \text{And}(X_0, Y_0)$ (* carry bit *);
2 $Z_0 \longleftarrow X_0 Y_0 / R^2$ (* $x_0 \oplus y_0$ *);
3 **for** $i = 1$ *to* $\ell - 1$ **do**
4 $\quad A \longleftarrow \text{Xor}(X_i, Y_i)$;
5 $\quad Z_i \longleftarrow \text{Xor}(A, R)$;
6 $\quad R \longleftarrow (X_i Y_i R / Z_i)^{\frac{1}{2}}$;
7 **return** $Z_0, \cdots, Z_{\ell-1}, R$

Algorithm 59: AddKnown

Requires: $(X_0, \cdots, X_{\ell-1})$, bit-wise encryption of x
$(y_0, \cdots, y_{\ell-1})$, the bits of y
Outputs: $(Z_0, \cdots, Z_\ell)$, bit-wise encryption of $x+y$

1 $R \longleftarrow X_0^{y_0}$ (* carry bit *);
2 $Z_0 \longleftarrow X_0 E_{y_0} / R^2$ (* $x_0 \oplus y_0$ *);
3 **for** $i = 1$ *to* $\ell - 1$ **do**
4 $\quad A \longleftarrow X_i^{y_i}$;
5 $\quad Z_i \longleftarrow \text{Xor}(A, R)$;
6 $\quad R \longleftarrow (X_i E_{y_i} R / Z_i)^{\frac{1}{2}}$;
7 **return** $Z_0, \cdots, Z_{\ell-1}, R$

The subtraction is a bit more tricky, as it may result in a negative value. To circumvent this, it is usual to perform all the computations modulo 2^ℓ. This gives Algorithm 60, in which the new value of the borrow bit is deduced by evaluating the boolean formula $(y_i \wedge r) \vee [(y_i \vee r) \wedge \neg x_i]$ at line 9.

As we use 2's complement, computing a representation of $-x$ from that of x is not as simple as flipping a single bit sign. For this reason, we give Algorithm 62 which is another adaptation from the schoolbook.

5.1.3 Comparisons and tie breaking

Apart from the additions and the subtractions, the most common operations in electronic voting are the equality tests and the comparisons. For the latter, we remark that we can already derive a comparison test from the subtraction algorithm. Indeed, $x < y$ if and only if $x - y < 0$ so that we can use Algorithm 60 and return the last bit. We denote this protocol Lt. In Section 5.2.4, we will propose another protocol that offers another communication/computation trade-off. For the equality test, one strategy is to first compute the bitwise XOR of x and y and to check that the resulting bits are all 0. For this purpose, one can use Algorithm 51.

Since the Eq and Lt protocols shares some CSZ in common, it is possible to same some additional computations if we need to evaluate the result of both operations. We denote the

Algorithm 60: Sub

Requires: $(X_0, \cdots, X_{\ell-1}), (Y_0, \cdots, Y_{\ell-1})$, bit-wise encryptions of x, y
Outputs: $(Z_0, \cdots, Z_{\ell-1})$, bit-wise enc. of $x - y$ modulo 2^ℓ
R, an enc. of the bit sign (1 if the result is negative)

$A \longleftarrow \text{And}(X_0, Y_0)$ (* carry bit *);
$Z_0 \longleftarrow X_0 Y_0 / A^2$ (* $x_0 \oplus y_0$ *);
$R \longleftarrow Y_0 / A$ (* $y_0 \wedge \neg x_0$ *);
for $i = 1$ *to* $\ell - 1$ **do**
  $A \longleftarrow \text{And}(Y_i, R)$;
  $B \longleftarrow Y_i R / A^2$ (* $y_i \oplus r$ *);
  $C \longleftarrow \text{And}(X_i, B)$;
  $Z_i \longleftarrow X_i B / C^2$ (* $x_i \oplus y_i \oplus r$ *);
  $R \longleftarrow Y_i R / (AC)$;
return $(Z_0, \cdots, Z_{\ell-1}), R$

Algorithm 61: SubKnown

Requires: $(X_0, \cdots, X_{\ell-1})$, bit-wise encryption of x
$(y_0, \cdots, y_{\ell-1})$, the bits of y
Outputs: $(Z_0, \cdots, Z_{\ell-1})$, bit-wise enc. of $x - y$ modulo 2^ℓ
R, an enc. of the bit sign (1 if the result is negative)

$A \longleftarrow X_0^{y_0}$ (* carry bit *);
$Z_0 \longleftarrow X_0 E_{y_0} / A^2$ (* $x_0 \oplus y_0$ *);
$R \longleftarrow E_{y_0} / A$ (* $y_0 \wedge \neg x_0$ *);
for $i = 1$ *to* $\ell - 1$ **do**
  $A \longleftarrow R^{y_i}$;
  $B \longleftarrow E_{y_i} R / A^2$ (* $y_i \oplus r$ *);
  $C \longleftarrow \text{And}(X_i, B)$;
  $Z_i \longleftarrow X_i B / C^2$ (* $x_i \oplus y_i \oplus r$ *);
  $R \longleftarrow E_{y_i} R / (AC)$;
return $(Z_0, \cdots, Z_{\ell-1}), R$

Algorithm 62: Neg

Requires: $(X_0, \cdots, X_{\ell-1})$, a bit-wise encryption of x modulo 2^ℓ
Outputs: $(Z_0, \cdots, Z_{\ell-1})$, a bit-wise encryption of $-x$ modulo 2^ℓ

$Z_0 \longleftarrow X_0$; $R_0 \longleftarrow \text{Not}(X_0)$;
for $i = 1$ *to* $\ell - 1$ **do**
  $R_i \longleftarrow \text{And}(\text{Not}(X_i), R_{i-1})$;
  $Z_i \longleftarrow \text{Not}(X_i) R_{i-1} / R_i^2$;
return $Z_0, \cdots, Z_{\ell-1}$;

Algorithm 63: Eq

Requires: $(X_0, \cdots, X_{\ell-1}), (Y_0, \cdots, Y_{\ell-1})$, bit-wise encryptions of x, y
Outputs: Z, an encryption of 1 if $x = y$, of 0 otherwise

for $i = 0$ *to* $\ell - 1$ *(in parallel)* **do**
  $A_i \longleftarrow \text{Not}(\text{And}(X_i, Y_i))$;
return $\text{And}(A_0, \cdots, A_{\ell-1})$;

Algorithm 64: EqKnown

Requires: $(X_0, \cdots, X_{\ell-1})$, bit-wise encryption of x
$(y_0, \cdots, y_{\ell-1})$, the bits of y
Outputs: Z, an encryption of 1 if $x = y$, of 0 otherwise

for $i = 0$ *to* $\ell - 1$ *(in parallel)* **do**
  $A_i \longleftarrow \text{Not}(X_i^{y_i})$;
return $\text{And}(A_0, \cdots, A_{\ell-1})$;

resulting protocols LtEq and LtEqKnown.

Tie breaking. In the context of electronic voting, the integers $x_1, \cdots, x_k$ that we are going to compare would typically represent the "score" of a candidate at a specific moment in the tally process. Now, suppose that we want to apply a specific rule for tie-breaking (*e.g.*, the oldest candidate wins in case of a tie). Enforcing this using Lt, Eq and If is definitely possible, but may result into unnecessary redundant operations. Instead, we propose to directly "encode" this rule into the least significant bits of $x_1, \cdots, x_k$. More precisely, suppose that we have the candidates $C_1, \cdots, C_k$, for which a public rule for tie-breaking has been decided. Without a loss of generality, we represent this rule as a permutation σ of $[1, k]$ such that C_i wins over C_j (in case of an equality) if $\sigma_i > \sigma_j$. Then assume that we obtained the bitwise encryptions $\boldsymbol{X_1}, \cdots, \boldsymbol{X_k}$ of the scores of the candidates. Then, by adding a bit-wise encryption of $\sigma_1, \cdots, \sigma_k$ in the least significant bits, we end up with pair-wise distinct scores that are compliant with both the tie-break rule and the initial scores (*i.e.* this does not change the result, except if a tie occurs). More precisely, if $\sigma_i = \sum_{j=0}^{\log k} s_{i,j} 2^j$ for all j, then we use $E_{s_{i,0}} || \cdots || E_{s_{i,\log k}} || \boldsymbol{X_i}$ instead of $\boldsymbol{X_i}$. Since the Lt protocol requires 2CSZ per bit, this only costs $2 \log k$ additional CSZ per comparison.

Interestingly, the same strategy could be applied if we want the tie break rule to be random and secret: indeed, the prefixes can be shuffled using a reencryption mixnet.

5.2 Advanced algorithms

Although the most common operations in voting are the additions and the comparisons, it is possible that evaluating a counting function requires more complex operations. For this reason, we give more advanced algorithms, which include a way to obtain the encrypted data from the encrypted ballots.

5.2.1 Multiplication and division

Basic arithmetic operations include multiplication and division. For the multiplication, we give Algorithm 65 which is adapted from the schoolbook binary algorithm, also known as "peasant multiplication". This shows that multiplying two encrypted integers is expensive in the ElGamal setting, as it requires a quadratic number of conditional gates.

Algorithm 65: Mult

Requires: $(X_0, \cdots, X_{\ell_x - 1}), (Y_0, \cdots, Y_{\ell_y - 1})$, bitwise encryptions of x and y
Outputs: $Z_0, \cdots, Z_{\ell_x + \ell_y - 1}$, bitwise encryption of xy

for $i \in [0, \ell_x - 1]$ (in parallel) **do** $A_{i,0}, \cdots, A_{i,\ell_y - 1} \longleftarrow \mathrm{CSZ}(\boldsymbol{Y}, X_i)$;
$Z_0 \longleftarrow A_{0,0}$;
$(T_0, \cdots, T_{\ell_y - 1}) \longleftarrow (A_{0,1}, \cdots, A_{0,\ell_y - 1}, E_0)$;
for $i = 1$ *to* $\ell_x - 1$ **do**
    $(T_0, \cdots, T_{\ell_y}) \longleftarrow \mathrm{Add}((T_0, \cdots, T_{\ell_y - 1}), (A_{i,0}, \cdots, A_{i,\ell_y - 1}))$;
    $Z_i \longleftarrow T_0$;
    for $j = 0$ to $\ell_y - 1$ **do** $T_j \longleftarrow T_{j+1}$;
for $i = \ell_x$ to $\ell_x + \ell_y - 1$ **do** $Z_i \longleftarrow T_{i - \ell_x}$;
return $Z_0, \cdots, Z_{\ell_x + \ell_y - 1}$;

For the division, we choose to represent fractions with a fixed number of binary places so that a fraction is encoded and encrypted as an integer (instead of, for instance, a couple of integers). This allows to re-use most of the primitives from this section, while providing a certain degree of precision and generality. From the schoolbook division algorithm, we derive Algorithm 66, which takes as inputs bit-wise encryptions of x and y with $y > x$ and return the r first binary places of x/y. In other words, if we interpret the output z of the division as an integer, we have $|\frac{x}{y} - \frac{z}{2^r}| < 2^{-r}$. This algorithm could be generalized for any pair (x, y) (*i.e.* the condition $y > x$ is not necessary), but the restriction is useful in the special case of STV, and gives a simpler description. Note that in Algorithm 66, we choose to stick to the convention where the least significant bit is given first, which means that the output binary places are output in "reverse" order compared to the usual. This way, it is possible to add two fractions together without having to reverse their binary representation.

Algorithm 66: Div

Requires: $(X_0, \cdots, X_{\ell-1}), (Y_0, \cdots, Y_{\ell-1})$, bit-wise encryptions of $0 \leq x < y$, r, the number of bits of precision

Outputs: $Z_0, \cdots, Z_{r-1}$, encryptions of the first r binary places of x/y (z_0 is the least significant bit)

$\boldsymbol{A} \longleftarrow E_0 || \boldsymbol{X}$ (* $2x$ *);
$\boldsymbol{Y} \longleftarrow \boldsymbol{Y} || E_0$ (* y, padded to have the same size *);
for $i = 0$ *to* $r - 1$ **do**
    $\boldsymbol{B}, R_i = \mathrm{Sub}(\boldsymbol{A}, \boldsymbol{Y})$;
    $\boldsymbol{A} \longleftarrow \mathrm{ShiftR}(\mathrm{If}(R_i, \boldsymbol{A}, \boldsymbol{B}))$ (* this *right* shift is non-standard: we use a representation with the LSB first; it corresponds to a multiplication by 2 *);
    $Z_{r-1-i} \longleftarrow \mathrm{Not}(R_i)$
return $Z_0, \cdots, Z_{r-1}$

5.2.2 Solving ordering related problems

Voting consists of finding the "most preferred" option. Consequently, it is common to encounter an algorithmic problem related to ordering.

Maximum and minimum. The most obvious problem is to find the largest or the smallest element of a list. A natural solution would be to linearly scan the list, using a comparison algorithm. However, the min and max operators are associative and as such, allow tree-based parallelization as we did in Algorithm 51. This gives Algorithm 67, which finds the maximum, the minimum and their respective position, using a logarithmic number of rounds of communications. In this algorithm, we denote j^{bits} the trivial bitwise encryption of the integer j, with a fixed number of bits. We denote Min (resp. Max) the protocol that only returns a bitwise encryption of the minimum (resp. the maximum) as well as its position in the list.

Finding the s largest elements. A related problem is to find the s largest values of a list. For this purpose, we propose two different approaches: the selection approach and the insertion approach, base on insertion sort and selection sort. The insertion approach consists of first sorting the s first elements of the list so that we have the list of the s largest elements of the s first elements of the list. Then, we iteratively update this small list by inserting the remaining elements of the large list, so that at the kth iteration, the small list consists of the s largest elements of the $s + k$ first elements of the list. This approach imitates what the selection

Algorithm 67: MinMax

Requires: $(\boldsymbol{X_1}, \cdots, \boldsymbol{X_N})$ bitwise encryptions of $x_1, \cdots, x_N$
ℓ, the common bitsize of the x_i's
Outputs: $\boldsymbol{Z}$, a bitwise encryption of $\min_{i=1}^{N}(x_i)$
$\boldsymbol{I}$, a bitwise encryption of its index in the input list
$\boldsymbol{T}$, a bitwise encryption of $\max_{i=1}^{N}(x_i)$
$\boldsymbol{J}$, a bitwise encryption of its index in the input list

$m \longleftarrow \lceil \log N \rceil$;
for $j = 0$ *to* $N - 1$ **do**
  $\boldsymbol{Z_{1,j}} \longleftarrow \boldsymbol{X_{j+1}}$;
  $\boldsymbol{I_{1,j}} \longleftarrow j + 1^{\text{bits}}$;
  $\boldsymbol{T_{1,j}} \longleftarrow \boldsymbol{X_{j+1}}$;
  $\boldsymbol{J_{1,j}} \longleftarrow j + 1^{\text{bits}}$;
for $i = 1$ *to* m **do**
  for $j = 0$ *to* $\lfloor N/2 \rfloor - 1$ *(in parallel)* **do**
    (* The two following operations can be done in parallel *)
    $B_Z \longleftarrow \text{Lt}(\boldsymbol{Z_{i,2j}}, \boldsymbol{Z_{i,2j+1}})$;
    $B_T \longleftarrow \text{Lt}(\boldsymbol{T_{i,2j}}, \boldsymbol{T_{i,2j+1}})$;
    (* The four following operations can be done in parallel *)
    $\boldsymbol{Z_{i+1,j}} \longleftarrow \text{If}(B_Z, \boldsymbol{Z_{i,2j}}, \boldsymbol{Z_{i,2j+1}})$;
    $\boldsymbol{I_{i+1,j}} \longleftarrow \text{If}(B_Z, \boldsymbol{I_{i,2j}}, \boldsymbol{I_{i,2j+1}})$;
    $\boldsymbol{T_{i+1,j}} \longleftarrow \text{If}(B_T, \boldsymbol{T_{i,2j+1}}, \boldsymbol{Z_{i,2j}})$;
    $\boldsymbol{J_{i+1,j}} \longleftarrow \text{If}(B_T, \boldsymbol{J_{i,2j+1}}, \boldsymbol{J_{i,2j}})$;
  if N *is odd* **then**
    $\boldsymbol{Z_{i+1,\lfloor N/2 \rfloor}} \longleftarrow \boldsymbol{Z_{i,N-1}}$;
    $\boldsymbol{I_{i+1,\lfloor N/2 \rfloor}} \longleftarrow \boldsymbol{I_{i,N-1}}$;
    $\boldsymbol{T_{i+1,\lfloor N/2 \rfloor}} \longleftarrow \boldsymbol{T_{i,N-1}}$;
    $\boldsymbol{J_{i+1,\lfloor N/2 \rfloor}} \longleftarrow \boldsymbol{J_{i,N-1}}$;
  $N \longleftarrow \lceil N/2 \rceil$;
return $\boldsymbol{Z_{m+1,0}}, \boldsymbol{I_{m+1,0}}, \boldsymbol{T_{m+1,0}}, \boldsymbol{J_{m+1,0}}$;

Algorithm 68: sInsert

Requires: $\boldsymbol{X_0}, \cdots, \boldsymbol{X_{N-1}}$, bitwise encryptions of $x_0, \cdots, x_{N-1}$
s, a positive integer

Outputs: $\boldsymbol{Z_1}, \cdots, \boldsymbol{Z_s}$, bitwise enc. of the s largest values
$\boldsymbol{I_1}, \cdots, \boldsymbol{I_s}$, bitwise encryptions of their indexes

for $i = 1$ *to* s **do**
  $\boldsymbol{Z_i} \longleftarrow \boldsymbol{X_{i-1}}$;
  $\boldsymbol{I_i} \longleftarrow i - 1^{\text{bits}}$;
  for $j = i - 1$ *down to* 1 **do**
    $B \longleftarrow \text{Lt}(\boldsymbol{Z_j}, \boldsymbol{Z_{j+1}})$;
    $\boldsymbol{Z_j}, \boldsymbol{Z_{j+1}} \longleftarrow \text{Swap}(B, \boldsymbol{Z_j}, \boldsymbol{Z_{j+1}})$;
    $\boldsymbol{I_j}, \boldsymbol{I_{j+1}} \longleftarrow \text{Swap}(B, \boldsymbol{I_j}, \boldsymbol{I_{j+1}})$;
for $i = s + 1$ *to* N **do**
  $B \longleftarrow \text{Lt}(\boldsymbol{Z_s}, \boldsymbol{X_{i-1}})$;
  $\boldsymbol{Z_s} \longleftarrow \text{If}(B, \boldsymbol{X_{i-1}}, \boldsymbol{Z_s})$;
  $\boldsymbol{I_s} \longleftarrow \text{If}(B, \boldsymbol{i - 1^{\text{bits}}}, \boldsymbol{I_s})$;
  for $j = s - 1$ *down to* 1 **do**
    $B \longleftarrow \text{Lt}(\boldsymbol{Z_j}, \boldsymbol{Z_{j+1}})$;
    $\boldsymbol{Z_j}, \boldsymbol{Z_{j+1}} \longleftarrow \text{Swap}(B, \boldsymbol{Z_j}, \boldsymbol{Z_{j+1}})$;
    $\boldsymbol{I_j}, \boldsymbol{I_{j+1}} \longleftarrow \text{Swap}(B, \boldsymbol{I_j}, \boldsymbol{I_{j+1}})$;
return $(\boldsymbol{Z_1}, \cdots, \boldsymbol{Z_s})$, $(\boldsymbol{I_1}, \cdots, \boldsymbol{I_s})$;

Algorithm 69: sSelect

Requires: $\boldsymbol{X_0}, \cdots, \boldsymbol{X_{N-1}}$, bitwise encryptions of $x_0, \cdots, x_{N-1}$
s, a positive integer

Outputs: $\boldsymbol{Z_1}, \cdots, \boldsymbol{Z_s}$, bitwise enc. of the s largest values
$\boldsymbol{I_1}, \cdots, \boldsymbol{I_s}$, bitwise encryptions of their indexes

for $i = 1$ *to* s **do**
  $\boldsymbol{Z_i}, \boldsymbol{I_i} \longleftarrow \text{Max}(\boldsymbol{X_0}, \cdots, \boldsymbol{X_{N-1}})$;
  for $j = 0$ *to* $N - 1$ *(in parallel)* **do**
    Writes j in base 2:
    $j = \sum_{k=0}^{\ell} m_k 2^k$;
    for $k = 0$ *to* ℓ **do**
      $J_k \longleftarrow E_{1-m_k} I_{i,k}^{2m_k - 1}$;
      (* $\text{Eq}(I_{i,k}, m_k)$ *)
    $B \longleftarrow \text{And}(J_0, \cdots, J_\ell)$;
    $\boldsymbol{X_j} \longleftarrow \text{CSZ}(\boldsymbol{X_j}, B)$;
return $(\boldsymbol{Z_1}, \cdots, \boldsymbol{Z_s})$, $(\boldsymbol{I_1}, \cdots, \boldsymbol{I_s})$;

sort would do, but avoids the quadratic cost by maintaining a small list of size s. However, the drawback is that it is expensive communication-wise, since the process is mostly iterative. For this reason, we propose another approach, based on selection sort. It consists of using the Max protocol to get the maximum value in a logarithmic number of rounds, as well as its respective index in the list. Then, using the index, the equality test and the CSZ protocol, we can "remove" this maximum from the list (actually, we replace it by a 0 value) without leaking its position. This way, we can iteratively get the s largest elements, using only s iterations.

Sorting. Finally, another recurrent problem is to sort a list. Using Lt and Swap, it is possible to sort encrypted data without revealing any side information. For this purpose, we need a *data-oblivious* sorting algorithm, that is an algorithm whose control flow does not depend on the result of the comparisons. The popular fast sorting algorithms, such as Quicksort, Mergesort or Heapsort, do not verify this property. Consequently, we use the OddEvenMergeSort by Batcher [Bat68], which has a quasi-linear complexity and is used in practice for sorting networks in GPU. This gives Algorithm 70, adapted from [Knu73, Section 5.2.2, Algorithm M]). This sorting algorithm requires approximately $\frac{1}{4} N \log(N)^2$ comparisons and conditional swaps, in approximately $\frac{1}{2} \log(N)^2$ rounds of communications, where N is the number of elements to be sorted. Remark that in Algorithm 70, we consider that we want to sort some values (for

instance, the index of the candidate) with respect to a corresponding key. It is possible to adapt this algorithm for a setting where we just want to sort bitwise encrypted integers, which are not linked to a specific value, or to have the values be bitwise encrypted.

Another usual solution for sorting in an MPC setting is to first shuffle the data, then use a more efficient algorithm such as Mergesort, but which requires to leak the result of all the comparisons. This usually leads to a better computational efficiency, but a far worse communication efficiency (typically, Mergesort would require a linear number of synchronization steps compared to the number of elements to sort). In addition, the security of the resulting protocol would not be guaranteed by the SUC framework since there is currently no known SUC-secure reencryption mixnet.

Algorithm 70: OddEvenMergeSort

Requires: $(V_i, \boldsymbol{K_i})_{i=0}^{N-1}$, where, for all i, V_i is a ciphertext and $\boldsymbol{K_i}$ is a bitwise encryption of an integer k_i

Outputs: $(V'_i, \boldsymbol{K'_i})_{i=0}^{N-1}$, reencryptions of the same values, but sorted with increasing k_i

$t \longleftarrow \lceil \log N \rceil$; $p \longleftarrow 2^{t-1}$
while $p > 0$ **do**
  $q \longleftarrow 2^{t-1}$; $r \longleftarrow 0$; $d \longleftarrow p$
  while $d > 0$ **do**
    for $i = 0$ *to* $n - d - 1$ *(in parallel)* **do**
      if $\mathsf{BitwiseAnd}(i, p) = r$ **then**
        $B \longleftarrow \mathrm{Lt}(\boldsymbol{K_{i+d}}, \boldsymbol{K_i})$
        $V_i, V_{i+d} \longleftarrow \mathrm{Swap}(B, V_i, V_{i+d})$
        $\boldsymbol{K_i}, \boldsymbol{K_{i+d}} \longleftarrow \mathrm{Swap}(B, \boldsymbol{K_i}, \boldsymbol{K_{i+d}})$
    $d \longleftarrow q - p$; $q \longleftarrow \lfloor q/2 \rfloor$; $r \longleftarrow p$
  $p \longleftarrow \lfloor p/2 \rfloor$
return $(V_i, \boldsymbol{K_i})_{i=0}^{n-1}$

5.2.3 Aggregation of several encrypted binary values

In electronic voting, it is usual that the voter encrypts their ballots using encryptions of 0 or 1, as this is the case in Helios. Afterwards, we need to aggregate those encrypted bits into bitwise encrypted integers, so that we can use the various protocols of this toolbox. Since the addition is associative, we use again a tree-based strategy that allows to compute several additions in parallel. By adding together integers of (almost) the same size and by performing as many additions in parallel as possible, we gain in efficiency both communication-wise and computation-wise. The algorithm is similar to the previous tree-based protocols, but we have to take care of the size of the operands. For this reason, we introduce the notation $\mathrm{Add}_{x,y}$ which denotes a protocol, derived from Add, which is able to add two operands of different (known) sizes $s_x = \lfloor \log x \rfloor + 1$ and $s_y = \lfloor \log y \rfloor + 1$. The size of the output is $\lfloor \log(x + y) \rfloor + 1$, the number of communication rounds is $\max(s_x, s_y)$ and the number of CSZ is $s_x + s_y - 1$. This gives Algorithm 71 where, at the ith iteration, we add integers of size at most i, which costs at most $2i$CSZ communication-wise, and up to $2iN/2^i$CSZ computation-wise. Hence, the number of rounds of communication

is about $\frac{1}{2}\log(N)^2$ while the computational cost is at most

$$\sum_{i=1}^{m} \frac{2i-1}{2^i} N\text{CSZ} \leq \sum_{i=1}^{\infty} \frac{2i-1}{2^i} N\text{CSZ} \leq 3N\text{CSZ}.$$

In some cases, the quadratic number of synchronization steps may be a problem. For this reason, we can use the UFCAdd (introduced in Section 5.2.4) instead of the regular addition. This leads to a number of rounds of communication of at most $2\log N \log\log N$, and a computational cost of at most

$$\sum_{i=1}^{m} \frac{3i\log(i+1)}{2^{i+1}} N\text{CSZ} \leq \sum_{i=1}^{\infty} \frac{3i\log(i+1)}{2^{i+1}} N\text{CSZ} \leq 5.54N\text{CSZ}.$$

Algorithm 71: Aggreg

Requires: $B_1, \cdots, B_N$, encryptions of $b_1, \cdots, b_N \in \{0,1\}$
Outputs: $S_0, \cdots, S_{m-1}$, a bitwise encryption of $s = \sum_{i=1}^{N} b_i$

$m \longleftarrow \lceil \log N \rceil$;
for $j = 0$ *to* $N-1$ **do**
  $\boldsymbol{B_{1,j}} \longleftarrow B_{j+1}$;
  $c_{1,j} \longleftarrow 1$;
for $i = 1$ *to* m **do**
  for $j = 0$ *to* $\lfloor N/2 \rfloor - 1$ *(in parallel)* **do**
    $\boldsymbol{B_{i+1,j}} \longleftarrow \text{Add}_{c_{i,2j},c_{i,2j+1}}(\boldsymbol{B_{i,2j}}, \boldsymbol{B_{i,2j+1}})$;
    $c_{i+1,j} \longleftarrow c_{i,2j} + c_{i,2j+1}$;
  if N *is odd* **then**
    for $j = \lfloor N/2 \rfloor$ *down to* 1 **do**
      $\boldsymbol{B_{i+1,j}} \longleftarrow \boldsymbol{B_{i+1,j-1}}$;
      $c_{i+1,j} \longleftarrow c_{i+1,j-1}$;
    $\boldsymbol{B_{i+1,0}} \longleftarrow \boldsymbol{B_{i,N-1}}$;
    $c_{i+1,0} \longleftarrow c_{i,N-1}$;
  $N \longleftarrow \lceil N/2 \rceil$;
return $\boldsymbol{B_{m+1,0}}$;

5.2.4 Different communication/computation trade-offs

Compared to the existing MPC toolbox that use the Paillier encryption scheme, our toolbox proposes several protocols which require n_T synchronization steps (*i.e.* a round of communication) per encrypted bit. Yet, it is important to keep the number of synchronization steps down. For this reason, we propose to use more sophisticated boolean circuits, following the (now classical) approach of Brent and Kung [BK82]. We do not reproduce their full algorithm here but we sketch the key idea and give the resulting algorithms and their complexity.

First, recall that the ith bit of $x+y$ is $z_i = x_i \oplus y_i \oplus r_i$, where r_i is the ith carry bit. The idea is to first compute all the $x_i \oplus y_i$ in parallel, then to compute all the r_i in parallel, so as to deduce the result. To perform the second step efficiently, Brent and Kung's approach consists

of computing the variables (p_i, g_i) where $p_i = x_i \vee y_i$ and $g_i = x_i \wedge y_i$. Those variables are used to encode elements of a set $\Sigma = \{P, G, K\}$, where P is encoded by $(1,0)$, K by $(0,0)$ and G by $(0,1)$ and $(1,1)$. They represent the fact that the carry bit will be propagated, generated of killed in the ith position. They define an operation $\circ$ as follows (which we slightly modify into an equivalent operation for the sake of presentation).

$$\begin{aligned} P \circ P &= P \\ G \circ P &= G \\ K \circ P &= K \\ x \circ G &= G \\ x \circ K &= K. \end{aligned}$$

In the boolean representation, the $\circ$ law can be computed with the following formula.

$$(p, g) \circ (p', g') = (p \wedge p', g' \vee (p' \wedge g)).$$

It is possible to show that $\circ$ is associative [BK82]; this enables tree-based parallelization for computing all the prefixes of $(p_0, g_0) \circ \cdots \circ (p_{m-1}, g_{m-1})$, which gives essentially the ith carry bit for all i. From here onward, we diverge from [BK82]'s work since we are not interested in designing hardware, so that the unbounded fan-in is not an issue. From Brent and Kunt's circuit design, that we reproduce in Fig. 13, we deduce the UFC algorithm (see Algorithm 72), which stands for unbounded fan-in composition. It allows to compute all the prefixes of a sequence, when composed with an associative composition law. Then, we derive another protocol for the addition, given in Algorithm 73. The resulting protocol is more efficient in term of communications than the Add protocol: it only requires about $\log(m)$ times more synchronization steps than what is required for $\circ$. However, this comes with an increase in term of computations as the number of calls to $\circ$ is about $\frac{1}{2}\ell \log(\ell)$. Therefore, the linear approach could be preferable in some cases.

Algorithm 72: UFC

Requires: $\circ$, an associative composition law
$a_0, \cdots, a_N$, some elements compatible with this law
Outputs: $z_0, \cdots, z_N$, where $z_i = \circ_{j=0}^{i} a_j$ for all i

1 **for** $i = 0$ to $N - 1$ **do** $z_i \longleftarrow a_i$ $m \longleftarrow \lceil \log N \rceil$;
2 **for** $i = 1$ *to* m **do**
3 **for** $j = 0$ *to* $\lfloor ((N-1)/2^{i-1} - 1)/2 \rfloor$ *(in parallel)* **do**
4 idx $\longleftarrow (2j+1)2^{i-1} - 1$;
5 stop $\longleftarrow \min(2^{i-1}, N - (2j+1)2^{i-1})$;
6 **for** $k = 1$ *to* stop *(in parallel)* **do**
7 idy $\longleftarrow$ idx $+ k$;
8 $z_{\text{idy}} \longleftarrow \circ(a_{\text{idx}}, a_{\text{idy}})$;
9 **return** $z_0, \cdots, z_{N-1}$;

The same idea can be used for computing a subtraction: we only need to change the initialization of the p_i's and g_i's. For the subtraction, we have initially $p_i = x_i \oplus y_i$ and $g_i = y_i \wedge \neg x_i$. Hence, to obtain UFCSub, one can replace line 4 by $P_i \longleftarrow B_i$ and line 5 by $G_i \longleftarrow Y_i / A_i$ in Algorithm 73.

Algorithm 73: UFCAdd

Requires: $(X_0, \cdots, X_{\ell-1}), (Y_0, \cdots, Y_{\ell-1})$, bit-wise encryptions of x and y.
Outputs: $(Z_0, \cdots, Z_{\ell-1})$, bit-wise encryption of $x + y \bmod 2^\ell$

for $i = 0$ *to* $\ell - 1$ *(* in parallel *)* **do**
  $A_i \longleftarrow \mathrm{And}(X_i, Y_i)$;
  $B_i \longleftarrow X_i Y_i / A_i^2$ (* $x_i \oplus y_i$ *);
  $P_i \longleftarrow X_i Y_i / A_i$ (* $x_i \vee y_i$ *);
  $G_i \longleftarrow A_i$ (* $x_i \wedge y_i$ *);
$m \longleftarrow \lceil \log \ell \rceil$;
for $i = 1$ *to* m **do**
  for $j = 0$ *to* $\lfloor ((\ell - 1)/2^{i-1} - 1)/2 \rfloor$ *(in parallel)* **do**
    $\mathrm{idx} \longleftarrow (2j + 1)2^{i-1} - 1$;
    $\mathrm{stop} \longleftarrow \min(2^{i-1}, \ell - (2j + 1)2^{i-1})$;
    for $k = 1$ *to* stop *(in parallel)* **do**
      $\mathrm{idy} \longleftarrow \mathrm{idx} + k$;
      $T \longleftarrow \mathrm{And}(P_{\mathrm{idy}}, G_{\mathrm{idx}})$;
      $P_{\mathrm{idy}} \longleftarrow \mathrm{And}(P_{\mathrm{idx}}, P_{\mathrm{idy}})$;
      $G_{\mathrm{idy}} \longleftarrow \mathrm{Or}(T, G_{\mathrm{idy}}))$;
$Z_0 \longleftarrow B_0$;
for $i = 1$ *to* $\ell - 1$ *(in parallel)* **do**
  $Z_i \longleftarrow \mathrm{Xor}(B_i, G_{i-1})$;
return $Z_0, \cdots, Z_{m-1}$

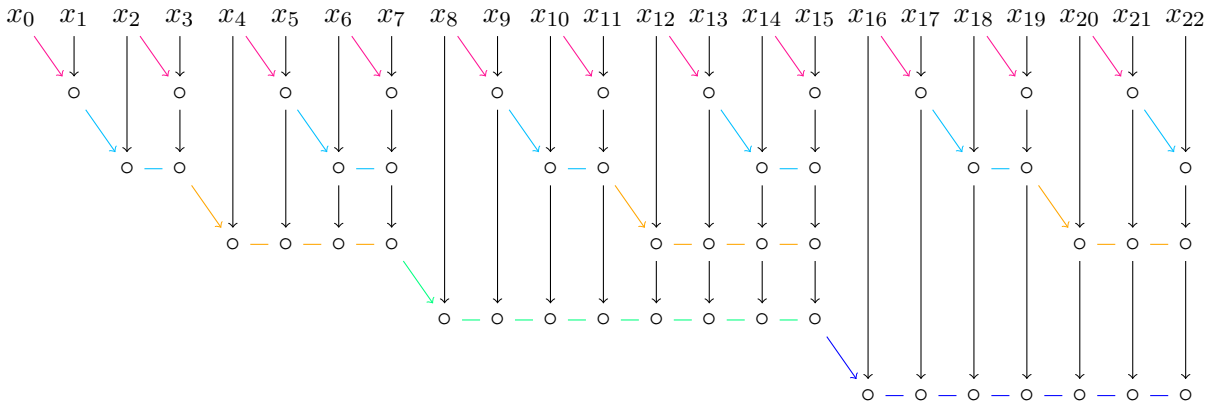


Figure 13: Illustration of Brent and Kunt's algorithm, for 23 operands

When it comes to comparing two integers, only the last carry bit is of interest so we do not need to compute all the prefixes. In this case, a simpler algorithm exists and allows to compute the comparison with $\ell - 1$ calls to $\circ$ but a communication cost which remains of the order of $\log(\ell)$. We call this algorithm CLt, which stands for chained lesser than (see Algorithm 74). Note that this algorithm returns an additional bit R which tells whether the two inputs are equal. If this bit is not needed, some computations can be saved (remove line 10).

Algorithm 74: CLt

Requires: $(X_0, \cdots, X_{\ell-1}), (Y_0, \cdots, Y_{\ell-1})$ bit-wise encryption of x and y.
Outputs: Z, s.t. Z is an encryption of 1 if $x < y$, 0 otherwise
R, s.t. R is an encryption of 1 of $x = y$, 0 otherwise

1 $m \longleftarrow \lceil \log \ell \rceil$;
2 **for** $i = 0$ *to* $\ell - 1$ *(in parallel)* **do**
3 $A_i \longleftarrow \text{And}(X_i, Y_i)$;
4 $P_{0,i} \longleftarrow X_i Y_i / A_i^2$;
5 $G_{0,i} \longleftarrow Y_i / A_i$;
6 $B_{0,i} \longleftarrow \text{Not}(P_i)$;
7 $m \longleftarrow \lceil \log \ell \rceil$;
8 **for** $i = 1$ *to* m **do**
9 **for** $j = 0$ *to* $\lfloor \ell/2 \rfloor - 1$ *(in parallel)* **do**
10 $B_{i,j} \longleftarrow \text{And}(B_{i-1,2j}, B_{i-1,2j+1})$;
11 $T \longleftarrow \text{And}(P_{i-1,2j+1}, G_{i-1,2j})$;
12 $P_{i,j} \longleftarrow \text{And}(P_{i-1,2j}, P_{i-1,2j+1})$;
13 $G_{i,j} \longleftarrow \text{Or}(G_{i-1,2j+1}, T)$;
14 **if** ℓ *is odd* **then**
15 $B_{i,\lfloor \ell/2 \rfloor} \longleftarrow B_{i-1,\ell-1}$;
16 $P_{i,\lfloor \ell/2 \rfloor} \longleftarrow P_{i-1,\ell-1}$;
17 $G_{i,\lfloor \ell/2 \rfloor} \longleftarrow G_{i-1,\ell-1}$;
18 $\ell \longleftarrow \lceil \ell/2 \rceil$;
19 **return** $G_{m,0}, B_{m,0}$;

5.3 Comparison with other approaches

Now that we have presented our toolbox, we propose to compare it to the existing MPC protocols, especially those who are available in the ABB framework, and rely on the Paillier setting. For this purpose, we first give the complexities of our different protocols in Table 8. As explained in Section 4.3.3, the main metrics that we consider are the number of exponentiations, the number of synchronization steps and the size of the transcript. In the ElGamal setting, the exponentiations are cheaper and the key size is smaller, which impacts the size of the transcript. However, we usually require a larger number of synchronization steps, which could be a problem. For this reason, we propose various communication/computation trade-offs, based on careful parallelization.

Table 7: Leading terms and estimated run time for the cost of the MPC primitives of Ordinos and our toolbox; n_T is the number of participants, ℓ is the bit-length of the operands. The precomputable part can be evaluated before the tally; the run time is estimated for $n_T = 3$ and $\ell = 10$; all logarithms are in base 2 and the key sizes are $|q| = 256$ in the ElGamal setting and $|n| = 3072$ in the Paillier setting.

Functionality	Protocol	# exp.	time (s)	Synch. locks	Transcript size
Addition	–	0	–	0	0
	Add	$66\ell n_T$	0.20	$2\ell n_T$	$68\ell n_T\|q\|$
Multiplication	Mul	$9n_T$	0.14	2	$17n_T\|n\|$
	Mult	$99\ell^2 n_T$	3.0	$2\ell^2 n_T$	$102\ell^2 n_T\|q\|$
Equality	EQH	**precomp.** $39n_T\ell$ **comp.** $19n_T$	6.1	**precomp.** $O(n_T)$ **comp.** 4	$69n_T\ell\|n\|$
	Eq	$66\ell n_T$	0.20	$\log \ell n_T$	$68\ell n_T\|q\|$
Comparison	GTH	**precomp.** $43n_T\ell$ **comp.** $33n_T$	6.5	**precomp.** $O(n_T)$ **comp.** $7\log \ell$	$89n_T\ell\|n\|$
	Lt	$66\ell n_T$	0.20	$2\ell n_T$	$68\ell n_T\|q\|$

5.3.1 Comparison with Ordinos

Our first element of comparison is the concurrent contribution of Ordinos, which also proposes to use MPC protocols to achieve tally-hiding, using the ABB framework based in the Paillier encryption scheme that we introduced in Section 4.2. The main protocols used in Ordinos are the multiplication, the equality test and the comparison. Due to the number of different subprotocols that they use, it is difficult to come up with a meaningful formula to express their complexity: in particular, there are many exponentiations which have a different nature, with modulus and exponents of various sizes. In Table 7, we give some undervalued approximates for the various primitives or Ordinos, which will be our first element of comparison. Since the exponentiations are more expensive in the Paillier setting, this reveals that the computational costs and the transcript sizes for the comparison and the equality test are by one order of magnitude cheaper in our ElGamal toolbox compared to the existing MPC protocols based on the Paillier encryption. In addition to allowing a more efficient tally, the ElGamal setting is also beneficial on the voter-side. Indeed, using our toolbox or not is essentially transparent for the voter: recall that it is usual that they have to send several ElGamal encryptions of bits anyway, as this is the case in Helios. On the other hand, switching to the Paillier setting may be up to a thousand times more expensive for the voter, which is definitely an issue, if not prohibitive.

However, the multiplication will typically be cheaper in the Paillier setting, and the addition is free. In addition, the protocols in the Paillier setting usually require less synchronization steps than our protocols. Finally, most of the computations can be precomputed. For all these reasons, depending on which operation is the most used, the Paillier setting may still be preferable, especially when the number of voters is large since the aggregation is free in the Paillier setting.

5.3.2 Public tally hiding

Another approach to achieve tally-hiding was proposed in Kryvos [HKK+22]. By contrast with our approach and Ordinos's approach, they do not rely on MPC protocols. Instead, they allow the talliers to learn the "full" result of the election (*i.e.* what would be typically output by a traditional tally that relies on homomorphic tally or mixnet). From this, they locally compute the result r of the counting function, but only publish r as well as a ZKP of correctness. This gives the notion of *publicly* tally-hiding, where the talliers learn more information than the public. For the ZKP, Kryvos proposes to use Groth's SNARK (see Section 2.3.5) which, according to their benchmark, allows a more efficient tally than traditional MPC. In addition, the verification by the external auditor is way faster.

There are four major drawbacks with the approach of Kryvos. First, publicly tally hiding is not the same as fully tally hiding. In general, tally hiding is a counter-measure against Italian attacks, which are a mean for coercing voters into choosing a specific voting option. Yet, although a publicly tally hiding protocol would offer *some* protection against the coercer, all of it is lost when the coercer is a tallier.

A second remark concerns the technical solution proposed by Kryvos: to allow for efficient SNARK of correct tally, they need the ballots to have the form of a homomorphic commitment. This means that whenever Kryvos's solution can be used, it is also possible to use a homomorphic tally, for which there are arguably fewer risks of an Italian attack. For a counting function such as STV where tally hiding is needed the most, there is no smart way to encode a voting option into n_{choices} bits, in such a way that two encodings can be meaningfully added together. The only obvious solution would be to let n_{choices} be the total number of possible choices, as discussed in [HKK+22, Section 4.3] when they propose their solution for instant runoff voting (IRV), which is a (simple) specific case of STV. In the example of IRV, using their proposed solution would lead to $n_{\text{choices}} = \sum_{i=0}^{n_C} \binom{n_C}{i} i!$, where n_C is the number of candidates. This is impractical as soon as $n_C \geq 6$. More generally, there is a "real" risk for an Italian attack when n_{choices} is larger than the number of voters. Since Kryvos requires the voters to compute $n_T n_{\text{choices}}$ encryptions, it means that Kryvos is either impractical on the voter side, or not-quite-so necessary as it could be replaced by a classical homomorphic tally.

A third drawback is that Kryvos imposes a lot of computation stress on the voter side. Compared to Helios, the voters must typically compute n_T times more encryptions, and the corresponding ZKP is much more expensive: according to the provided benchmark, the time required for the voter to prove the validity of their ballot can be similar to that required for the talliers to compute the tally and prove its validity.

Finally, a fourth drawback is that Kryvos requires a honestly generated common reference string, as discussed in Section 2.3.5. In practice, the common reference string is obtained with an MPC protocol, which may be expensive given the number of scalars and group elements to generate. Although this generation can be done in advance, it is not clear how its cost compares to that of the tally itself, so that its computational complexity should be taken into account. In addition, the use of the SNARK imposes a not usual trust assumption on the participants of this MPC protocol: if the SNARK is not honestly generated, then verifiability is lost.

Table 8: Leading terms of the costs of the MPC primitives; n_T is the number of participants, N is the number of operands, ℓ is the bit-length of the operands, r is the precision in the division. All logarithms are in base 2. For the CSZ protocol, we express the computation cost as the number of exponentiations per participants, and the unit of the transcript size is the key size (typically 256 bits). For the other protocols, we express their cost as the number of CSZ required.

Functionality	Option	Protocol	# exp.	Synch. locks	Transcript size
Not	–	–	0	0	0
CSZ	original	[ST04]	$19n_T$	n_T	$18n_T$
	SUC-secure	CSZ	$33n_T$	n_T	$34n_T$
And, Or, Xor	–	And, Or, Xor	CSZ	CSZ	CSZ
If, Cond. swap	–	If, Swap	CSZ	CSZ	CSZ
Select, Cond. shift	–	Select, CLS, CRS	NCSZ	CSZ	NCSZ
Addition, subtraction	linear	Add, Sub	2ℓCSZ	2ℓCSZ	2ℓCSZ
	sublinear	UFCAdd	$\frac{3}{2}\ell\log\ell$CSZ	$2\log\ell$CSZ	$\frac{3}{2}\ell\log\ell$CSZ
Opposite	–	Neg	ℓCSZ	ℓCSZ	ℓCSZ
Aggregation	–	Aggreg	$3N$CSZ	$\frac{1}{2}\log(N)^2$CSZ	$3N$CSZ
	UFC	(use UFCAdd)	$5.54N$CSZ	$2N\log N\log\log N$CSZ	$5.54N$CSZ
Multiplication	–	Mult	$3\ell^2$CSZ	$2\ell^2$CSZ	$3\ell^2$CSZ
Division	–	Div	$3r\ell$CSZ	$2r\ell$CSZ	$3r\ell$CSZ
Equality	–	Eq	2ℓCSZ	$\log\ell$CSZ	2ℓCSZ
Comparison	linear	Lt	2ℓCSZ	2ℓCSZ	2ℓCSZ
	lin. + eq	LtEq	3ℓCSZ	2ℓCSZ	3ℓCSZ
	sublinear	CLt	4ℓCSZ	$2\log\ell$CSZ	4ℓCSZ
	sublin. + eq	CLt	5ℓCSZ	$2\log\ell$CSZ	5ℓCSZ
Min, max	linear	Min, Max	$(3\ell+\log N)N$CSZ	$2\ell\log N$CSZ	$(3\ell+\log N)N$
	sublinear	(CLt instead of Lt)	$(5\ell+\log N)N$CSZ	$2\log\ell\log N$CSZ	$(5\ell+\log N)N$CSZ
s largest	comp.	sInsert	$(N-\frac{s}{2})s(3\ell+\log N)$CSZ	$2\ell s(N-\frac{s}{2})$CSZ	$(3\ell+\log N)N$CSZ
	trade-off	sSelect	$Ns(3\ell+\log N)$CSZ	$2s\ell\log N$CSZ	$Ns(3\ell+\log N)$CSZ
	comm.	(use sublin. Max)	$Ns(5\ell+\log N)$CSZ	$2s\log\ell\log N$CSZ	$Ns(5\ell+\log N)$CSZ
Sorting	oblivious	OddEvenMergeSort	$\frac{3}{4}N\log(N)^2\ell$CSZ	$\ell\log(N)^2$CSZ	$\frac{3}{4}N\log(N)^2\ell$CSZ
	with CLt		$\frac{5}{4}N\log(N)^2\ell$CSZ	$\log\ell\log(N)^2$CSZ	$\frac{5}{4}N\log(N)^2\ell$CSZ

Chapter 6

Application of the toolbox to electronic voting

The toolbox that we provide in Chapter 5 allows several talliers, that each have a secret share of an ElGamal public key, to securely evaluate *any* function from the encrypted inputs. However, if one wants to evaluate some specific function, one needs to design a specific boolean circuit that corresponds to this function, and it is not clear whether this would lead to an efficient and practical MPC protocol. For this reason, we studied a large variety of popular counting functions and designed an MPC protocol for some of them, including Condorcet-Schulze, STV, Majority Judgment and the D'Hondt method. For each of these counting functions, we explain how the MPC protocol was designed and we give estimates for the resulting complexities; in addition, we provide an implementation in the case of the Condorcet-Schulze [sou22] method.

Contents

6.1 Homomorphic tally for the Condorcet methods 127
6.1.1 Existing approaches for Condorcet methods 127
6.1.2 A new proof of well-formedness for homomorphic ranked voting . . 128
6.2 A tally-hiding protocol for Condorcet-Schulze 133
6.2.1 The Schulze method 133
6.2.2 Ballots as lists of integers 134
6.2.3 Obtaining the adjacency matrix from the encrypted ballot 135
6.2.4 Computing the result from the encrypted adjacency matrix 135
6.2.5 Condorcet-Schulze, the bottom-line 135
6.2.6 Comparison with Ordinos 136
6.2.7 Implementation 138
6.2.8 A possible adaptation for the ranked pairs variant 139
6.3 A solution for single transferable vote 140
6.3.1 Existing solutions for STV in electronic voting 141
6.3.2 Choosing one version of STV 142
6.3.3 Ballots as lists of candidates 143
6.3.4 A tally-hiding protocol for academic STV 143
6.3.5 Complexity analysis 146
6.4 Majority Judgment 148
6.4.1 Existing approaches for computing the Majority Judgment 149

6.4.2 A new algorithm for cleartext Majority Judgment 150
6.4.3 Adaptation to the Paillier setting 150
6.4.4 An adaptation to the ElGamal setting 156
6.4.5 Comparison with [CPST18] 159
6.5 Single choice voting 159
6.5.1 Basic single choice voting 161
6.5.2 List voting: computing the D'Hondt method in MPC 161
6.6 Security of the toolbox in the context of electronic voting . . . 163
6.6.1 Universal verifiability 165
6.6.2 Privacy 165
6.7 Lessons learned 169

6.1 Homomorphic tally for the Condorcet methods

The Condorcet method was proposed to determine the most rightful candidate [Con85]. For this purpose, every voter must rank the candidates by order of preference, possibly with equalities. Then a *Condorcet winner* is a candidate that is preferred to every other candidate by a majority of voters. More formally, for all pair (i, j) of candidates, we denote $d_{i,j}$ the number of voters who (strictly) prefer candidate i over j. Then a Condorcet winner is a candidate i such that $d_{i,j} > d_{j,i}$ for all $j \neq i$. There can be at most one such winner.

Such a Condorcet winner may not exist, this is the *Condorcet paradox*. For example, consider Fig. 14, in which the ape is preferred from the beaver by a majority, the beaver is preferred from the capybara and the capybara is preferred ape by a majority. This shows that the relation "is preferred by a majority" is not transitive. It is commonly accepted that the Condorcet paradox occurs quite often, even if there is a large number of voters. For instance, if we suppose that all the preferences are equiprobable, the probability that the there is no Condorcet winner decreases towards $\frac{15}{16}$ when the number of voters increases [Geh81].

Alice's ballot
Ape
Beaver
Capybara

Bob's ballot
Beaver
Capybara
Ape

Charlie's ballot
Capybara
Ape
Beaver

Figure 14: Illustration of a Condorcet paradox with 3 voters and 3 candidates

To circumvent the Condorcet paradox, several methods exist. Those variants would pick the Condorcet winner as the winner whenever there is a Condorcet winner, but provide a way to determine a single winner otherwise (except with a small probability). Hence, they are Condorcet-compliant and we consider them as a Condorcet methods. The most popular ones are the Schulze method and ranked pairs; the Schulze method is notably used for Ubuntu elections [Ubu12].

6.1.1 Existing approaches for Condorcet methods

For most of the Condorcet methods, the result of the counting function can be deduced from the *preference matrix* m, whose coefficient $m_{i,j}$ is defined by $m_{i,j} = d_{i,j} - d_{j,i}$, where $d_{i,j}$ is the number of candidates that (strictly) prefer i over j. We use this matrix rather than d for

a technical reason only: for instance, it is antisymmetric. However, some authors may prefer to use d directly. Interestingly, this global preference matrix can be deduced as the sum of the individual preference matrix of the voters. Therefore, apart from using a decryption mixnet, there is a second natural strategy which consists of having the voter encode their choice (*i.e.* a permutation $c_1, \cdots, c_{n_C}$ of the candidates, but possibly with equalities) as a preference matrix m, where c_i is the rank of candidate i and

$$m_{i,j} = \begin{cases} 1 & \text{if } c_i < c_j \\ 0 & \text{if } c_i = c_j \\ -1 & \text{otherwise.} \end{cases}$$

The voters then encrypt each element of this matrix independently, which gives an encrypted matrix M. Then the matrix given by every voter can by homomorphically added together, so that decrypting the resulting product would only reveal the global preference matrix and not the individual preferences of any voter. However, to turn this idea into an actual voting system, we also need to explain how a voter can *prove* that the ballot is well-formed, that is, corresponds to the matrix of a total order (with equalities). This requires in particular to prove that if the voter prefers i over j and j over k then i is also preferred over k:

$$(m_{i,j} = 1) \wedge (m_{j,k} = 1) \Rightarrow (m_{i,k} = 1);$$

and similar relations when $m_{i,j}$ and $m_{j,k}$ are equal to 0 or -1, yielding $O({n_C}^3)$ statements, where n_C is the number of candidates. This is roughly how the voters produce their ballot in [HHK+21], where it is also assumed that voters cannot give the same rank to two candidates (*i.e.* the case $c_i = c_j$ is forbidden). To discharge the voter from such a proof effort, it is proposed in [HPT19] that the authorities shuffle each preference matrix in blocks and then decrypt it to check whether the ballot is well formed. However, this yields a privacy breach, unnoticed by the authors: for each voter, everyone learns the number of candidates placed at equality. Indeed, even though the matrix is shuffled before being decrypted, it is still possible to count the number of 0 which is encrypted in the matrix. In particular, everyone learns who votes blank.

6.1.2 A new proof of well-formedness for homomorphic ranked voting

In this thesis, we propose an alternative approach which allows the voters to prove the well-formedness of their matrix in $O({n_C}^2)$ exponentiations, while still allowing the voter to give the same rank to several candidates. This is of great interest if one is ready to leak the preference matrix, as the tally can be computed without any MPC protocol apart from the threshold decryption. We first present our proof strategy on the cleartexts, and then instantiate it on the ciphertexts using standard ZKP.

Proof strategy on the cleartexts. Suppose that Alice wants to vote the ordering $(1, \cdots, k)$ (*i.e.* the candidate number i is ranked i^{th}). Then her preference matrix would be as follows:

$$m_{\text{init}}[i,j] = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } i < j \\ -1 & \text{otherwise.} \end{cases}$$

Now, assume that Alice wants to rank $\sigma(i)^{th}$ the candidate number i, for some permutation σ. If the candidate number i were numbered $\sigma(i)$ instead, Alice could have voted using m_{init} as above.

This means that the preference matrix of Alice m_a is such that $m_a[\sigma^{-1}(i), \sigma^{-1}(j)] = m_{\text{init}}[i, j]$ for all (i, j). Therefore m_a can be obtained by using the permutation σ to shuffle m_{init} (using the permutation on the rows, then on the columns).

Finally, assume that Alice wants to give the same rank to several candidates and let r_i be the rank of candidate i according to her personal preferences. Alice first sorts the candidates according to their rank, in increasing order. For this purpose, she uses a permutation σ such that $\sigma(i) < \sigma(j) \implies r_i \leq r_j$. To obtain her preference matrix m_a from m_{init}, Alice first transforms m_{init} into m_σ, such that $m_\sigma[i, j] = m_a[\sigma^{-1}(i), \sigma^{-1}(j)]$. For this purpose, she computes a vector $\boldsymbol{b}$ of size $k-1$ such that for all i, $b_i = 1$ if $r_{\sigma^{-1}(i)} = r_{\sigma^{-1}(i+1)}$, and 0 otherwise. Afterwards, Alice modifies m_{init} diagonal by diagonal, so as to indicate that some candidates are ranked equal. (Since the preference matrix is antisymmetric, Alice only needs to compute the upper half, from which she deduces the lower half.) For the first diagonal, we have $m_\sigma[i, i+1] = 1 - b_i$.

For the $(j+1)^{th}$ diagonal $(i, i+j+1)_i$, assume that the previous diagonal has been computed. Then, as the candidates are sorted in order of preference, we have

$$m_\sigma[i, i+j+1] = \begin{cases} 0 & \text{if } (m_\sigma[i, i+j] = 0) \wedge (m_\sigma[i+1, i+j+1] = 0), \\ 1 & \text{otherwise.} \end{cases}$$

Therefore, Alice can apply an iterative algorithm, using the following formula:

$$\begin{aligned} m_\sigma[i, i+j+1] &= 1 - (1 - m_\sigma[i, i+j])(1 - m_\sigma[i+1, i+j+1]) \\ &= m_\sigma[i, i+j] + m_\sigma[i+1, i+j+1] - m_\sigma[i, i+j] m_\sigma[i+1, i+j+1]. \end{aligned} \tag{2}$$

Once m_σ is obtained, Alice can finally derive m_a by shuffling the rows and the columns, using the permutation σ.

An illustrative example with five candidates. Suppose, for instance, that there are five candidates, say, the Ape, the Beaver, the Capybara, the Dolphin and the Elephant, that are numbered from 1 to 5. Suppose that Alice wants to give them the ranks $r_1 = 3$, $r_2 = 3$, $r_3 = 1$, $r_4 = 3$ and $r_5 = 2$, as depicted in Table 9. In other words, Alice's preferred candidate is the Capybara, then Alice prefers the Elephant, and then Alice likes the Beaver just as much as the Dolphin and the Ape.

Table 9: Alice's choice when ranking the five candidates

Candidate	Ape	Beaver	Capybara	Dolphin	Elephant
Number	1	2	3	4	5
Rank	3	3	1	3	2
σ	3	4	1	5	2
σ^{-1}	3	5	1	2	4

To sort them, Alice can use the permutations σ, as depicted in Fig. 9. There may be several permutations that are consistent with the ranking, and Alice may also choose another permutation that, for instance, permutes the Beaver, the Dolphin and the Ape. From this permutation σ, she computes the vector b, depicted in Table 10. This vector is used to modify the first diagonal of m_{init}, as shown in Fig. 15.

Afterwards, the second, the third and the fourth diagonals are modified in turn, as shown in Fig. 16. This modification is done thanks to Eq. (2), which gives the next diagonal as a function of the previous one.

Table 10: The vector $\boldsymbol{b}$ that corresponds to Alice's choice

i	1	2	3	4
b_i	0	0	1	1

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 1 & 1 \\ -1 & -1 & 0 & 1 & 1 \\ -1 & -1 & -1 & 0 & 1 \\ -1 & -1 & -1 & -1 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & \mathbf{1} & 1 & 1 & 1 \\ -1 & 0 & \mathbf{1} & 1 & 1 \\ -1 & -1 & 0 & \mathbf{0} & 1 \\ -1 & -1 & -1 & 0 & \mathbf{0} \\ -1 & -1 & -1 & -1 & 0 \end{pmatrix}$$

Figure 15: Modification of the first diagonal, to encode equalities

Once the upper half is obtained, the bottom half is deduced thanks to the antisymmetric property. Finally, the final preference matrix is obtained by shuffling m_σ column by column then row by row, using the permutation σ, as shown in Fig. 17.

$$\begin{pmatrix} 0 & 1 & \mathbf{1} & 1 & 1 \\ -1 & 0 & 1 & \mathbf{1} & 1 \\ -1 & -1 & 0 & 0 & \mathbf{0} \\ -1 & -1 & -1 & 0 & 0 \\ -1 & -1 & -1 & -1 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & 1 & 1 & \mathbf{1} & 1 \\ -1 & 0 & 1 & 1 & \mathbf{1} \\ -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 0 & 0 \\ -1 & -1 & -1 & -1 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & 1 & 1 & 1 & \mathbf{1} \\ -1 & 0 & 1 & 1 & 1 \\ -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 0 & 0 \\ -1 & -1 & -1 & -1 & 0 \end{pmatrix}$$

Figure 16: Modification of the matrix, diagonal by diagonal, to ensure consistency

$$m_\sigma = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 1 & 1 \\ -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 \end{pmatrix} \xrightarrow[rows]{\text{Shuffle}} \begin{pmatrix} -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ -1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 1 \end{pmatrix} \xrightarrow[columns]{\text{Shuffle}} \begin{pmatrix} 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & -1 & 0 & -1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & -1 & 0 & -1 \\ 1 & 1 & -1 & 1 & 0 \end{pmatrix} = m_a$$

Figure 17: Derivation of the preference matrix by permuting m_σ

Algorithm on the ciphertexts. To summarize our construction, we recap the procedure to provide a ballot and prove its well-formedness in Algorithm 75. In this algorithm, we require at line 7 that the voter produces a ZKP that a ciphertext B_i is indeed an encryption of a bit $b_i \in \{0, 1\}$. For this purpose, a standard ZKP such as Algorithm 16 can be used. In addition, we also require at line 16 that the voter proves that some ciphertext $Z_{i,i+j+1}$ encrypts the product of the plaintexts of two other ciphertexts $M_\sigma[i, i+j]$ and $M_\sigma[i+1, i+j+1]$. We can easily build such a proof using standard ZKP; see Algorithm 76. Finally, we also require that the voter shuffles the matrix. For this purpose, we can use ShuffleMatrix, which is presented in Section 2.4.3.

Technically, a ballot can be considered as a matrix M_σ that encodes which candidate are ranked at equality, a matrix M_a that encodes the permutation and a ZKP Π. However, since M_σ can be recovered from Π, we do not include it in the output of Algorithm 75.

To verify that a ballot is well-formed, one can first verify all the ZKP $\pi_i^{0/1}$. Then, using the B_i's, M_{init} and the $Z_{i,i+j+1}$'s, the verifier can compute the matrix M_σ. The fact that M_σ is well-formed can be verified by checking all the ZKP $\pi^{\text{Mult}}_{i,i+j+1}$, using M_σ and the $Z_{i,i+j+1}$'s. Finally, the verifier verifies the proofs of a shuffle using π^{Shuffle}, M_σ and M_a.

Algorithm 75: MatrixBallot

Requires: pk, a public exponential ElGamal encryption key
E_1, E_0, E_{-1}, trivial encryptions of 1, 0 and -1
$r_1, \cdots, r_{n_C}$, the rank of each candidate
Outputs: M_a, the encrypted matrix of preferences
Π a ZKP of well-formedness

Sort the candidate by increasing order of rank;
This gives a permutation σ such that $\sigma(i) < \sigma(j) \implies r_i \leq r_j$;
for $i = 1$ *to* n_C **do**
  if $r_{\sigma^{-1}(i)} = r_{\sigma^{-1}(i+1)}$ **then** $b_i \longleftarrow 1$ **else** $b_i \longleftarrow 0$;
  $r_i \xleftarrow{\$} \mathbb{Z}_q$;
  $B_i \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(b_i, r_i)$;
  $\pi_i^{0/1} \longleftarrow \mathrm{Pok}^{0/1}(B_i, b_i, r_i)$ (* see Algorithm 16 *);
  $m_\sigma[i,i] \longleftarrow 0$; $m_\sigma[i,i+1] \longleftarrow 1 - b_i$;
  $r_{i,i} \longleftarrow 0$; $r_{i,i+1} \longleftarrow -r_i$;
  $M_\sigma[i,i] \longleftarrow E_0$; $M_\sigma[i,i+1] \longleftarrow E_1/B_i$;
for $j = 1$ *to* $n_C - 1$ **do**
  for $i = 1$ *to* $n_C - j - 1$ **do**
    $m_\sigma[i,i+j+1] \longleftarrow m_\sigma[i,i+j] + m_\sigma[i+1,i+j+1] - m_\sigma[i,i+j]m_\sigma[i+1,i+j+1]$;
    $X \longleftarrow M_\sigma[i,i+j]$; $x \longleftarrow m_\sigma[i,i+j]$; $r_x \longleftarrow r_{i,i+j}$;
    $Y \longleftarrow M_\sigma[i+1,i+j+1]$; $r_z \xleftarrow{\$} \mathbb{Z}_q$;
    $Z_{i,i+j+1}, \pi^{\mathrm{Mult}}_{i,i+j+1} \longleftarrow \mathrm{ZKPmult}(X, x, r_x, Y, r_z)$;
    $r_Z \longleftarrow x r_{i+1,i+j+1} + r_z$;
    $M_\sigma[i,i+j+1] \longleftarrow M_\sigma[i,i+j]M_\sigma[i+1,i+j+1]/Z_{i,i+j+1}$;
    $r_{i,i+j+1} \longleftarrow r_{i,i+j} + r_{i+1,i+j+1} - r_Z$;
for all (i,j) s.t. $i < j$ **do** $M_\sigma[j,i] = 1/M_\sigma[i,j]$;
$M_a, \pi^{\mathrm{Shuffle}} \longleftarrow \mathrm{ShuffleMatrix}(M_\sigma, \sigma)$;
$\Pi \longleftarrow \pi^{\mathrm{Shuffle}} || (Z_{i,i+j+1}, \pi^{\mathrm{Mult}}_{i,i+j+1})_{i,j} || (B_i, \pi_i^{0/1})_i$;
return M_a, Π;

Algorithm 76: ZKmult

Requires: G a group of prime order q
pk, a public exponential ElGamal encryption key
X, x, r_x, s.t. $X = \mathrm{Enc}_{\mathsf{pk}}(x, r_x)$
Y, any ciphertext
$r_z \in \mathbb{Z}_q$, a randomness

Outputs: Z, a reencryption of Y^x
π^{Mult}, a ZKP of well-formedness

$Z \longleftarrow \mathrm{Renc}_{\mathsf{pk}}(Y^x, r_z)$;
$\alpha, \rho_x, \rho_z \xleftarrow{\$} \mathbb{Z}_q$;
$c_z \longleftarrow \mathrm{Renc}_{\mathsf{pk}}(Y^\alpha, \rho_z)$;
$c_x \longleftarrow \mathrm{Enc}(\alpha, \rho_x)$;
$d \longleftarrow \mathrm{hash}(\mathsf{pk}||X||Y||Z||c_x||c_z)$;
$a_x \longleftarrow \alpha + xd$; $a_{rx} \longleftarrow \rho_x + r_x d$;
$a_{rz} \longleftarrow \rho_z + r_z d$;
$\pi^{\mathrm{Mult}} \longleftarrow (c_x, c_z, a_x, a_{rx}, a_{rz})$;
return Z, π^{mul}

Algorithm 77: Verification algorithm

Requires: G a group of prime order q
pk, a public exponential ElGamal encryption key
X, Y, Z, three ciphertexts
$\pi^{mul} = (c_x, c_z, a_x, a_{rx}, a_{rz})$, a ZKP

Outputs: 1 if the proof is valid, 0 otherwise

$d \longleftarrow \mathrm{hash}(\mathsf{pk}||X||Y||Z||c_x||c_z)$;
Verify the following equalities:
  $c_x \stackrel{?}{=} \mathrm{Enc}_{\mathsf{pk}}(a_x, a_{rx})X^{-d}$;
  $c_z \stackrel{?}{=} \mathrm{Renc}_{\mathsf{pk}}(Y^{a_x}, a_{rz})Z^{-d}$;
if so **then return** 1 **else return** 0;

Table 11: Complexity of the proof of validity of the ballot matrix, where n_C is the number of candidates

Prover (# exp.)	Verifier (# exp.)	Transcript (×256 bits)
$11.5{n_C}^2$	$11.5{n_C}^2$	$8.5{n_C}^2$

Security of the construction. Our construction inherits the completeness, the computational soundness and the statistical zero knowledge from that of the standard PoK and the Terelius-Wikström proof of a shuffle. Indeed, if we consider that a ballot is a couple of encrypted matrices M_σ, M_a, then the ciphertexts $Z_{i,i+j+1}$ and B_i can be recovered from M_σ and M_{init}. Then, the remaining of Π can be simulated thanks to the zero knowledge property of the standard PoK and the proof of a shuffle.

Complexity analysis. We now give the complexity of our construction. To simplify the expression, we only give the leading term of the formula. First, the voter has to produce $n_C - 1$ encryptions of a bit and the corresponding ZKP, which leads to $O(n_C)$ exponentiations. Then, for $j \in [1, n_C - 1]$, the $(j+1)$th diagonal has $(n_C - j - 1)$ ciphertexts to compute. For each of these ciphertexts, the only expensive operation is to compute ZKPmult; indeed, the other operations are arithmetic operations in $\mathbb{Z}_q$. Since $x \in \{0, 1\}$, computing Y^x is cheap; therefore, ZKPmult requires 7 exponentiations to compute and to verify. Finally, the cost of ShuffleMatrix is given in Table 3. Overall, producing and proving the validity of a ballot requires $\frac{23}{2}{n_C}^2$ exponentiations. For comparison, the "naive" cubic algorithm requires $18{n_C}^3$ exponentiations for the prover and $20{n_C}^3$ exponentiations for the verifier.

6.2 A tally-hiding protocol for Condorcet-Schulze

The previous section gives a solution for most of the Condorcet methods, based on a homomorphic tally. However, this implies to reveal the global preference matrix, which contains more information than just the set of the winners. For this reason, we investigate the possibility of performing a fully tally hiding protocol, that only reveals which candidate wins the election. We especially focus on the Schulze method, as it is by far the most popular variant of Condorcet.

6.2.1 The Schulze method

The Schulze method consists of several steps. First, for all pair (i, j) of candidates, compute $d_{i,j}$, which is the number of voters who (strictly) prefer candidate i over candidate j. Second, deduce $b_{i,j} = d_{i,j} - d_{j,i}$. For all pair of candidates (u, v), a *path* p of length ℓ from u to v is a finite sequence of $\ell + 1$ candidates such that $u = p_0$ and $v = p_\ell$. We say that $(i, j) \in p$ if there exists an index $0 \leq k < \ell$ such that $i = p_k$ and $j = p_{k+1}$. The *strength* of a path p is defined as $s(p) = \min_{(i,j)\in p} b_{i,j}$. With this in mind, the third step of the Schulze method is to compute $f_{i,j} = \max_{\sigma\in[i\rightsquigarrow j]} s(\sigma)$, where $[i \rightsquigarrow j]$ denotes the set of all paths from i to j. Finally, i is a winner by the Schulze method if $f_{i,j} \geq f_{j,i}$ for all j.

Note that it is possible that there are several winners according to the Schulze method. However, when such a case arises, then for all pair of winner (i, j), we have $f_{i,j} = f_{j,i}$. Therefore, the Schulze method has the resolvability property, which means that a single winner is output with high probability, and the probability that a tie occurs decreases towards 0 when the number of voters increases. Hence, the Schulze method is indeed a solution to the Condorcet paradox.

Computing $f_{i,j}$ for all (i, j) is actually a well-known problem in graph theory, which is called the *maximum capacity* problem [Pol60]. To solve this, we can use the Floyd-Warshall algorithm [Flo62, War62] on the matrix b, but where the min operator is replaced by the max operator and the + operator is replaced by the min operator, as explained in [Pai66]. However, remark that some of the coefficients of b may be negative, which can be a problem in our ElGamal toolbox where the subtractions are computed modulo a power of 2. For this reason, we use the *adjacency matrix* a, defined by

$$a_{i,j} = \begin{cases} d_{i,j} - d_{j,i} & \text{if } d_{i,j} \geq d_{j,i} \\ 0 & \text{otherwise,} \end{cases}$$

which only has non-negative elements. In Lemma 10, we show that the Schulze winner can be deduced from the positive matrix a instead of b.

Lemma 10. *Let n_C be the number of candidates, and b an antisymmetric matrix of size $n_C \times n_C$ (i.e. for all i, j, $b_{i,j} = -b_{j,i}$). Let a be the matrix defined by the coefficients $a_{i,j} = \max(0, b_{i,j})$. For all (i, j), we denote*

$$f_{i,j} = \max_{\sigma\in[i\rightsquigarrow j]} \min_{(k,\ell)\in\sigma} b_{k,\ell}$$
$$f'_{i,j} = \max_{\sigma\in[i\rightsquigarrow j]} \min_{(k,\ell)\in\sigma} a_{k,\ell}.$$

With these notations, we have $\forall i, (\forall j, f_{i,j} \geq f_{j,i}) \iff (\forall j, f'_{i,j} \geq f'_{j,i})$.

Proof. For all path p, we denote $s(p) = \min_{(i,j)\in p} b_{i,j}$ and $s'(p) = \min_{(i,j)\in p} a_{i,j}$.

Let i be a candidate, suppose that for all j, $f_{i,j} \geq f_{j,i}$ (*i.e.* i is a Schulze winner). Let j be any candidate. If $j = i$, clearly $f'_{i,j} \geq f'_{j,i}$, so we assume that $j \neq i$. Since $j \neq i$, there is no path

from i to j (nor from j to i) of length 0. Now, let p be a path from i to j which maximizes $s(p)$, so that $f_{i,j} = s(p)$. We consider two cases:

First, assume that $b_{p_k,p_{k+1}} < 0$ for some k. Then $f_{i,j} = s(p) < 0$ and, for all path p' from j to i, $s(p') \leq f_{j,i} < 0$. Consequently, there exists k' such that $b_{p'_{k'},p'_{k'+1}} < 0$, so that $a_{p'_{k'},p'_{k'+1}} = 0$, thus $s'(p') = 0$. Since this is holds for all p', $f'_{j,i} = 0 \leq f'_{i,j}$.

Second, if $b_{p_k,p_{k+1}} \geq 0$ for all k, then for all k, $a_{p_k,p_{k+1}} = b_{p_k,p_{k+1}}$, so that $s(p) = s'(p)$. Now consider any path p' from j to i. If $b_{p'_{k'},p'_{k'+1}} \geq 0$ for all k', then $s'(p') = s(p') \leq f_{j,i} \leq f_{i,j} = s(p) = s'(p) \leq f'_{i,j}$. If there exists k' such that $b_{p'_{k'},p'_{k'+1}} < 0$, then $s'(p') = 0 \leq f'_{i,j}$. Hence, $s'(p') \leq f'_{i,j}$ for all p', so that $f'_{j,i} \leq f'_{i,j}$.

Conversely, let i such that $f'_{j,i} \leq f'_{i,j}$ for all j. Let j be any candidate (as above, w.l.o.g. we assume that $i \neq j$). We consider three cases.

First, suppose that $f_{i,j} < 0$. Then for all path p from i to j, $s(p) \leq f_{i,j} < 0$ so that there exists $(u, v) \in p$ such that $b_{u,v} < 0$ (we call this proposition $*$). In particular, (i, j) is a path from i to j. Hence, $b_{i,j} < 0$, so $b_{j,i} = -b_{i,j} > 0$, therefore $b_{j,i} = a_{j,i}$ and $f'_{j,i} \geq s'(j, i) = a_{j,i} = b_{j,i} > 0$. On the other hand, by $*$ we have $f'_{i,j} = 0$, which contradicts $f'_{j,i} \leq f'_{i,j} = 0$. Therefore $f_{i,j} \geq 0$.

Second, suppose that $f_{i,j} = 0$. Then for all path p from i to j, $s(p) \leq f_{i,j}$ so that there exists $(u, v) \in p$ such that $b_{u,v} \leq 0$, hence $f'_{i,j} = 0$. Let p' be a path from j to i (of length $n' > 0$). Suppose that for all $(u, v) \in p'$, $b_{u,v} > 0$. Then $0 < s'(p) \leq f'_{j,i}$, which contradicts $f'_{j,i} \leq f'_{i,j}$. Consequently, there exists $(u, v) \in p'$ such that $b_{u,v} \leq 0$, therefore $s(p') \leq 0 = f_{i,j}$. This holds for all p' so $fj, i \leq f_{i,j}$.

Finally, suppose that $f_{i,j} > 0$. Let p' be a path from j to i. If there exists $(u, v) \in p'$ such that $b_{u,v} \leq 0$, then $s(p') \leq 0 < f_{i,j}$. Otherwise, for all $(u, v) \in p'$, $b_{u,v} > 0$ so $s(p') = s'(p') \leq f'_{j,i} \leq f'_{i,j} \leq f_{i,j}$. □

6.2.2 Ballots as lists of integers

Before describing the tally process, we need to present the expected form of the ballots. We decide to make the cost on the voter side as cheap as possible, because it is often critical in electronic voting. While the voter may only have a limited computational power, the tallier and the auditor may have access to a powerful server or rent a supercomputer. For this reason, we do not use the solution from Section 6.1.2. Instead, we ask the voter to provide $\log n_C$ encryption of 0 or 1 for each candidate, which encodes the desired rank for the candidate. This way, they can give the rank they want to each candidate, without restriction, which accounts for the possibility of giving the same rank to several candidates. Note that there may be gaps in the given ranks; for instance, a voter may give the rank 0 to one candidate and the rank 2 to two candidates, without giving the rank 1 to any candidate. In addition, since $n_C - 1$ may not be a power of 2, a voter may give a rank that exceeds $n_C - 1$. This is not a problem and we consider that any ballot which encrypts $n_C \log n_C$ bits is valid. Thanks to the relative absence of restrictions, the voter can produce and prove the validity of their ballot in about $6n_C \log n_C$ exponentiations. On this occasion, we refer to [DPP22a], where various solutions to simultaneously encrypt many bits and produce the corresponding ZKP in the ElGamal setting are proposed. Compared to Algorithm 16, they remark that it is possible to save 2 exponentiation per bit without changing anything except the way the proof is generated. Since the voter also needs to compute 2 exponentiation per encryption, this gives a total of $6n_C \log n_C$ exponentiations per voter ballot, which is way cheaper than in the homomorphic variant.

6.2.3 Obtaining the adjacency matrix from the encrypted ballot

We now explain how the talliers can collectively turn every ballot into the corresponding individual preference matrix. For this purpose, they use an MPC protocol that operates on encrypted data. First, we assume here that each ballot consists of $n_C \lceil \log(n_C+1) \rceil$ ciphertexts, along with as many ZKP that prove that they are encryptions of 0 or 1. Those ciphertexts are interpreted as n_C bitwise encrypted integers, denoted $\boldsymbol{R_1^i}, \cdots, \boldsymbol{R_{n_C}^i}$, where i is the index of the voter. The corresponding integers represent the rank that the voter i gives to each candidate.

To compute an encryption of the preference matrix of a voter, the talliers can use the protocol LtEq which returns two encrypted bits: one for the lesser-than test and the other for the equality test (see Section 5.1.3). This gives Algorithm 78. At this point, the encrypted matrices can be aggregated using the homomorphic property, and the resulting product can be decrypted. This gives yet another possibility for the homomorphic tally. However, since we assumed that the talliers are ready to perform complex MPC, we can also aggregate the matrices using the Aggreg algorithm, which gives the bitwise encryption of the global preference matrix. Finally, the adjacency matrix can be deduced using the Sub protocol. Remark that in this protocol, the last encrypted bit indicates whether the result is negative or not. Hence, to ensure that the result are all positive, we can use the CSZ protocol to set the negative values to zero as desired.

Algorithm 78: BtoM

Requires: A group G of prime order q
pk, an exponential ElGamal public key
E_1, E_0, trivial encryptions of 1 and 0
$\boldsymbol{R_1}, \cdots, \boldsymbol{R_{n_C}}$, bitwise encryptions of the same size
$\ell = \lceil \log(n_C+1) \rceil$, where n_C is the number of candidates

Outputs: M, the corresponding encrypted preference matrix

```
for i = 1 to n_C (in parallel) do
    M_{i,i} ⟵ E_0;
    for j = i + 1 to n_C (in parallel) do
        Z, T ⟵ LtEq(R_i, R_j);
        M_{i,j} ⟵ Z;
        M_{j,i} ⟵ CSZ(E_1/Z, T);
return M;
```

6.2.4 Computing the result from the encrypted adjacency matrix

Finally, once the global adjacency matrix has been obtained, one can derive the result from the Floyd-Warshall algorithm; see Algorithm 80.

6.2.5 Condorcet-Schulze, the bottom-line

We provide a bottom line where we give again all the necessary details. First, to submit a ballot, a voter can use the procedure given in Algorithm 81. We made it as simple as possible, with the objective to minimize the computation stress on the voter side. Namely, the voter can give any rank to any candidate, without any restriction (except that the rank must in the range $[0, 2^\ell - 1]$, where n_C is the number of candidates and $\ell = \lceil \log(n_C+1) \rceil$). This way, the voter only needs $6 n_C \log n_C$ exponentiations to vote. The previous approach, presented in Section 6.1.2, requires

Algorithm 79: FW (Floyd-Warshall)

Requires: $(P)_{i,j}$, the encrypted adjacency matrix
n_C, the # of candidates
Outputs: $(S)_{i,j}$, s.t. $\boldsymbol{S_{i,j}}$ is an encryption of the strength of the strongest path from i to j

$S \longleftarrow P$;
for $k = 1$ *to* n_C **do**
  for $i = 1$ *to* n_C *(in parallel)* **do**
    for $j = 1$ *to* n_C *(in parallel)* **do**
      (* proceed only if $(i \neq j)$ *);
      $T \longleftarrow \mathrm{Lt}(\boldsymbol{S_{i,k}}, \boldsymbol{S_{k,j}})$;
      $\boldsymbol{A_{i,j}} \longleftarrow \mathrm{If}(T, \boldsymbol{S_{i,k}}, \boldsymbol{S_{k,j}})$;
      $T \longleftarrow \mathrm{Lt}(\boldsymbol{A_{i,j}}, {}_b mS_{i,j})$;
      $\boldsymbol{B_{i,j}} \longleftarrow \mathrm{If}(T, \boldsymbol{S_{i,j}}, \boldsymbol{A_{i,j}})$;
  for all $(i \neq j)$ **do** $\boldsymbol{S_{i,j}} \longleftarrow \boldsymbol{B_{i,j}}$;
return S;

Algorithm 80: Schulze

Requires: $(P)_{i,j}$, the encrypted adjacency matrix
n_C, the # of candidates
Outputs: w, the indicator of the Schulze winners

$(S)_{i,j} \longleftarrow \mathrm{FW}(P)$;
for $i = 1$ *to* n_C *(in parallel)* **do**
  for $j \neq i$ *(in parallel)* **do**
    $B_j \longleftarrow \mathrm{Not}(\mathrm{Lt}(\boldsymbol{S_{i,j}}, \boldsymbol{S_{j,i}}))$;
  $W_i \longleftarrow \mathrm{And}((B_j)_{j \neq i})$;
  $w_i \longleftarrow \mathrm{Dec}(W_i)$;
return $w_1, \cdots, w_{n_C}$;

about $11.5{n_C}^2$ exponentiations. Note that only the ordering of the candidates is of interest, so that ranking three candidates 1, 1 and 2 is the same as ranking them 0, 0 and 3.

Second, the talliers turn each individual ballot into the corresponding encrypted preference matrix, using the BtoM protocol. Then they aggregate all the individual preference matrices into a single preference matrix, using the Aggreg protocol. Finally, they turn the global preference matrix into an adjacency matrix and deduce the result from the Schulze algorithm. This gives Algorithm 82, whose complexity is detailed in Table 12.

6.2.6 Comparison with Ordinos

Ordinos [KLM+20] is a concurrent contribution that allows tally-hiding in the Paillier setting. It was extended in [HHK+21] to cover various counting functions, including the Schulze method. Therefore, we discuss how our work compares to that of Ordinos. First, Ordinos does not allow the voters to rank several candidates at equality, which is too restrictive; therefore they do not provide a solution for Condorcet-Schulze in general. Second, Ordinos requires the Paillier encryption scheme and asks the voters to compute $O({n_C}^3)$ exponentiations to cast a ballot. By comparison, we use the ElGamal encryption scheme (hence the exponentiation is a lot cheaper on the voter side) and we only require the voters to compute $6n_C \log n_C$ exponentiations to cast their ballot. That being said, we can still compare the overall performances of both tally-hiding schemes. For this purpose, we use [HHK+21, Fig. 7] to deduce the overall complexity of computing a Condorcet-Schulze tally with Ordinos. However, we also include the cost of verifying the validity of the ballots in the task of the talliers: if they do not verify the ZKP, then a malicious server can break privacy, for instance using a replay attack.

In Table 13, we give the complexity estimates of various solutions for tallying the Condorcet-Schulze method, including our fully tally hiding protocol as well as Ordinos'. From the resulting

Algorithm 81: Ballot casting procedure for the Condorcet-Schulze method

Requires: G, a group of prime order q
pk, an exponential ElGamal public key
n_C, the number of candidates
$\ell = \lceil \log(n_C + 1) \rceil$
$r_1, \cdots, r_{n_C}$ the ranks given to each candidate

for $i = 1$ to n_C **do** Writes r_i in base 2: $r_i = \sum_{j=0}^{\ell-1} b_{i,j} 2^j$;
for $i = 1$ *to* n_C **do**
  for $j = 0$ *to* $\ell - 1$ **do**
    $\rho_{i,j} \longleftarrow \mathbb{Z}_q$;
    $R_{i,j} \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(b_{i,j}, \rho_{i,j})$;
    $\pi_{i,k}^{0/1} \longleftarrow \mathrm{PoK}(R_{i,j}, b_{i,j}, \rho_{i,j})$ (* see [DPP22a] *);
return $(R_{i,j}, \pi_{i,j})_{i,j}$;

Algorithm 82: Condorcet-Schulze

Requires: $(\boldsymbol{R_1^i}, \cdots, \boldsymbol{R_{n_C}^i})_{i=1}^{n_V}$, the n_V encrypted ballots
n_C, the number of candidates

Outputs: w, the indicator of the set of the Schulze winners

for $v = 1$ *to* n_V *(in parallel)* **do**
  $M_v \longleftarrow \mathrm{BtoM}(\boldsymbol{R_1^v}, \cdots, \boldsymbol{R_{n_C}^v})$;
for $i = 1$ *to* n_C *(in parallel)* **do**
  for $j = 1$ *to* n_C *(in parallel)* **do**
    $\boldsymbol{M[i,j]} \longleftarrow \mathrm{Aggreg}(M_1[i,j], \cdots, M_{n_V}[i,j])$;
for $i = 1$ *to* n_C *(in parallel)* **do**
  $\boldsymbol{A[i,i]} \longleftarrow 0^{\mathrm{bits}}$;
  for $j = i + 1$ *to* n_C *(in parallel)* **do**
    $\boldsymbol{D}, T \longleftarrow \mathrm{Sub}(\boldsymbol{M[i,j]}, \boldsymbol{M_{[j,i]}})$;
    $\boldsymbol{A[j,i]} \longleftarrow \mathrm{CSZ}(\mathrm{Neg}(\boldsymbol{D}), T)$;
    $\boldsymbol{A[i,j]} \longleftarrow \mathrm{CSZ}(\boldsymbol{D}, \mathrm{Not}(T))$;
return $\mathrm{Schulze}(A)$;

Table 12: Leading terms in the complexity for fully tally-hiding the Schulze method; the computation, communication and transcript sizes are given as the number of CSZ required

Part	Computations	Communications	Transcript
BtoM	$\frac{3}{2} n_V {n_C}^2 \log n_C$	$2 \log n_C$	$\frac{3}{2} n_V {n_C}^2 \log n_C$
Aggreg	$3 n_V {n_C}^2$	$\frac{1}{2} \log(n_V)^2$	$3 n_V {n_C}^2$
Adjacency	$5 {n_C}^2 \log n_V$	$3 \log n_V$	$5 {n_C}^2 \log n_V$
FW	$6 {n_C}^3 \log n_V$	$4 n_C \log n_V$	$6 {n_C}^3 \log n_V$
Result	$3 {n_C}^2 \log n_V$	$3 \log n_V$	$3 {n_C}^2 \log n_V$
Total	$\frac{3}{2} n_V {n_C}^2 \log n_C$ $+6 {n_C}^3 \log n_V$	$4 n_C \log n_V$	$\frac{3}{2} n_V {n_C}^2 \log n_C$ $+6 {n_C}^3 \log n_V$

Table 13: Leading terms of the cost of various solutions for Condorcet-Schulze. n_V is the number of voters, n_C is the number of candidates, n_T is the number of talliers. The unit of the transcript size is the key size, which is 256 bits in the ElGamal setting and 3072 bits in the Paillier setting.

Version	Voters	Authorities		Transcript size
	# exp.	# exp.	# synch. locks	
[HPT19]	$6{n_C}^2$[1]	$14n_V{n_C}^2n_T$	$2n_T$	$3n_V{n_C}^2n_T$
[HHK+21] (Paillier setting)	$5{n_C}^3$[2]	**precomp.** $78{n_C}^3n_T \log n_V$ **comp.** $6n_V{n_C}^3$[3] $+66{n_C}^3n_T$	**precomp.** $O(n_T)$ **comp.** $14n_C \log\log n_V$	$9n_V{n_C}^3+$ $178{n_C}^3n_T \log n_V$
Section 6.1.2[4]	$17.5{n_C}^2$	$15.5n_V{n_C}^2$	1	$8.5n_V{n_C}^2$
Partial MPC[4] (ours)	$6n_C \log n_C$	$49.5n_V{n_C}^2n_T \log n_C$	$2n_T \log n_C$	$50n_V{n_C}^2n_T \log n_C$
Full MPC (ours)	$6n_C \log n_C$	$49.5n_V{n_C}^2n_T \log n_C$ $+198{n_C}^3n_T \log n_V$	$4n_Cn_T \log n_V$	$50n_V{n_C}^2n_T \log n_C$ $+204{n_C}^3n_T \log n_V$

[1] [HPT19] leaks the adjacency matrix. In addition, for each ballot, the number of candidates ranked at equality is public. In particular, who voted blank is known to everyone.
[2] [HHK+21] does not allow voters to give the same rank to several candidates.
[3] [HHK+21] originally does not take into account the cost of verifying the ZKP provided by the voters.
[4] Leaks the adjacency matrix.

formulas, it appears that the computational time in Ordinos is greater than ours. Indeed, recall that the cost of an exponentiation in the Paillier setting is larger than in the ElGamal setting. Nevertheless, most of the computations in the Ordinos setting can be precomputed, so that Ordinos' performances are roughly comparable to ours (except that precomputing does have a cost). Apart from that, it appears that the solution of Ordinos result in a transcript which is about one order of magnitude larger than ours, due to the Paillier ciphertexts being about 12 times larger than the typical ElGamal ciphertexts.

One metric for which Ordinos performs better than our toolbox is the number of synchronization steps. However, if this becomes a real problem, we can use the CLt protocol instead of the Lt protocol for the comparison, which would lead to a similar communication cost to that of Ordinos. As shown in Table 8, the extra cost on the computation side is reasonable and we would still be comparable to Ordinos in terms of computational efficiency. Given the gain in efficiency in the voter size and all the other advantages of the ElGamal setting compared to the Paillier setting, we outperform Ordinos when it comes to computing a Condorcet-Schulze tally.

That being said, Table 13 only gives approximate formulas, and the latter are pretty complex. Therefore, a complementary way to compare Ordinos' performances to that of our toolbox would be to run both implementations. We discuss of this comparison in Section 6.2.7.

6.2.7 Implementation

In order to evaluate the practical feasibility of our approach, we have written a prototype implementation in the ElGamal setting. The `libsodium` library is used for randomness and all elliptic curve and hashing operations. The rest is implemented as a standalone C++ program. It is available in [sou22] and is published as a free software. Most of the primitives of our toolbox

Table 14: Wall-clock time and transcript size of fully tally-hiding Condorcet-Schulze.

voters	5 candidates	10 candidates	20 candidates
64	1m50s / 49 MB	8m30s / 0.30 GB	45m / 1.8 GB
128	2m40s / 87 MB	12m / 0.51 GB	1h27m / 2.9 GB
256	4m35s / 160 MB	20m / 0.88 GB	2h37m / 4.8 GB
512	8m10s / 305 MB	34m / 1.6 GB	4h43m / 8.6 GB
1024	15m / 595 MB	1h05m / 3.1 GB	8h50m / 16 GB

have been implemented and, as a proof of concept, we have written a fully tally-hiding protocol for Condorcet-Schulze.

We ran our software with various parameters. In order to compare to Ordinos [HHK+21], we also considered 3 trustees (and no threshold). Our experimental setting is a single server hosting two 16-core AMD EPYC 7282 processors and 128 GB or RAM. Each of the 3 trustees runs 4 computing threads and a few scheduling and I/O threads. The communication between the trustees is emulated via the loopback network interface. Thus, all the network system calls are performed by the program, even though this is just a simulation. In Table 14, we summarize the cost in terms of wall-clock time and the size of the transcript, measured by the program.

This experiment demonstrates that the approach is sound and in the realm of practicability, for moderate-sized elections. By comparison, [HHK+21] reports a computation time of more than 9 days and 10 hours for tallying a 20-candidates Condorcet-Schulze election, which does not account for the verification of the validity of the ballots. Since each ballot requires a cubic number of ZKP with respect to the number of candidates, verifying all the ballots may be more expensive than computing the tally in the Schulze setting, depending on the number of voters. In addition, the implementation of Ordinos, which is available at [Ord], does not account for the precomputation, *i.e.* the subprotocols of the equality test MPC protocol that are presented in Section 4.2.2. Yet, most of the computations in Ordinos are precomputations, so that it is not clear whether they can be disregarded. Typically, their solution would require a few months of precomputation time.

6.2.8 A possible adaptation for the ranked pairs variant

The Schulze method is not the only variant of Condorcet. Among them, Tideman's method, also known as ranked pairs [Tid87], appears to be the second most popular. Compared to the Schulze method, it provides the same properties as a counting function: it is Condorcet-compliant, has resolvability and is resistant against strategical voting. For these reasons, we also explored the possibility to provide a fully tally-hiding protocol for ranked pairs, that we discuss below. In Ordinos [HHK+21], other variants, namely plain Condorcet, weak Condorcet, and the Smith, Copeland and Minimax methods are discussed. The plain, weak, Smith and Copeland methods are considered for the sole purpose of comparison, and are not popular variants of the Condorcet method since they do not have the resolvability property, which means that they do not address the Condorcet paradox entirely. In practice, it is still possible to use them, but in conjunction with a tie-break method (typically, a voting method named instant runoff voting). As for the Minimax variant, the Schulze method was originally designed to address some of its shortcomings, as it was too vulnerable to strategical voting. It is still possible to prefer the Minimax variant for its greater simplicity, and we mention that it is easy to evaluate it with our MPC toolbox, using the Min and the Max protocols.

The ranked pairs variant.

In the ranked pairs method, the adjacency matrix is seen as the adjacency matrix of a graph G. Then the method is divided into three steps. The first step is to sort the edges of G in decreasing order of weights. Then, the second step is to sequentially add those edges to a graph G' which initially has n_C vertices and no edge; however, we do not add the edges which create a cycle. Finally, as G' is an oriented graph without cycle, it can be seen as the graph of a partial order over the candidates. The sources of the graph are the winners.

Assuming that we have the encrypted adjacency matrix, an MPC version of the ranked pairs method goes as follows. First, to sort the edges, we can use the OddEvenMergeSort algorithm. We encode the edges with three ciphertexts, one for the source, one for the destination and one for the weight. The sources and the destination would each require $\log n_C$ ciphertexts, where n_C is the number of candidates, and the weight requires $\log n_V$ ciphertexts, where n_V is the number of valid ballots. Hence, applying the OddEvenMergeSort would require about ${n_C}^2 \log(n_C)^2(3 \log n_V + 2 \log n_C)$ CSZ in terms of computation and transcript size, and $2 \log(n_C)^2 \log n_V$ CSZ in terms of synchronization steps.

Then, the main procedure is to update an encrypted matrix $B_{i,j} = \mathrm{Enc}(b_{i,j})$, where $b_{i,j} = 1$ if there is a path from i to j, and 0 otherwise. Initially, B is a trivial encryption of the identity matrix. To add the edge (i, j), we compute $b'_{s,t}$ for all (s, t), as follows:

$$b'_{s,t} = b_{s,t} \vee (b_{s,i} \wedge b_{j,t}).$$

However, since i and j are encrypted, we first need to recover (the encrypted) $b_{s,i}$ and $b_{j,t}$ for all s, t. For this purpose, we can use equality tests and the Select protocol, which accounts for a total of $4{n_C}^2 \log n_C$CSZ in terms of computation and $2 \log n_C$CSZ in terms of synchronization steps. The edge will create a cycle if and only if $b'_{s,t} = b'_{t,s} = 1$ for some (s, t), hence we compute the encryption of the boolean

$$c = \vee_{s \neq t}(b'_{s,t} \wedge b'_{t,s}).$$

Finally, we can update $b_{i,j}$ using If and c. Overall, since we need to do this for the $O({n_C}^2)$ edges, it means that the cost of the second step is $O({n_C}^4 \log n_C\mathrm{CSZ})$ in terms of computation, and $O({n_C}^2 \log n_C)$ in terms of synchronization steps.

Finally, finding the sources of the graph can be done by exhaustive search on the final B, which cost $O({n_C}^2\mathrm{CSZ})$. The whole process can be performed in $O({n_C}^4 \log n_C)\mathrm{CSZ}$ in terms of computation and transcript size, and $O({n_C}^2 \log n_C)\mathrm{CSZ}$ in terms of synchronization steps. This means that computing ranked pairs in MPC is more expensive than computing the Schulze method, but still possible.

6.3 A solution for single transferable vote

Single transferable vote (STV) is a counting function which is widely used for politically binding elections, for instance in Australia, Canada, the United States and the United Kingdom. Its goal is to designate an electoral board or committee; in other words, given a number s of seats and several candidates, STV allows to elect s candidates. There are various versions of STV; nevertheless, the main idea is the following. First, each voter chooses a subset of the candidates: those are the candidates that they *like*, *i.e.* for which they would be happy if they are given a seat. In addition, they order those candidates according to their personal preferences; however, by contrast with Condorcet voting, the order is strict. For instance, if there are are four candidates represented by the number 1, 2, 3 and 4, Alice can vote $(1, 3)$ while Bob can vote $(4, 1, 2)$.

Once the ballots are cast, each is attributed a *weight*, initially 1. Then the counting process consists of several rounds: during each round, each ballot grants a number of votes (equal to the ballot's weight) to the first candidate mentioned in the ballot. If some candidates meet a certain quota q (which is fixed during the whole process), they are selected. The selected candidates keep q votes for themselves but, for each received ballot, they transfer some of its weight to the next candidate on the ballot, for instance with a transfer coefficient. If no candidate reach the quota, the ones with the lowest number of votes is eliminated and transfer all of their ballot to the next candidate in the ballot, but with the same weight. The process terminates when s candidates are elected, or when the number of candidates that remain is equal to the number of (still) available seats.

The various versions of STV may differ in several aspects: for instance, one or several candidates may be elected simultaneously. If several candidates are elected simultaneously, they may transfer some votes to each other, which causes some additional complications; see for instance the Meek method [Mee69] which proposes to solve this using a system of (polynomial) equations. Similarly, several candidates may be eliminated simultaneously. Finally, the exact rule for transferring votes does not necessarily imply a transfer coefficient. For instance, we can also choose a fraction of the ballots at random and transfer them entirely to the next candidate on the ballot.

6.3.1 Existing solutions for STV in electronic voting

Since STV demands the voters to give a permutation of some candidates, there are a lot of possible ballots and using a mixnet (*i.e.* revealing all the chosen ballots in the clear, but without leaking which voter cast which ballot) is not an ideal solution. Indeed, this allows Italian attacks. For this reason, it is important to provide a tally-hiding protocol for STV.

When we looked for academic solutions in the literature, we could not find many contributions. The most interesting one is called Shuffle-sum [BMN+09], which proposes a strategy where some information is revealed between each round of the STV algorithm, for instance the score (*i.e.* the number of votes) of all the candidates. The authors acknowledge that revealing the intermediates scores might be too much; in particular, they propose realistic scenarios where a coercer could successfully use this information. Consequently, they also propose a variant where the most crucial information are only leaked to the trustees: for an external observer, the only available side information is an approximation of the transfer coefficient, as well as the score of the selected candidate at each round. As discussed in Section 5.3.2, giving some information to the talliers only is certainly interesting, but does not protect the voters in the scenario where the coercer is a tallier.

Another solution is proposed in the technical report [WB08]. Just as Shuffle-sum, their solution implies to leak some information at each round, namely whether the round is a selection or an elimination, and the score of the selected candidate. In addition, we remark that their technique involves a very sequential first phase with a number of rounds of communications that is proportional to the number of ballots, which may be impractical.

A related contribution, which is not a solution for generic STV but for IRV (a specific case where there is only one seat to provide), can be found in [RCPT19]. In this proposal based on generic MPC, the score of all the candidates is revealed at each round.

In any case, none of the existing approach is *fully* tally hiding, and a careful analysis of whether the leaked information can or cannot be exploited is necessary. Thanks to our toolbox, we can design a completely leakage-free tally scheme.

6.3.2 Choosing one version of STV

Since there are many variants of STV, we had to choose one before providing an MPC solution for computing the corresponding tally function. We took guidance from Australian academics, and decided to consider an academic version of STV, where one candidate is elected or eliminated at each round. As for the quota, we define it as the Droop quota $q = \left\lfloor \frac{n_V}{s+1} \right\rfloor + 1$, where n_V is the number of valid ballots and s the number of seats. Finally, when a candidate is selected, we use the following rule. First, we denote v the sum of the weights of the ballots received by the selected candidate, and we compute the transfer coefficient $t = (v - q)/v$ (remark that since the candidate was selected, $v > q$). Then, for each ballot received by the candidate, we multiply the weight by t and transfer the ballot to the next candidate on the list. Finally, we remove the name of the selected candidate from every ballot.

Before applying this solution in MPC, however, we need a way to handle the fractions. Indeed, all along the STV algorithm, the weights of the ballots and the transfer coefficients are rational numbers that can be stored as pairs of integers. While this looks as the cleanest approach, we noticed that this leads to an exponential worse-case complexity. Indeed, the transfer coefficient t_i at a round i is a fraction whose height typically doubles at each round where a candidate is selected, and we get a complexity that is exponential in the number of seats. This observation is a major problem in an MPC setting where the worst-case complexity is also the best-cast complexity, since we want to hide every side-information. Also, outside any cryptographic consideration, we recovered the data of the 2019 election of the Legislative Council of New South Wales in Australia [NSW19], where there were 21 seats, 346 candidates and 3.5 millions ballots. With a basic implementation using Sagemath, we ran the academic STV algorithm on the publicly available ballots and remarked that the memory required to store all the fractions is exponential with respect to the number of selections, with a regression coefficient of $r^2 \geq 0.997$ (see Fig. 18).

Selections	Bytes
1	15139638
2	25708340
3	37498896
4	55311551
5	84089187
6	137366005

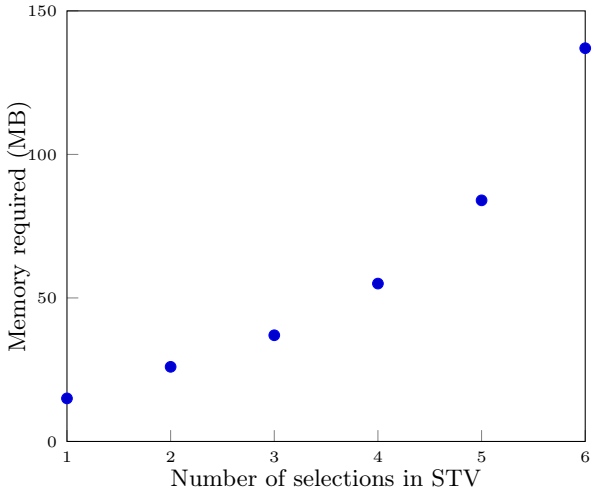


Figure 18: Memory required to run the STV algorithm, as a function of the number of selections

Since there can be up to 21 selections, this means that one may need up to 82 GB of central memory just to store all the fractions. In real elections, and due to the fact that elections were initially counted by hand, approximations of fractions are used instead. Therefore, we also represent fractions with a fix-point arithmetic, allowing r binary digits after the radix point.

Note that depending on the version of STV which is applied, the result might differ, see for

instance [GL17] for an analysis.

6.3.3 Ballots as lists of candidates

Now that the version of STV is fixed, we are almost ready to explain how to build a tally-hiding protocol for computing the result with no leakage. Before that, we must however explain what is the expected format for the ballots and how the voters can produce them. Just as for Condorcet-Schulze which was the subject of Section 6.2, we focus on the voter-side complexity, and tries to make the ballots as simple as possible. For this reason, we propose that the voters give a list of candidate $\boldsymbol{C_0}, \cdots, \boldsymbol{C_{n_C}}$, where n_C is the number of candidates. So that the voter does not reveal how many candidates are ranked in the ballot, the ballot always consists of $n_C + 1$ candidate. However, there is a *sink* candidate, candidate number 0, that represents the end of the ballot and that cannot be selected nor eliminated. All the candidates that are ranked after the candidate 0 are not actually a part of the ballot and are here for the sole purpose of hiding the size of the ballot. Since we want to be able to use our MPC toolbox on the ballots, we need the candidates to be bitwise encrypted. For this reason, for all i, $\boldsymbol{C_i}$ consists of $\ell = \lceil \log(n_C + 1) \rceil$ encryptions of a bit. Also, we require that the corresponding bitstrings encode a permutation of $[0, n_C]$.

To produce a ballot and to prove its validity, the voter shuffles the initial ballot $B_{\text{init}} = (0^{\text{bits}}, \cdots, {n_C}^{\text{bits}})$, which consists of n_C trivial bitwise encryptions of the same size ℓ; *i.e.* 0^{bits} is $E_0, \cdots, E_0$, 1^{bits} is $E_1, E_0, \cdots, E_0$ and so on. To prove that the ballot is well-formed, the voter provides a proof of a shuffle, using ShuffleRow (see Algorithm 19).

6.3.4 A tally-hiding protocol for academic STV

We are now ready to present our tally-hiding protocol for STV. On this occasion, we recall that we use the following parameters:

- n_C is the number of candidate (an additional artificial candidate is considered, not included in this number);
- n_V is the number of valid ballots;
- $m = \lceil \log(n_V + 1) \rceil$ is the maximum bitsize of the number of voters who ranked first a given candidate;
- s is the number of seats to attribute;
- $q = \left\lfloor \frac{n_V}{s+1} \right\rfloor + 1$ is a quota above which a candidate is selected;
- r is the precision for the fractions, *i.e.* the number of binary places after the radix point;
- $\ell = \lceil \log(n_C + 1) \rceil$ is the bitsize used to represent a candidate, numbered from 0 to n_C;
- E_0 and E_1 are trivial encryptions of 0 or 1;
- For $i \in [0, n_C]$, i^{bits} consists of the ℓ trivial encryptions of the bits of i.

The first task of the talliers is to initialize a data structure as follows:

- H is the *hopeful* vector. It contains n_C encryptions of bits (initially E_1) which state whether a candidate can still be selected or not.

- W is the *winner* vector. It contains n_C encryptions of bits (initially E_0) which state whether a candidate has been selected or not.
- S is the score vector. It contains n_C bitwise encrypted integers of size $m + r$; it contains the sum of the weights of the ballots received by each candidate.
- V is the value matrix. For all $i \in [1, n_V]$ V_i is a bitwise encrypted integer of size $r + 1$, initially $(E_0, \cdots, E_1)$; the r less significant bits (on the left) represent the r binary places.
- B is the ballot matrix. For all $i \in [1, n_V]$, $B_i[0], \cdots, B_i[n_C]$ are bitwise encrypted integers of size ℓ that represent a candidate.

The initialization only requires to read the ballots submitted by the voters and can be considered free compared to the cost of the remaining of the protocol. Then the talliers will loop $n_C - 1$ iterations of the following operations:

1. **Finished?** (Algorithm 83.) From the candidate data structure, compute the number of candidates (apart from candidate 0) that got a seat or are still hopeful. If this is equal to the number of available seats s, then mark as selected all the hopeful candidates. Note that during the first iterations of the loop, this test is not necessary.
2. **Count votes.** (Algorithm 84.) For each ballot B_i, take the candidate ranked first $Bi[0]$, and add the weight V_i of the ballot to the score S_i of this candidate.
3. **Search min-max.** (Algorithm 85.) Compute i and j the indexes of the candidates that have the maximum and minimum score. If the score s_i of candidate i is larger than the quota, set the variables a to 0, c to i and t to $(s_i - q)/s_i$. Otherwise, the candidate j will be eliminated and set a to 1, c to j and t to 1.
4. **Select, delete, transfer.** (Algorithm 86.) Mark the candidate c as no longer hopeful, and c as a winner if a is 0. Also, for all ballot, remove the candidate c. This is done in one pass over the list of preferences of each ballot. For each ballot, if c was in the first position, multiply its weight by the transfer value t.

After this, the vector W is decrypted to reveal which of the candidates won the election.

In theory, the STV process stops when s candidates have been selected or when the number of candidates that remain is equal to the number of (still) available seats. However, we do not want to reveal *when* the process finished, as this would constitute a side-information. For this reason, we will let it continue by computing some additional rounds, even after reaching the point when it should have stopped. After that s candidates are selected, adding some additional rounds will not modify the result as it is not possible for $s + 1$ or more candidates to reach the quota. Consequently, no subsequent selection would occur and W will no longer be modified. However, if the number of candidates that remain is equal to the number of remaining seats, adding an additional round may lead to an elimination if no candidate reach the quota, so it is important to select all the candidates right away, before they get eliminated. Since a candidate is either selected or eliminated each round, the round index i is such that the number of remaining candidates is equal to $n_C - i$. Moreover, the number of remaining seats is $s - N$, where N is the number of selected candidates. Therefore we need to test whether $n_C - i = s - N$, in which case we must select all the remaining candidates. Note that since $n_C - i > 0$, this condition is exclusive with $N = s$, so that we do not have to worry about this causing some extra candidate being selected if s were already selected. Also, this condition can only occur once, since afterwards

Algorithm 83: Finished

Requires: s, the number of seats
n_C, the number of candidates
i, the round index (0 for the first iteration)
Inputs: H, the encrypted hopeful vector
W, the encrypted winners vector
Outputs: H, W

$\boldsymbol{N} \longleftarrow \text{Aggreg}(W_1, \cdots, W_{n_C})$ (* number of winners *);
$F \longleftarrow \text{EqKnown}(\boldsymbol{N}, s - n_C + i)$ (* $f = 1$ if the process finished this round *);
for $i = 1$ *to* n_C *(in parallel)* **do**
  $T_i \longleftarrow \text{CSZ}(H_i, \text{Not}(F))$ (* sets as 0 if $f = 1$ *) ;
  $O_i \longleftarrow H_i\text{Not}(T_i)$ (* $h_i \wedge f$ *);
  $W_i \longleftarrow \text{Or}(W_i, O_i)$ (* sets as winners the hopeful *);
  $H_i \longleftarrow T_i$;
return H, W

Algorithm 84: CountVotes

Requires: n_V, the number of valid ballots
n_C, the number of candidates
Inputs: B, the matrix of the encrypted ballots
V, the encrypted vector of the value of each ballot
Outputs: S, the encrypted vector of the scores of the candidates

for $i = 1$ *to* n_V *(in parallel)* **do**
  for $j = 1$ *to* n_C *(in parallel)* **do**
    $T_{i,j} \longleftarrow \text{EqKnown}(\boldsymbol{B_i}[\boldsymbol{0}], j)$;
    $\boldsymbol{C_{i,j}} \longleftarrow \text{CSZ}(\boldsymbol{V_i}, T_{i,j})$;
for $j = 1$ *to* n_C *(in parallel)* **do**
  $S_j \longleftarrow \text{Add}(\boldsymbol{C_{1,j}}, \cdots, \boldsymbol{C_{n_V,j}})$;
return S;

$N = s$. Finally, since we mark every candidate as no longer hopeful when the condition is met, no further modification of W can occur during the subsequent rounds.

In the CountVote protocol, we use a tree-based parallelization protocol to add n_V different values, as we already did many times for other associative laws in Chapter 5; see for instance Algorithm 51.

Algorithm 85: SearchMinMax

Requires: r, the precision for the division
Inputs: S, the encrypted vector of the scores of the candidates
Outputs: A, an encryption of 0 if we have a selection, of 1 if we have an elimination
$\boldsymbol{C}$,a bitwise encryption of the index of the candidate to select or eliminate
$\boldsymbol{T}$, a bitwise encryption of the transfer coefficient

1 $\_, \boldsymbol{I}, \boldsymbol{Z}, \boldsymbol{J} \longleftarrow \text{MinMax}(\boldsymbol{S_1}, \cdots, \boldsymbol{S_{n_C}})$;
2 $\boldsymbol{\Delta}, A \longleftarrow \text{SubKnown}(\boldsymbol{Z}, q)$;
3 $\boldsymbol{T} \longleftarrow \text{CSZ}(\text{Div}(\boldsymbol{\Delta}, \boldsymbol{Z}, r), \text{Not}(A)) || A$;
4 $\boldsymbol{C} \longleftarrow \text{If}(A, \boldsymbol{I}, \boldsymbol{J})$;
5 **return** $A, \boldsymbol{C}, \boldsymbol{T}$

In the SearchMinMax protocol, we first compute the maximum score z and the index i, j of the candidates that have the minimum and the maximum score. Then, if $z \geq q$, we set a to 0 and compute $t = (z - q)/z$. For this purpose, we use the Div algorithm which returns the r first binary places and we add a 0 as the most significant bit. If $z < q$, we set a to 1 and set all the binary places of t to 0, but add 1 as the most significant bit.

Finally, the SelectDeleteTransfer protocol is a naive transposition of the cleartext algorithm in the MPC toolbox.

6.3.5 Complexity analysis

To analyze the complexity of our protocol, we give in Table 15 the complexity of each sub-protocol. Apart from the Finished sub-protocol which is only called s times (the number of seats), every sub-protocol is called $n_C - 1$ times exactly, hence the bottom line of the table. Note that the Finished protocol can be computed in parallel with the CountVotes and SearchMinMax protocols. Therefore, we do not include its communication cost in the total.

The intrinsic cost of STV is that of $\Omega(n_V {n_C}^2)$ operations, since there are n_C rounds, and each of them modifies n_V ballots that contains n_C candidates. Therefore, it seems that the computational cost of our MPC approach is reasonable when compared to the best possible complexity. However, the number of synchronization steps would be quadratic in r, n_C and $\log n_V$, which could be problematic since there can be hundreds of candidates. Thankfully, we provide some computation / communication trade-offs in our toolbox. Hence, it is possible to modify the naive transposition of STV using more communication-efficient primitives. For this purpose, we can proceed as follows.

- In SearchMinMax, modify the MinMax protocol by using the communication-efficient variant, based on CLt;
- In SearchMinMax, use UFCSub instead of Sub to compute the subtractions;
- In SearchMinMax, modify the division by replacing the Sub protocol by UFCSub;

Algorithm 86: SelectDeleteTransfer

Requires: n_V, the number of valid ballots
n_C, the number of candidates

Inputs: A, an encryption of 0 if we have a selection, of 1 if we have an elimination
$\boldsymbol{C}$,a bitwise encryption of the index of the candidate to select or eliminate
$\boldsymbol{T}$, a bitwise encryption of the transfer coefficient
H, the encrypted hopeful vector
W, the encrypted winners vector
B, the matrix of the encrypted ballots
V, the encrypted vector of the value of each ballot

Outputs: H, W, B, V

for $i = 1$ *to* n_C *(in parallel)* **do**
  $T \longleftarrow \text{EqKnown}(\boldsymbol{C}, i)$;
  $H_i \longleftarrow \text{CSZ}(H_i, \text{Not}(T))$;
  $T \longleftarrow \text{And}(T, \text{Not}(A))$;
  $W_i \longleftarrow \text{Or}(W_i, T)$;
for $i = 1$ *to* n_V *(in parallel)* **do**
  $A \longleftarrow \text{Eq}(\boldsymbol{B_i}[\boldsymbol{0}], \boldsymbol{C})$;
  $F \longleftarrow A$ (* f indicates whether the candidate was found in the list *);
  for $j = 0$ *to* $n_C - 1$ **do**
    $\boldsymbol{B_i}[\boldsymbol{j}] \longleftarrow \text{If}(F, \boldsymbol{B_i}[\boldsymbol{j+1}], \boldsymbol{B_i}[\boldsymbol{j}])$;
    $F \longleftarrow \text{Or}(F, \text{Eq}(\boldsymbol{B_i}[\boldsymbol{j+1}], \boldsymbol{C})$;
  $\boldsymbol{M} \longleftarrow \text{Mult}(\boldsymbol{V_i}, \boldsymbol{T})$ (* we only keep the $m + r$ msb *);
  $\boldsymbol{V_i} \longleftarrow \text{If}(A, \boldsymbol{M}, \boldsymbol{V_i})$;
return H, W, B, V;

Table 15: Leading terms of the cost of the various sub-protocols for STV (naive approach); n_C is the number of candidates, n_V is the number of valid ballots, r is the number of binary places after the radix point; the cost are expressed in the number of CSZ required; the transcript size can be deduced directly from the second column.

Protocol	Computations	Communications
Finished	$5n_C$	$\frac{1}{2}\log(n_C)^2$
CountVotes	$n_V n_C(3r + 3\log n_V + \log n_C)$	$2\log n_V(r + \log n_V)$
SearchMinMax	$(6n_C + 3r)(r + \log n_V) + 2n_C \log n_C$	$2(\log n_V + r)(r + \log n_C)$
SelDelTrans	$n_V(3n_C \log n_C + 3r(r + \log n_V))$	$n_C \log\log n_C + 2r(r + \log n_V)$
Total	$n_V {n_C}^2(3r + 3\log n_V + 4\log n_C)$	$2n_C(r + \log n_V)(2r + \log n_V + \log n_C)$ $+{n_C}^2 \log\log n_C$

- In SelectDeleteTransfer, modify the multiplication by replacing the Add protocol by UFCAdd;
- In SelectDelateTransfer, replace the innermost for loop using an UFC to computes all the values of F in parallel (see Algorithm 87).

Algorithm 87: SelectDeleteTransfer (communication efficient)

Requires: n_V, the number of valid ballots
n_C, the number of candidates
Inputs: A, an encryption of 0 if we have a selection, of 1 if we have an elimination
$\boldsymbol{C}$,a bitwise encryption of the index of the candidate to select or eliminate
$\boldsymbol{T}$, a bitwise encryption of the transfer coefficient
H, the encrypted hopeful vector
W, the encrypted winners vector
B, the matrix of the encrypted ballots
V, the encrypted vector of the value of each ballot
Outputs: H, W, B, V

for $i = 1$ *to* n_C *(in parallel)* **do**
  $T \longleftarrow \mathrm{EqKnown}(\boldsymbol{C}, i)$;
  $H_i \longleftarrow \mathrm{CSZ}(H_i, \mathrm{Not}(T))$;
  $T \longleftarrow \mathrm{And}(T, \mathrm{Not}(A))$;
  $W_i \longleftarrow \mathrm{Or}(W_i, T)$;
for $i = 1$ *to* n_V *(in parallel)* **do**
  for $j = 0$ to $n_C - 1$ (in parallel) **do** $F_i \longleftarrow \mathrm{Eq}(\boldsymbol{B_i}[\boldsymbol{j}], \boldsymbol{C})$;
  $F_0, \cdots, F_{n_C-1} \longleftarrow \mathrm{UFC}(\mathrm{Or}, F_0, \cdots, F_{n_C-1})$;
  for $j = 0$ to $n_C - 1$ (in parallel) **do** $\boldsymbol{B'_i}[\boldsymbol{j}] \longleftarrow \mathrm{If}(F_j, \boldsymbol{B_i}[\boldsymbol{j+1}], \boldsymbol{B_i}[\boldsymbol{j}])$;
  $\boldsymbol{M} \longleftarrow \mathrm{Mult}(\boldsymbol{V_i}, \boldsymbol{T})$ (* we only keep the $m + r$ msb *);
  $\boldsymbol{V_i} \longleftarrow \mathrm{If}(F_0, \boldsymbol{M}, \boldsymbol{V_i})$;
return H, W, B, V;

This gives a more advanced approach, where all the quadratic terms in the number of synchronization steps are removed. With this approach, the number of synchronization steps become reasonable, at the cost of a small constant in the computation cost (see Table 16).

6.4 Majority Judgment

Majority Judgment is a counting method which has been notably used in the primary election for the 2022 presidential in France, where about 400 000 voters participated [gua22]. It is defined in [BL10], and is often cited by French researchers as a strategy-resistant alternative to the current voting system [mie]. In this method, the voter must give a grade to each candidate, for instance Excellent, Good, Medium, Bad or Reject. The number of possible grades is supposed to be small, typically 5 to 7, and we denote it n_G. Each grade is represented by a number, from 1 to n_G. However, the tradition in MJ is to use a reversed ordering (*i.e.* 1 is a better / higher grade than 2). Since each voter has to grade each candidate, each candidate ends up with a list of n_V grades, where n_V is the number of voters who did not abstain or vote blank. For simplicity, we assume that the lists are sorted in decreasing order (highest grades first). Thus,

Table 16: Leading terms of the cost of the various sub-protocols for STV (advanced approach); n_C is the number of candidates, n_V is the number of valid ballots; ; for any expression expr, we denote $(\text{expr})^*$ the expression $\text{expr}\log\text{expr}$; the cost are expressed in the number of CSZ required; the transcript size can be deduced directly from the second column.

Protocol	Computations	Communications
Finished	$6n_C$	$\frac{1}{2}\log(n_C)^2$
CountVotes	$\frac{3}{2}n_V n_C(r+\log n_V)^*$	$2\log n_V \log(r+\log n_V)$
SearchMinMax	$(r+\log n_V)(10n_C+\frac{3}{2}r\log(r+\log n_V))$	$2(r+\log n_C)^*$
SelectDeleteTransfer	$n_V(\frac{7}{2}{n_C}^*+\frac{3}{2}r(r+\log n_V)^*)$	$2r\log(r+\log n_V)$
Total	$n_V {n_C}^2\left(\frac{3}{2}(r+\log n_V)^*+\frac{7}{2}\log n_C\right)$	$2n_C\log n_V\log(r+\log n_V)$ $+2n_C(r+\log n_C)^*$ $+2n_C r\log(r+\log n_V)$

we consider that each candidate has a sorted n_V-tuple. Note that two n_V-tuples are equal if and only if the candidates received exactly the same number of each grade. Given a sorted n_V-tuple $u_1,\cdots,u_{n_V}$, the *median* of u is $\text{med}(u)=u_{\lceil n_V/2\rceil}$, and we denote $\hat{u}$ the (n_V-1)-tuple $u_1,\cdots,u_{\lceil n/2\rceil-1},u_{\lceil n/2\rceil+1},\cdots,u_{n_V}$; that is, the tuple u in which the median element has been removed. Finally, the relation $\leq_{\text{maj}}$ is defines as follows, where $<$ stands for the grade-wise comparison (which is the opposite of the natural comparison of integers).

Definition 19. *Let u and v be two n-tuples of grades, sorted in decreasing order. If $n=1$, $u<_{\text{maj}}v$ if $u_1<v_1$. Otherwise, $u<_{\text{maj}}v$ if one of the following conditions holds:*

- $\text{med}(u)<\text{med}(v)$
- $\text{med}(u)=\text{med}(v)$ *and* $\hat{u}<_{\text{maj}}\hat{v}$.

Finally, $u\leq_{\text{maj}}v$ if $u=v$ or $u<_{\text{maj}}v$.

It is possible to show that $\leq_{\text{maj}}$ is a total order, and that the Majority Judgment declares as a winner any candidate whose grades form a maximal n_V-tuple (once sorted) for $\leq_{\text{maj}}$.

6.4.1 Existing approaches for computing the Majority Judgment

Interestingly, it is possible to adapt the Helios protocol to cover the Majority Judgment. For this purpose, one can define the grade matrix G as a matrix of n_C rows and n_G columns, where n_C is the number of candidates and n_G is the number of grades. Then, giving a grade to each candidate is the same as choosing a single cell for each row of the matrix. Hence, a voter can encrypt a matrix of $n_C n_G$ bits, and use a standard ZKP to prove that the ballot is well-formed (*i.e.* that each ciphertext is indeed an encryption of 0 or 1 and that each row contains exactly one 1; to allow blank voting, it may also be authorized to encrypt a matrix of zeros). Once every ballot has been submitted, a homomorphic tally can be used to reveal the aggregated grade matrix, which states how many voters gave each grade to each candidate. From this, the result of the Majority Judgment can be deduced.

In [CPST18], an MPC implementation based on the Paillier encryption scheme was proposed to compute the majority judgment in a fully tally-hiding fashion. For this purpose, they used a heuristic from [BL10], which is called the *majority gauge*. It is known that this heuristic is sound: if a winner can be determined with this approach, it is indeed a MJ winner. However,

Table 17: Estimated probability that the majority gauge fails to determine the MJ winner(s).

# voters	10	100	1000
uniform distribution over 5 candidates	0.384	0.220	0.080

it may also fail to conclude. An experiment run in [BL10] on real ballots of a political election with 12 candidates is reassuring: the simplified approach fails only with probability $0,001$ for an election with 100 voters. However, this is due to the fact that in this particular election, there was a high correlation between candidates: if a voter likes a candidate, another candidate from a similar political party is also likely to be liked. If there are less candidates and if the distribution of votes is uniform, then the probability of failure raises up to 22%, as shown in Table 17. In any case, the approach of [CPST18] leaks more information about the ballots than just the result, since it reveals whether the result can be determined with the majority gauge.

In order to fix the flaw of [CPST18] and to compare the performance of our toolbox with that of a similar approach that relies the Paillier encryption scheme, we developed a strategy for full tally-hiding in the MJ setting.

6.4.2 A new algorithm for cleartext Majority Judgment

One of the main reasons why [CPST18] used the majority gauge instead of the real MJ algorithm is because the latter is quite complex: if n_V is the number of voters, comparing two candidates can take up to n_V comparisons between grades, and is a highly sequential process. The book [BL10] does provide an alternative algorithm to compute the result faster, but the algorithm is also a bit complex and not easy to adapt in MPC. For this reason, we propose a new algorithm for computing the result of the Majority Judgment (see Algorithm 88, which requires $O(n_C n_G)$ comparisons between grades. Instead of comparing the medians one by one, this algorithm uses the fact that the sequences formed by the successive medians (that we call the *median sequence*) contains batches of identical elements. Therefore, we can speed up the comparison by comparing the median sequence batch by batch instead of element by element. We prove the correctness of our algorithm in Theorem 4, which is the subject of the remaining of this subsection.

Theorem 4. *Algorithm 88 returns the set of maxima according to* $\leq_{\text{maj}}$ *in* $O(n_C n_G)$ *comparisons between grades, where* n_C *is the number of candidates and* n_G *the number of grades.*

The proof of this theorem is available in Appendix C.

6.4.3 Adaptation to the Paillier setting

Before giving a protocol in the ElGamal setting, we propose to perform a first iteration to adapt Algorithm 88 to the MPC setting, but in the context of Paillier. This allows not to address all the difficulties at once. Since we only focus on the tally phase and since obtaining (an element-wise encryption of) the aggregated matrix from the ballots is easy in the Paillier setting, we consider that (an element-wise encryption of) the aggregated matrix is available. We first rewrite the algorithm into Algorithm 91 and prove that the new algorithm is equivalent to Algorithm 88. Using our MPC toolbox, it is easy to implement Algorithm 91 in MPC (see Algorithm 93). This subsection details all the steps that we used to adapt the MJ algorithm in the Paillier setting, and proves the correctness of the resulting MPC protocol. As usual, we use the notation E_m to

Algorithm 88: Majority Judgment

Requires: n_C, the number of candidates
n_G, the number of grades
n_V, the number of voters

Inputs: a, the aggregated grade matrix s.t.
$a[i,j]$ is the number of voters who gave the rank j to the candidate i

$m \longleftarrow \max\{m_i \mid m_i \text{ is the median of candidate } i\}$;
$C \longleftarrow \{i \mid m_i = m\}$;
$I^- \longleftarrow 1$; $I^+ \longleftarrow 1$;
$s \longleftarrow 1$;
for $i \in C$ **do**
  $p_i \longleftarrow \sum_{j=1}^{m-1} a_{i,j}$;
  $q_i \longleftarrow \sum_{j=l+1}^{n_G} a_{i,j}$;
  $m_i^- \longleftarrow \lfloor n_V/2 \rfloor - p_i$;
  $m_i^+ \longleftarrow \lfloor n_V/2 \rfloor - q_i$;
while $|C| > 1$ ***and*** $s \neq 0$ **do**
  for $i \in C$ **do**
    if $m_i^- \leq m_i^+$ **then** $s_i \longleftarrow p_i$;
    else $s_i \longleftarrow -q_i$;
  $s \longleftarrow \max\{s_i \mid i \in C\}$;
  $C \longleftarrow \{i \in C \mid s_i = s\}$;
  if $s \geq 0$ **then**
    for $i \in C$ **do**
      $m_i^+ \longleftarrow m_i^+ - m_i^-$;
      $m_i^- \longleftarrow a_{i,m-I^-}$;
      $p_i \longleftarrow p_i - a_i i, m - I^-$;
    $I^- \longleftarrow I^- + 1$;
  else
    for $i \in C$ **do**
      $m_i^- \longleftarrow m_i^- - m_i^+$;
      $m_i^- \longleftarrow a_{i,m+I^+}$;
      $q_i \longleftarrow q_i - a_{i,m+I^+}$;
    $I^+ \longleftarrow I^+ + 1$;
return C;

denote a trivial encryption of m. However, as m may be a complex expression, we may also use the notation $\mathrm{Enc}(m)$.

We first provide Algorithm 89 which returns the grade vector as defined in [CPST18]. It is a (term-by-term) encryption of g such that $g_j = 1$ if j is strictly better than the best median m, and $g_j = 0$ otherwise. The idea of this algorithm is that, for all candidate i and grade j, j is strictly better than the best median if and only if the number of grades better than j is strictly lower than half the number of grades. This translates into the formula $2\sum_{\ell=1}^{j} a_{i,\ell} < \sum_{\ell=1}^{n_G} a_{i,\ell}$, which allows to compute $c_{i,j}$ for all (i,j), where $c_{i,j} = 1$ if j is strictly better than i's median. To deduce the grade vector, we compute the logical conjunction column by column.

Algorithm 89: Grade (Paillier setting)

Requires: n_G, the number of grades
n_C, the number of candidates
Inputs: A, the encrypted aggregated grade matrix
Outputs: G, the encrypted grade vector s.t. for all j, G_j is an encryption of 1 if j is strictly better than the best median, of 0 otherwise.

1 $V \longleftarrow \prod_{j=1}^{n_G} A_{1,j}$;
2 **for** $i = 1$ *to* n_C **do**
3 **for** $j = 1$ *to* n_G **do**
4 $B \longleftarrow \left(\prod_{\ell=1}^{j} A_{i,\ell}\right)^2$;
5 $C_{i,j} \longleftarrow \mathrm{Not}(\mathrm{GTH}(B, V))$
6 **for** $j = 1$ *to* n_G **do**
7 $G_j \longleftarrow C_{1,j}$;
8 **for** $i = 2$ *to* k **do**
9 $G_j \longleftarrow \mathrm{Mul}(G_j, C_{i,j})$;
10 Return G

Once the grade vector is computed, we can initialize p_i, m_i^-, m_i^+ and q_i with Algorithm 90, which is adapted from [CPST18].

The idea is that p_i can be obtained from G thanks to $p_i = \sum_{j=1}^{n_G} a_{i,j} g_j$ while q_i can be obtained similarly with a right shift of G's negation. Indeed, $\mathrm{Not}(G)$ is the vector of encryptions of 1 if j is worse than the best median, of 0 otherwise. Its right shift is therefore encryptions of 1 if j is strictly worse than the best median, of 0 otherwise.

At this point, we remark that we can replace C as defined in line 2 of Algorithm 88 by the whole set of candidates, this without affecting the result, (see Lemma 11). In what follows, we call Algorithm 88.11 the Algorithm 88 in which this transformation has been done.

Lemma 11. *In Algorithm 88, replacing line 2 by "$C \longleftarrow [1, n_C]$" will not alter the output.*

Proof. We show that after the first iteration of the loop, the C sets of both algorithms are the same, which shows that invariants from Lemma 18 are verified at the beginning of the second iteration of the loop, if any (if not the output is correct as well since the sets are the same).

Algorithm 90: InitD (Paillier setting)

Requires: n_V, the number of voters
Inputs: $(A)_{i,j}$, the encrypted aggregated grade matrix
G, the encrypted grade vector
Outputs: P, M^-, M^+, Q where, for all i,

- P_i is an encryption of p_i, the number of grades received by i which are strictly better than the best median,
- M_i^- is an encryption of $\lfloor n_V/2 \rfloor - p_i$,
- Q_i is an encryption of the number q_i of grades received by i which are strictly worse than the best median,
- M_i^+ is an encryption of $\lfloor n_V/2 \rfloor - q_i$.

1 **for** $i = 1$ *to* n_C **do**
2 $\quad P_i \longleftarrow \prod_{j=1}^{n_G} \mathrm{Mul}(A_{i,j}, Gj)$;
3 $\quad M_i^- \longleftarrow \mathrm{Enc}(\lfloor n/2 \rfloor)/P_i$;
4 $\quad Q_i \longleftarrow \prod_{j=2}^{n_G} \mathrm{Mul}(A_{i,j}, \mathrm{Not}(G_{j-1}))$;
5 $\quad M_i^+ \longleftarrow \mathrm{Enc}(\lfloor n_V/2 \rfloor)/Q_i$;
6 **return** P, M^-, M^+, Q;

Let m be the best median, and a and b be two candidates such that $\mathrm{med}(b) < \mathrm{med}(a) = m$. For all i, after line 7 in both algorithms, p_i is the number of grades strictly better than m received by candidate i while q_i is the number of grades strictly worse than m received by candidate i. By definition of the median, we have $q_a \leq \lfloor n_V/2 \rfloor$. On the other hand, $p_b \leq \lfloor n_V/2 \rfloor < q_b$. But after line 9, we have $m_i^- + p_i = m_i^+ + q_i = \lfloor n_V/2 \rfloor$ for all i so $m_b^- > m_b^+$ and $S_b = -q_b$ after line 13. As $S_a \in \{p_a, -q_a\}$ with $p_a \geq -q_a \geq -\lfloor n_V/2 \rfloor > -q_b$, we have $S_b < S_a$. Therefore b is discarded from C at line 15. □

Lemma 11 allows to initialize p_i, m_i^-, m_i^+ and q_i for all candidate i with no care of whether i's median is m or not. Now we explain how to run the while loop in MPC without revealing the number of iterations, nor the number of candidates which remain at any given point (see Lemma 12).

Lemma 12. *In Algorithm 88.11, we can replace the while loop by a for loop with n_G iterations, without affecting the result. Moreover, invariants from Lemma 18 are still preserved.*

Proof. Following the proof of Lemma 18, we remark that the proof does not depend on the number of iterations, so the loop invariants are preserved even if additional iterations are performed. Since the number of iterations is at most n_G as explained in the proof of Theorem 4, this concludes the proof. □

In what follows, we denote Algorithm 88.12 the Algorithm 88.11 in which the while loop is replaced by a for loop with n_G iterations.

To encode C, we use its indicator (which we also denote C). To show the implied modification, we explicitly give Algorithm 91, where the transformations induced by Lemmas 11 and 12 have been made.

To prove its correctness, we give the following lemma.

Lemma 13. *In Algorithm 91, c is the indicator of C from Algorithm 88.12.*

Proof. We verify that this property holds as a loop invariant.

Initialization. Before the first loop iteration, we have $c_i = 1$ for all $i \in [1, n_C]$ and $C = [1, n_C]$ so c is C's indicator.

Heredity. Suppose that before the j^{th} iteration in Algorithm 91, c is the indicator of the set C such as before the j^{th} iteration in Algorithm 88.12. Then for $i \in C$, $c_i = 1$ so s_i is the same in both algorithms. On the other hand, for $i \notin C$, $c_i = 0$ so $s_i = -n$ in Algorithm 91. By Lemma 11, after the first loop iteration in Algorithm 88.11, C only contains candidates of median m. They therefore have at least a grade equal to m, so for all $i \in C$, $q_i \leq n_V - 1 < n_V$ after the first iteration. Since q_i can only decrease, we always have $p_i \geq -q_i > -n_V$ for $i \in C$, hence $s_i > -n_V$. Therefore, for $i \in C$ and $j \notin C$, $s_i > s_j$. This is also true in Algorithm 88.12, so s is the same in both algorithms after line 15. □

Now we explain how to get $a_{i,m-I^-}$ and $a_{i,m+I^+}$ without revealing $m - I^-$ et $m + I^+$. We use two vectors L and R of size n_G such that L_j is an encryption of 1 if $j = m - I^-$, of 0 otherwise, while R_j is an encryption of 1 if $j = m + I^+$, of 0 otherwise. This way $a_{i,m-I^-}$ and $a_{i,m+I^+}$ can be obtained with Select. To initialize L and R, we use Algorithm 92 which uses the grade matrix g such that $g_j = 1$ if $j < m$, where m is the best median, and $g_j = 0$ otherwise. The idea is that $m - 1$ is the last index for which $g_j = 1$, so that $l_j = g_j - g_{j+1}$. Note that an initialization of R is obtained from L, with two right shifts. The only difficulty is when the best median is equal to the best possible grade, in which case g and l are null, while $r_2 = 1$. In any other case, $g_0 = 1$ and $r_2 = 0$, so we have $r_2 = 1 - g_0$.

In order to increment I^- and I^+, we use the conditional left and right shift, which are protocols of the toolbox. Note that we always have $L_{n_G} = \text{Enc}(0)$ while $R_{n_G} = \text{Enc}(0)$, so L and R can be processed as vectors of $n_G - 1$ ciphertexts.

The complete procedure is given in Algorithm 93, whose correctness is the claim of Theorem 5. In this algorithm, we add the constant n_V (the number of voters) to the candidates' scores at line 15, so that each integers to be compared are non-negative. The comparison requires therefore one additional bit but only for the first loop iteration. In the remaining iterations, we have $q_i \leq \lfloor n_V/2 \rfloor$ so that we can add $\lfloor n_V/2 \rfloor$ instead of n_V. Since $p_i \leq \lfloor n_V/2 \rfloor$, we no longer need an extra bit. For simplicity, we did not explicitly write this optimization in Algorithm 93. Another notable difference compared to Algorithm 91 is that instead of computing $m_i^- \leq m_i^+$, we compute $p_i \geq q_i$ (which is equivalent by invariant 1 from Lemma 18) since p_i and q_i are non-negative, while m_i^+ and m_i^- could be negative during the first loop iteration.

Theorem 5. *Algorithm 93 is correct.*

Proof. See Lemmas 11, 12, 13 and 18, as well as Lemma 14 below. □

Lemma 14. *In Algorithm 93, after the i^{th} loop iteration, L and R are such that L_j is an encryption of 1 if $j = m - I^-$ and 0 otherwise, and R_j is an encryption 1 if $j = m + I^+$, of 0 otherwise.*

Algorithm 91: MJ; with a fixed number of loops, and an indicator instead of a set.

Requires: n_C, the number of candidates
n_G, the number of grades
n_V, the number of voters
Inputs: a, the aggregated matrix.
Outputs: c, the indicator of the set of MJ winners.

Let m be the best median among all candidates;
Let c such that for $i \in [1, n_C]$, $c_i = 1$;
Let $I^- = 1$ and $I^+ = 1$ be counters;
for $i = 1$ *to* n_C **do**
    $p_i \longleftarrow \sum_{j=1}^{m-1} a_{i,j}$; $q_i \longleftarrow \sum_{j=m+1}^{n_G} a_{i,j}$;
    $m_i^- = \lfloor \frac{n_V}{2} \rfloor - p_i$; $m_i^+ = \lfloor \frac{n_V}{2} \rfloor - q_i$;
for $j = 1$ *to* n_G **do**
    for $i = 1$ *to* n_V **do**
        if $m_i^- \leq m_i^+$ **then** $s_i \longleftarrow p_i$;
        else $s_i \longleftarrow -q_i$;
    if $c_i = 0$ **then**
        $s_i \longleftarrow -n_V$ (* Already eliminated candidates are given a fake score *)
    $s \longleftarrow \max\{s_i \mid i \in [1, n_C]\}$;
    for $i = 1$ *to* n_C **do**
        if $s_i \neq s$ **then** $c_i \longleftarrow 0$;
    if $s \geq 0$ **then**
        for $i = 1$ *to* n_C **do**
            $m_i^+ \longleftarrow m_i^+ - m_i^-$;
            $m_i^- \longleftarrow a_{i,m-I^-}$;
            $p_i \longleftarrow p_i - a_{i,m-I^-}$;
        $I^- \longleftarrow I^- + 1$;
    else
        for $i = 1$ *to* n_C **do**
            $m_i^- \longleftarrow m_i^- - m_i^+$;
            $m_i^+ \longleftarrow a_{i,m+I^+}$;
            $q_i \longleftarrow q_i - a_{i,m+I^+}$
        $I^+ \longleftarrow I^+ + 1$;
return c;

Algorithm 92: InitP

Requires: n_G, the number of grades
Inputs: G, the grade matrix
Outputs: L, R, two vectors such that, for all i,

- L_i is an encryption of $i = m - 1$,
- R_i is an encryption of $i = m + 1$.

1 **for** $i = 1$ *to* $n_G - 1$ **do** $L_i \longleftarrow G_i / G_{i+1}$;
2 $L_{n_G} \longleftarrow E_0$;
3 **for** $i = 3$ *to* n_G **do** $R_i \longleftarrow L_{i-2}$;
4 $R_1 \longleftarrow E_0$; $R_2 \longleftarrow \mathrm{Not}(G_0)$;
5 **return** L, R;

6.4.4 An adaptation to the ElGamal setting

In the previous section, we gave an adaptation in MPC of the MJ counting function for the Paillier setting. However, we are interested in the ElGamal setting. Thankfully, most of Algorithm 93 is easy to adapt in the ElGamal setting thanks to the toolbox we provide. In this setting, the (encrypted) aggregated matrix must be encrypted in bit-encoding, so that obtaining the aggregated matrix from the list of encrypted ballots requires $n_G n_C$ parallel calls to the Aggreg protocol, which is the main drawback of this approach. Even if those computations can be made on the fly while the voters submit their ballot, if n_V is too large, the Paillier setting might be preferable as this phase would be too expensive.

A related difference is that in the Paillier setting, some procedures were performed "for free" thanks to the homomorphic property while they need the Add protocol in the ElGamal setting. As replacing each multiplication of two ciphertexts in Algorithm 93 by a call to Algorithm 58 might deteriorate the complexity too much, we made a few modifications listed below.

First, we give Algorithm 94 which allows to initialize p_i, m_i^-, m_i^+ and q_i, just as Algorithm 90, but also initialize L and R as in Algorithm 92. Finally Algorithm 94 also initializes C as the indicator of the candidates whose median is the best median. On this occasion, recall that m^{bits} is a (trivial) bitwise encryption of the integer m.

Algorithm 94 is a merger of Algorithms 89, 90 and 92. Merging all three algorithms together allows to exploit common intermediate computations. Note that at line 4, we compute $\lceil n_V/2 \rceil > s_{i,j}$ instead of $n_V > 2s_{i,j}$, so as to use one bit fewer. (See Lemma 15 which states that the two comparisons are equivalent.)

Lemma 15. *For all $n, s \in \mathbb{Z}$, we have $n > 2s$ if and only if $\lceil n/2 \rceil > s$.*

Proof. Let n, s be integers. If $n > 2s$, $\lceil n/2 \rceil \geq n/2 > s$. Conversely, suppose that $\lceil n/2 \rceil > s$. We first consider the case where n is even. Then $n/2 = \lceil n/2 \rceil$ so $n = 2\lceil n/2 \rceil > 2s$. If n is odd, we have $\lceil n/2 \rceil = (n+1)/2$ so $n+1 > 2s$, therefore $n+1 \geq 2s+1$, hence $n \geq 2s$. Since n is odd, $n \neq 2s$, thus $n > 2s$. □

In Algorithm 93, we did not have to initialize C (see Lemma 11). However, as the variables could be negative, we decided to add a constant. This would not be that easy in the ElGamal setting since adding a constant to a bit-encoded encrypted integers requires a non-trivial operation. In this case, eliminating the candidates who do not have the best median right away so as

Algorithm 93: MJ: MPC version (Paillier setting)

Requires: n_V, the number of voters
n_G, the number of grades
n_C, the number of candidates
Inputs: A, the (encrypted) aggregated matrix
Outputs: c, the indicator of the set of winners.

for $i = 1$ *to* n_C **do** $C_i \longleftarrow E_1$;
$G \longleftarrow$Grade(A);
$P, M^-, M^+, Q \longleftarrow$InitD$(A, G)$;
$L, R \longleftarrow$InitP(G);
for $j = 1$ *to* n_G **do**
  (* scores computation *)
  for $i = 1$ *to* n_C *(in parallel)* **do**
    $B_1 \longleftarrow \mathrm{GTH}(P_i, Q_i)$ (* $p_i \geq q_i$ *);
    $S_i \longleftarrow \mathrm{If}(B_1, P_i, 1/Q_i)$ (* p_i if $p_i \geq q_i$, $-q_i$ otherwise *);
    $S_i \longleftarrow \mathrm{If}(C_i, S_i, E_{-n_V})$ (* eliminated candidates get the fake $-n_V$ score*);
    $S_i \longleftarrow E_{n_V} S_i$ (* $s_i = s_i + n$ *);
  $S \longleftarrow S_1$ (* research of the best score *);
  for $i = 2$ *to* n_C **do**
    $B_2 \longleftarrow \mathrm{GTH}(S_i, S)$;
    $S \longleftarrow \mathrm{If}(B_2, S_i, S)$ (* s_i is $s_i \geq s$, s otherwise *)
  for $i = 1$ *to* n_C **do**
    $B_3 \longleftarrow \mathrm{EQH}(S, S_i)$;
    $C_i \longleftarrow \mathrm{Mul}(C_i, B_3)$ (* elimination of those who do not have the best score *);
  $B_4 \longleftarrow \mathrm{GTH}(S, E_n)$;
  for $i = 1$ *to* n_C *(in parallel)* **do**
    $A'_{i,m-I^-} \longleftarrow \mathrm{Select}((A_{i,1}, \cdots, A_{i,n_G-1}), L)$;
    $A'_{i,m+I^+} \longleftarrow \mathrm{Select}((A_{i,2}, \cdots, A_{i,n_G}), R)$;
    $T^+ \longleftarrow \mathrm{If}(B_4, M_i^+/M_i^-, A'_{i,m+I^+})$ (* $m_i^+ - m_i^-$ if $b_4 = 1$, $a_{i,m+I^+}$ otherwise *);
    $T^- \longleftarrow \mathrm{If}(B_4, A'_{i,m-I^-}, M_i^-/M_i^+)$ (* $a_{i,m+-I^-}$ if $b_4 = 1$, $m_i^- - m_i^+$ otherwise *);
    $P_i \longleftarrow \mathrm{If}(B_4, P_i/A'_{i,m-I^-}, P_i)$ (* $p_i - a_{i,m-I^-}$ if $b_4 = 1$, p_i otherwise *);
    $M_i^- \longleftarrow T^-$;
    $M_i^+ \longleftarrow T^+$;
    $Q_i \longleftarrow \mathrm{If}(B_4, Q_i, Q_i/A'_{i,m+I^+})$ (* q_i if $b_4 = 1$, $q_i - a_{i,m+I^+}$ otherwise *);
  $L \longleftarrow \mathrm{CLS}(L, B_4)$; $R \longleftarrow \mathrm{CRS}(R, \mathrm{Not}(B_4))$
$c \longleftarrow \mathrm{Dec}(C)$ (* bit-wise decryption *);
return c

Algorithm 94: InitALL

Requires: n_V, the number of voters
n_C, the number of candidates
n_G, the number of grades

Inputs: A, s.t., for all (i,j), $\boldsymbol{A_{i,j}}$ is a bitwise encryption of $a_{i,j}$, the number of voters who gave the grade j to candidate i

Outputs: P, M^-, M^+, Q, C s.t., for all $i \in [1, n_C]$,

- $\boldsymbol{P_i}$ is a bitwise encryption of p_i, the number of grades received by i which are strictly greater than the best median,
- $\boldsymbol{M_i^-}$ is a bitwise encryption of $\lfloor n_V/2 \rfloor - p_i$,
- $\boldsymbol{Q_i}$ is a bitwise encryption of q_i, the number of grades received by candidate i which are strictly worse than the best median,
- $\boldsymbol{M_i^+}$ is a bit-wise encryption of $\lfloor n/2 \rfloor - q_i$,
- C_i is an encryption of 1 if i's median is the best median, of 0 otherwise.

Outputs: L, R s.t., if N is the best median, then for all $j \in [1, n_G]$,

- L_j is an encryption of 1 if $j = N - 1$ and 0 otherwise,
- R_j is an encryption of 1 if $j = N + 1$ and 0 otherwise.

for $i = 1$ *to* n_C *(in parallel)* **do**
  $\boldsymbol{S_{i,1}} \longleftarrow \boldsymbol{A_{i,1}}$;
  for $j = 1$ *to* $n_G - 1$ **do**
    $D_{i,j} \longleftarrow \text{LtKnown}(\boldsymbol{S_{i,j}}, \lceil n_V/2 \rceil)$;
    $\boldsymbol{S_{i,j+1}} \longleftarrow \text{Add}(\boldsymbol{S_{i,j}}, \boldsymbol{A_{i,j+1}})$ (* $s_{i,j} = \sum_{k=1}^{j} a_{i,k}$ *);
for $j = 1$ *to* $n_G - 1$*(in parallel)* **do**
  $G_j \longleftarrow \text{And}(D_{1,j}, \cdots, D_{n_C,j})$;
for $i = 1$ *to* n_C *(in parallel)* **do**
  for $j = 1$ to $n_G - 1$ (in parallel) **do** $X_{i,j} \longleftarrow \text{Eq}(G_j, D_i, j)$;
  $C_i \longleftarrow \text{And}(X_{i,1}, \cdots, X_{i,n_G-1})$;
$L, R \longleftarrow$InitP(G);
for $i = 1$ *to* n_C *(in parallel)* **do**
  $\boldsymbol{P_i} \longleftarrow \prod_{j=1}^{n_G-1} \text{CSZ}(\boldsymbol{S_{i,j}}, L_j)$ (* Bit-wise product and CSZ, as in Select *);
  $\boldsymbol{Q_i} \longleftarrow \prod_{j=2}^{n_G} \text{CSZ}(\boldsymbol{S_{i,j}}, L_{j-1})$ (* same as above *);
  $\boldsymbol{Q_i} \longleftarrow \text{Sub}(\boldsymbol{S_{i,n_G}}, \boldsymbol{Q_i})$;
  $\boldsymbol{M_i^-} \longleftarrow \text{SubKnown}(\lfloor n_V/2 \rfloor, \boldsymbol{P_i})$;
  $\boldsymbol{M_i^+} \longleftarrow \text{SubKnown}(\lfloor n_V/2 \rfloor, \boldsymbol{Q_i})$;
return P, M^-, M^+, Q, C, L, R;

to initialize C consistently with Algorithm 88 has approximately the same cost. Afterwards, for all i, we have $|\ s_i\ | \leq \lfloor n_V/2 \rfloor$ so we can add the constant $2^{\ell-1}$ instead, where ℓ is the bit length of the integers. Indeed, $2^{\ell-1} > \lfloor n_V/2 \rfloor \geq q_i$ and $2^{\ell-1} + p_i \leq 2^{\ell-1} + \lfloor n_V/2 \rfloor < 2^{\ell}$. This is of interest because computing $2^{\ell-1} + p_i$ is completely free: we just add E_1 as the most significant bit. Therefore, we only need to call Neg once to compute $2^{\ell-1} - q_i$ instead of calling two Add. This gives Algorithm 95, our ElGamal version of a fully-hiding tallying of MJ.

6.4.5 Comparison with [CPST18]

Compared to [CPST18], we compute the real Majority Judgment counting function, and not the majority gauge heuristic. Hence, the fear is that our approach may be less efficient than that of [CPST18]. Actually, this is not as simple as that. Indeed, recall that the Paillier setting is more expensive than the ElGamal setting. Consequently, considering that $n_T \leq 5$, verifying the validity of the ballots *and* computing the aggregation phase in the ElGamal setting is only about twice as expensive as just verifying the validity of the ballots in the Paillier setting. Since the voting process can be several orders of magnitude cheaper on the voter side, this is an interesting trade-off. As for the remaining of the process, we remark that we are actually more efficient computation-wise, but less efficient communication-wise. Overall, our toolbox allows to compute the real Majority Judgment counting function without moving from practical to impractical. See Table 18 for the approximate complexities of both approaches.

6.5 Single choice voting

In single choice voting, the voters can only pick one choice between several possibilities. The choice can be a candidate (basic voting) or a list of candidates (list voting). In any case, single choice voting can be handled by a homomorphic tally: although the resulting voting system would leak more than just the result (*i.e.* the name of the winners), the risk for an Italian attack is arguably very low. Despite from that, the first iteration of [KLM+20] proposed a solution for single choice voting, which was designed to reveal the s choices (candidate or list of candidates) who received the most votes, where s is the number of seats. Unfortunately, their approach suffers from a shortcoming where more than s candidates may be output in case of a tie between two candidates. To solve this, it is possible to encode a tie-breaking mechanism in the least significant bits of the score of each candidate, as explained in Section 5.1.3.

The solution of Ordinos, once fixed, can be interesting when a choice corresponds to a single candidate. However, when it comes to list voting, revealing which lists received the most votes is not enough: often, a rule is applied to distribute the s seats among the different lists, depending on the number of votes they each received. One popular approach for this is the D'Hondt method, which is notably used in Belgium for politically binding elections.

If $c_1, \cdots, c_{n_C}$ is the number of votes received by each list and s the number of seats, the D'Hondt method defines the parameters $w_1, \cdots, w_s$ with $w_1 < w_2 < \cdots < w_s$, and constructs the values c_i/w_j for all i, j. The s greatest values from those coefficients each come from a list i (*i.e.*, they are of the form c_i/w_j for some j), and therefore grants a seat to the list i. The way that the list distributes the granted seats among its candidates is up to the political party (alternatively, it can be encoded in the ordering of the candidates in the list). Generally, it is common to take $w_j = j$ for all j, so that we only considered this possibility. In this thesis, we fix the shortcoming of Ordinos, where more than s candidates may be output in case of a tie, we propose an adaptation of Ordinos for the D'Hondt method and provide an equivalent in

Algorithm 95: MJ: MPC version (ElGamal setting)

Requires: n_C, the number of candidates
n_V, the number of voters
$\ell = \lceil \log(n_V + 1) \rceil$

Inputs: B, the n_V encrypted ballots (each is a bitwise encrypted grade matrix)

Outputs: c, the indicator of the set of winners.

for $i = 1$ *to* n_C *(in parallel)* **do**
  for $j = 1$ *to* n_G *(in parallel)* **do**
    $\boldsymbol{A_{i,j}} \longleftarrow \text{Aggreg}(B_i[j,1], \cdots, B_i[j,n_V])$;
$P, M^-, M^+, Q, C, L, R \longleftarrow \text{InitALL}(A)$;
for $j = 1$ *to* n_G **do**
  for $i = 1$ *to* n_C *(in parallel)* **do**
    $B_1 \longleftarrow \text{Not}(\text{Lt}(\boldsymbol{P_i}, \boldsymbol{Q_i}))$;
    $\boldsymbol{P_i^+} \longleftarrow P_{i,0}, \cdots, P_{i,\ell-2}, E_1$ (* $2^{\ell-1} + p_i$ *);
    $\boldsymbol{Q_i^+} \longleftarrow \text{Neg}(\boldsymbol{Q_i})$ (* $2^{\ell-1} - q_i$ *);
    $\boldsymbol{S_i} \longleftarrow \text{If}(B_1, \boldsymbol{P_i^+}, \boldsymbol{Q_i^+})$;
    $\boldsymbol{S_i} \longleftarrow \text{CSZ}(\boldsymbol{S_i}, C_i)$ (* give the fake score 0 to already eliminated candidates *);
  $\boldsymbol{S}, \longleftarrow \text{Max}(\boldsymbol{S_1}, \cdots, \boldsymbol{S_{n_C}})$ (* we do not compute the index of the maximum *);
  for $i = 1$ *to* n_C *(in parallel)* **do**
    $B_3 \longleftarrow \text{Eq}(\boldsymbol{S}, \boldsymbol{S_i})$;
    $C_i \longleftarrow \text{And}(C_i, B_3)$;
  $B_4 \longleftarrow S_{\ell-1}$ (* the m.s.b. of s tells whether $s \geq 2^{\ell-1}$ *);
  for $i = 1$ *to* n_C *(in parallel)* **do**
    $\boldsymbol{A'_{i,m-I^-}} \longleftarrow \text{Select}((\boldsymbol{A_{i,1}}, \cdots, \boldsymbol{A_{i,n_G-1}}), L)$;
    $\boldsymbol{A'_{i,m+I^+}} \longleftarrow \text{Select}((\boldsymbol{A_{i,1}}, \cdots, \boldsymbol{A_{i,n_G-1}}), R)$;
    $\boldsymbol{M_{+-}} \longleftarrow \text{Sub}(\boldsymbol{M_i^+}, \boldsymbol{M_i^-})$;
    $\boldsymbol{M_{-+}} \longleftarrow \text{Neg}(M_{+-}{}^{\text{bits}})$;
    $\boldsymbol{T^+} \longleftarrow \text{If}(B_4, \boldsymbol{M_{+-}}, \boldsymbol{A'_{i,m+I^+}})$;
    $\boldsymbol{T^-} \longleftarrow \text{If}(B_4, \boldsymbol{A'_{i,m-I^-}}, \boldsymbol{M_{-+}})$;
    $\boldsymbol{P_i} \longleftarrow \text{If}(B_4, \text{Sub}(\boldsymbol{P_i}, \boldsymbol{A'_{i,m-I^-}}), \boldsymbol{P_i})$;
    $\boldsymbol{M_i^-} \longleftarrow \boldsymbol{T^-}$;
    $\boldsymbol{M_i^+} \longleftarrow \boldsymbol{T^+}$;
    $\boldsymbol{Q_i} \longleftarrow \text{If}(B_4, \boldsymbol{Q_i}, \text{Sub}(\boldsymbol{Q_i}, \boldsymbol{A'_{i,m+I^+}}))$;
  $L \longleftarrow \text{CLS}(L, B_4)$; $R \longleftarrow \text{CRS}(R, \text{Not}(B_4))$;
$c = \text{Dec}(C)$ (* bit-wise decryption *);
return c

Table 18: Leading terms of the cost of computing the Majority Judgment in MPC; n_V is the number of voters, n_C is the number of candidates and n_G is the number of grades

Version	Voters (# exp.)	Talliers (# exp.)	Talliers (# synch. steps)	Transcript (key size)
[CPST18] (Paillier)	$5n_Cn_G$	$4n_Vn_Cn_G+$ $n_C \log n_V n_T(224n_C + 58n_G)$	$(4 \log n_V + n_G)n_T$	$6n_Vn_Cn_G+$ $n_C \log n_V n_T(280n_C + 62n_G)$
Ours (ElGamal)	$6n_Cn_G$	$99n_Vn_Cn_Gn_T+$ $33n_C \log n_V n_T n_G(23 + 2n_G)$	$\frac{1}{2} \log(n_V)^2 n_T+$ $2n_G \log n_C \log n_V n_T$	$102n_Vn_Cn_Gn_T+$ $34n_C \log n_V n_T n_G(23 + 2n_G)$

the ElGamal setting, using our toolbox. In addition, we propose two additional computation-communication trade-offs in the ElGamal setting.

6.5.1 Basic single choice voting

To find the s largest values in a list of N ciphertexts $c_1, \cdots, c_N$, the strategy of Ordinos consists of first building the (encrypted) matrix M_{rank} of the pairwise comparisons $c_i \geq c_j$ for all i, j. Then, to decide whether a candidate i is a winner, they compute an encryption of the sum $S_i = \sum_{j=1}^{N} \mathbb{1}_{c_i \geq c_j}$, using the homomorphic property of the Paillier encryption scheme. Finally, they produce an encryption of 1 if $S_i \geq N - s + 1$, of 0 otherwise, and decrypt the result of the test. Hence, the candidate i is a winner if there are at most $s - 1$ candidates which have strictly more votes than i. As mentioned above, this can lead to more than s candidates being elected in case of a tie. To fix this, we propose to encode the tie-beak function in the least significant bits of the score of each candidates. More precisely, if C_i is the encryption of the number of votes received by i and $r_i \in [1, N]$ is a number which encodes the tie break rule for i (*i.e.* if there is a tie between i and j, then the one with the largest r is preferred), then we replace C_i by $C_i^{2^\ell} E_{r_i}$, where $\ell = \lceil \log(N + 1) \rceil$ is the number of bits required to encode each r, and E_{r_i} is a trivial encryption of r_i. Hence, a tie can no longer occur and the strategy of Ordinos can be applied. The impact of this fix in the overall performances of Ordinos is low: we only increase the size of the integers to compare by ℓ, which means that we lose less than a factor 2.

Adaptation to the ElGamal setting. Thanks to our toolbox, it is easy to compute the winner of a basic single-choice voting election, using Aggreg and sSelect or OddEvenMergeSort. However, we can also adapt Ordinos strategy using pairwise comparisons. This leads to different computation/communication trade-offs; each can be interesting depending on the ratio between s and n_C. In Table 19, we give the approximate costs of all approaches, which includes the fixed solution of Ordinos. Once again, we conclude that our toolbox is more efficient computation-wise, but less efficient communication-wise. However, in this particular case, the additional synchronization steps that are required in the ElGamal setting are affordable, and it is still possible to switch to more communication-efficient protocols such as CLt ar UFCAdd if needed.

6.5.2 List voting: computing the D'Hondt method in MPC

We now explain how to adapt the strategies from the previous section to compute a D'Hondt tally in MPC. Although the D'Hondt method can be computed with a homomorphic tally, this is a good opportunity to evaluate the performances of our toolbox for a more complex counting function. First, the strategy of Ordinos can be adapted by computing the pairwise comparisons $c_i/w_j \geq c_{i'}/w_{j'}$. Sice comparing two fractions may be expensive, it is more efficient to precompute all the product $c_i w_{j'}$ beforehand. For this purpose, one can use the efficient UFC algorithm (see Algorithm 72), which, given s (dupicated) bitwise encryptions of c_i, returns the bitwise

Table 19: Leading terms of the cost of different MPC solutions for single choice voting; n_V is the number of voters, n_C the number of candidates, s the number of seats, n_T the number of talliers

Version	# exp.	# synch. steps	transcript
[KLM+20] (fixed)	**precomp.** $41{n_C}^2 \log(n_V n_C) n_T$ **comp.** $4 n_V n_C +$ $25{n_C}^2 \log(n_V n_C) n_T$	**precomp.** $O(n_T)$ **comp.** $14 \log\log(n_V n_C)$	$9 n_V n_C +$ $79.5{n_C}^2 \log(n_V n_C) n_T$
EG (adaptation)	$99 n_V n_C n_T +$ $33{n_C}^2 \log(n_V n_C) n_T$	$\frac{1}{2}(\log(n_V)^2 + \log(n_C)^2) n_T$	$102 n_V n_C n_T +$ $34{n_C}^2 \log(n_V n_C) n_T$
EG (sSelect)	$99 n_V n_C n_T +$ $33 n_C s(3 \log n_V + \log n_C) n_T$	$\frac{1}{2}\log(n_V)^2 n_T +$ $2s \log n_V \log n_C n_T$	$102 n_V n_C n_T +$ $34 n_C s(3 \log n_V + \log n_C) n_T$
EG (OddEven)	$99 n_V n_C n_T +$ $25 n_C \log(n_C)^2 \log n_V n_T$	$\frac{1}{2}\log(n_V)^2 n_T +$ $\log n_V \log(n_C)^2 n_T$	$102 n_V n_C n_T +$ $25.5 n_C \log(n_C)^2 \log n_V n_T$

Table 20: Leading terms of the cost of the different MPC solutions for the D'Hondt method; n_V is the number of voters, n_C is the number of lists of candidates, s is the number of seats, $m = \mathrm{lcm}(1, \cdots, s)$, n_T is the number of talliers and all the logarithms are in base 2

Version	# exp.	# synch. steps	transcript
Adaptation of [KLM+20]	$99 n_V n_C n_T +$ $+33{n_C}^2 s^2 \log(n_V n_C s) n_T$	$\frac{1}{2}(\log(n_V)^2 + \log(n_C)^2) n_T$ $+2 \log s \log n_V n_T$	$102 n_V n_C n_T$ $+34{n_C}^2 s^2 \log(n_V n_C s) n_T$
sSelect	$99 n_V n_C n_T +$ $33 n_C s^2 \log(m^3 {n_V}^6 n_C s) n_T$	$\frac{1}{2}\log(n_V)^2 n_T +$ $2s \log(m n_V) \log(n_C s) n_T$	$102 n_V n_C n_T +$ $34 n_C s^2 \log(m^3 {n_V}^6 n_C s) n_T$
OddEven	$99 n_V n_C n_T +$ $99 n_C s^2 \log n_V n_T +$ $25 n_C s \log(n_C s)^2 \log(m n_V) n_T$	$\frac{1}{2}\log(n_V)^2 n_T +$ $2 \log n_V \log m$ $\log(m n_V) \log(n_C s)^2 n_T$	$102 n_V n_C n_T +$ $102 n_C s^2 \log n_V n_T +$ $25.5 n_C s \log(n_C s)^2 \log(m n_V) n_T$

encryptions $\boldsymbol{S_{i,1}}, \cdots, \boldsymbol{S_{i,s}}$ of $c_i, 2c_i, \cdots, sc_i$. The cost of this protocol is negligible compared to the remaining of the process. Then, we can add the tie-breaking mechanism in the least significant bits and apply Ordinos' strategy. This leads to a solution which is efficient communication-wise, but requires $\Omega({n_C}^2 s^2)$ operations, where n_C is the number of candidates and s is the number of voters. Hence, it may be preferable to also adapt the other solutions for computing the s largest values, namely sSelect and OddEvenMergeSort. For those solution, however, precomputing all the $\boldsymbol{S_{i,j}}$ does not help much. Indeed, while $c_i/w_j \geq c_{i'}/w_{j'}$ is indeed equivalent to $c_i w_{j'} \geq c_{i'} w_j$, sSelect and OddEvenMergeSort imply a lot of conditional swaps, which means that the index i, j and i', j' are not known. Consequently, we propose to multiply by the least common multiple $m = \mathrm{lcm}(1, \cdots, s)$ to get an encryption of the integers $d_{i,j} = c_i \frac{m}{j}$ for all i, j. Since one of the operands is known, a slightly optimized version of Mult can be use, where a third of the computation is saved. To simplify the complexity analysis, we consider $e = \exp(1)$ and we note that, by [RS62, Theorem 12], $\log m < 1.039 s \log e$. Hence, the cost of computing the $n_C s$ multiplications in parallel is approximately that of $2 n_C s^2 \log n_V \log e \mathrm{CSZ} < 3 n_C s^2 \log n_V \mathrm{CSZ}$, and this approximation is valid for all s. This means that the cost of the multiplications is reasonable compared to the rest of the protocol. However, by multiplying the values to compare by m, we make the comparisons more expensive since there are $s \log e$ additional bits to process.

6.6 Security of the toolbox in the context of electronic voting

In Section 4.4, we proved the SUC-security of the conditional gate. Since our toolbox is only composed of conditional gates, it means that for every combination of protocols of our toolbox, the resulting protocol is as secure as if it was performed by some honest third party. Finally, in the context of electronic voting, it is usual that we require the talliers to actually decrypt something at some point; for instance, in the STV protocol, we decrypt the vector W of the winners. Since the threshold decryption itself is not SUC-secure, a risk is that we might lose the SUC-security because of this last step. For this reason, we give Theorem 6, which gives states that the SUC-security is not lost in our case. The intuition is that a conditional gate followed by a reencryption phase is the same as just a conditional gate. Hence, by Lemma 8, it follows that if the only elements that we decrypt are some outputs of a conditional gate, then the SUC-security is preserved. Note that the same result apply if we replace the conditional gate by, for instance, If, Or, Xor, And, Eq, Lt and their negations using Not. Indeed, since the CSZ protocol is SUC-secure, it is easy to show that they are also SUC-secure.

In Theorem 6, we use the following notations:

- We denote CS the counting function defined by the Condorcet-Schulze method and P_{CS} the protocol that we provide in Section 6.2 to compute CS (see Algorithm 82). We denote $\mathcal{F}_{\text{CS}}$ the trusted party that honestly evaluates P_{CS} and returns the output of all the conditional gates as well as the result (*i.e.* the set of the winners).
- We denote STV the counting function defined by the STV method and P_{STV} any of the two protocols that we provide in Section 6.3.4 to compute STV. We denote $\mathcal{F}_{\text{STV}}$ the trusted party that honestly evaluates P_{STV} and returns the output of all the conditional gates as well as the result (*i.e.* the set of the winners).
- We denote MJ the counting function defined by the Majority Judgment and P_{MJ} the protocol that we provide in Section 6.4.4 to compute MJ (see Algorithm 95). We denote $\mathcal{F}_{\text{MJ}}$ the trusted party that honestly evaluates P_{MJ} and returns the output of all the conditional gates as well as the result (*i.e.* the set of the winners).
- We denote DH the counting function defined by the D'Hondt method and P_{DH} any of the protocols that we provide in Section 6.5.2 to compute DH. We denote $\mathcal{F}_{\text{DH}}$ the trusted party that honestly evaluates P_{DH} and returns the output of all the conditional gates as well as the result (*i.e.* the set of the winners).

Theorem 6. *Under the DDH assumption and if at least one participant is honest, for* $\mathsf{count} \in \{\text{CS}, \text{STV}, \text{MJ}, \text{DH}\}$, P_{count} *SUC-securely computes* $\mathcal{F}_{\mathsf{count}}$ *in the* $\mathcal{F}_{RO}, \mathcal{F}_{B}$*-hybrid model. (Recall that they model the ROM and the ideal broadcast channel.)*

Proof. Let $\mathsf{count} \in \{\text{CS}, \text{STV}, \text{MJ}, \text{DH}\}$. First, by Theorem 3, the conditional gate protocol SUC-securely realizes $\mathcal{F}_{\text{CSZ}}$ in the $\mathcal{F}_{RO}, \mathcal{F}_{B}$-hybrid model. Therefore, we can replace every conditional subprotocol in P_{count} by a call to the trusted party $\mathcal{F}_{\text{CSZ}}$ and show that the resulting protocol SUC-securely computes $\mathcal{F}_{\mathsf{count}}$ in the $\mathcal{F}_{\text{CSZ}}$-hybrid model. This is a consequence of the composition theorem, stated in Lemma 5.

Now, we construct a simulator $\mathcal{S}$ which interacts with the environment in the ideal process and simulates the hybrid process by simulating the honest participants and the $\mathcal{F}_{\text{CSZ}}$ ideal functionality. First, by interacting with the ideal process, $\mathcal{S}$ gets the outputs of all the conditional gates, as well as the result r. Afterwards, $\mathcal{S}$ proceeds with the simulation of P_{count}.

Remark that for all of our MPC protocols, P_{count} is divided into two phases. First, the MPC part feature no communication between the participants, except during a conditional gate subprotocol. Second, the final step is to decrypt a vector W of ciphertexts, using the threshold decryption protocol. Note, in addition, that those ciphertexts consist of outputs of a conditional gate protocol.

Hence, to simulate the hybrid process, $\mathcal{S}$ can also proceed into two phases. During the first phase, $\mathcal{S}$ only has to simulate the answers of $\mathcal{F}_{\mathrm{CSZ}}$. For this purpose, $\mathcal{S}$ first look whether this answer is one of the ciphertexts of W or not (*i.e.* if the ciphertext will be decrypted in a subsequent threshold decryption protocol). If this is not the case, $\mathcal{S}$ uses the answer of the ideal functionality $\mathcal{F}_{\mathsf{count}}$, which includes the output of all the conditional gates. Otherwise, $\mathcal{S}$ uses a random encryption of the corresponding plaintext z, using a known randomness ρ. Note that $\mathcal{S}$ can deduce z from the result r output by $\mathcal{F}_{\mathsf{count}}$. This way, $\mathcal{S}$'s answers are perfectly indistinguishable from that of $\mathcal{F}_{\mathrm{CSZ}}$.

Once the first phase has terminated, $\mathcal{S}$ must simulate the interactions during the threshold decryption protocols. First, the ZKP of correct partial decryption can be simulated in the ROM thanks to their zero knowledge property. Therefore, it only remains to explain how the simulator can generate the partial decryptions. For this purpose, suppose that $\mathcal{S}$ needs to simulate the decryption of a ciphertext $Z = (x, y)$, which is an output of a conditional gate protocol. Then, we have $x = g^{\rho}$ and $y = zh^{\rho}$, where (g, h) is the public encryption key, z is the corresponding plaintext and ρ the randomness chosen by $\mathcal{S}$. As seen in the proof of Lemma 8, this allows the simulator to compute the partial decryptions of all the participants, and hence to perfectly simulate the threshold decryption protocol. Indeed, if h_i is the public commitment of the participant i, then the partial decryption of i is $w_i = h_i^{\rho}$.

With the above $\mathcal{S}$, the simulated hybrid process is perfectly indistinguishable from the real one. □

In what follows, we explain how this SUC-security can be used to prove the privacy and the verifiability of a voting system that uses our toolbox to compute the tally in MPC. For simplicity, we only give the proof in the case of Condorcet-Shulze. For this purpose, we define a minimal voting system that we call TH-voting; however, since we only considered the tally process, we do not detail how the other phases are taken care of. Hence, TH-voting is defined as follows:

Setup. We consider an ideal DKG that produces a public key pk, the public commitments $(h_i)_{i=1}^{n_T}$ and distributes their secret shares s_i to the talliers.

Register. We consider an ideal registration where each voter v received an ElGamal key pair $\mathsf{pk}_v, \mathsf{sk}_v$, and where the public key of each eligible voter is published on the board.

Vote. To vote, a voter produces $n_C \log n_C$ encryptions of 0 or 1, and give the corresponding PoK that they are all encryptions of 0 or 1. Finally, they sign the resulting ballot using sk_v. The ballot has the form $(\mathsf{pk}_v, B, \pi, s)$, where B is the matrix of the encrypted bits, π contains the corresponding PoK and s the signature of B.

Check. The voter checks that the last cast ballot B appears on the board PB, and that no subsequent ballot uses the same public signature key pk_v.

Valid. To verify the validity of a ballot, we verify the signature and the ZKP, and we also verify that no previously cast ballot uses the same matrix B.

Tally. To compute the tally, the talliers first keep, for each credential pk_v, the last valid ballot that uses pk_v as a verification key. Then they use the MPC protocol described in Algorithm 82; see Section 6.2 for more details.

Verify. To verify the validity of the tally, first verify the valididy of the ballots on the board and, from the list of the valid ballots and the given transcript, compute the output of all the

$\mathrm{Exp}^{\mathsf{verb}}(\lambda, n_T, t, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_{i=1}^{n_T}, \Pi^{\mathsf{S}} \leftarrow \mathsf{Setup}(\lambda, n_T, t)$;
$1^n \longleftarrow \mathbb{A}(\mathsf{pk}, \Pi^{\mathsf{S}})$;
$(c_i, \pi_i)_{i=1}^{n}, \Pi^{\mathsf{R}} \leftarrow \mathsf{Register}(\mathsf{pk}, n)$;
$\mathcal{CU} \longleftarrow \varnothing$;
for $i = 1$ *to* n **do**
  $\mathsf{HV}_i \longleftarrow \perp$; $L_i \longleftarrow \perp$; $\mathsf{Checked}_i \longleftarrow 0$;
$(\mathsf{PB}, r, \Pi) \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathsf{corrupt}}, \mathcal{O}_{\mathsf{vote}}}$;
$\mathbb{A}^{\mathcal{O}_{\mathsf{check}}}$;
if $\mathrm{Verify}(\mathsf{PB}, \Pi, r) = 0$ **then return** 0;
if $\exists\, \mathrm{L} \subset \{\!\{(i, \mathsf{HV}_i) \mid i \notin \mathcal{CU}, \mathsf{Checked}_i \neq 1, \mathsf{HV}_i \neq \perp\}\!\}$,
  $\exists C$ such that $\mid |C| \leq |\mathcal{CU}|$ **and**
  $r = \mathsf{count}(\{\!\{(i, \mathsf{HV}_i) \mid i \notin \mathcal{CU}, \mathsf{Checked}_i = 1\}\!\} \uplus \mathrm{L} \uplus \mathrm{C})$
  then return 0 **else return** 1;

$\mathcal{O}_{\mathsf{corrupt}}(\mathsf{id})$

$\mathcal{CU} \longleftarrow \mathcal{CU} \bigcup \{\mathsf{id}\}$;
return c_{id};

$\mathcal{O}_{\mathsf{vote}}(\mathsf{id}, \nu)$

$B \longleftarrow \mathsf{Vote}_{\mathsf{pk}}(\nu, c_{\mathsf{id}})$;
$\mathsf{HV}_{\mathsf{id}} \longleftarrow \nu$;
$L_i \longleftarrow B$;
$\mathsf{Checked}_{\mathsf{id}} \longleftarrow 0$;
return B;

$\mathcal{O}_{\mathsf{check}}(\mathsf{id})$

$\mathsf{Checked}_{\mathsf{id}} \leftarrow \mathsf{Check}(\nu, c_{\mathsf{id}}, L_{\mathsf{id}}, \mathsf{PB})$

conditional gates. Then use the transcript of the threshold decryptions to deduce the result and verify that is it corresponds to te given result. Finally, verify that each conditional gate and each threshold decryption has a corresponding valid ZKP.

6.6.1 Universal verifiability

The universal verifiability of our tally process is a direct consequence of the computational soundness of the ZKP and the correctness of the tally protocol. Indeed, consider the definition of end-to-end verifiability of [CGGI14] (introduced in Section 1.2.2), which combines the individual and the universal verifiability. Since allowing revoting would require to adapt the definition and is independent from the tally process, we do not consider it and assume that the adversary can call $\mathcal{O}_{\mathsf{vote}}$ at most once for all voter. (To improve readability, we reproduce Fig. 2 below, which describes the verifiability experiment considered in this definition.) In what follows, we give a proof sketch that our minimal voting system has end-to-end verifiability.

Theorem 7. *In the ROM and assuming the strong unforgeability of the signature scheme, TH-voting has end-to-end verifiability as of Definition 3.*

Proof sketch. To win the verifiability experiment, the adversary must give a transcript which contains valid ZKP. Yet, by the soundness of those ZKP, the result r must be the same as the one computed from PB using an instance of the tally protocol. Now, since all the happy voters verified that their ballot is in PB and that no subsequent ballot uses the same pk_v, it means that their ballots are included in the tally. In addition, by the strong unforgeability of the signature, for all valid ballot in the board such that pk_v is not the credential of a honest voter, pk_v must be the credential of a corrupted voter. Hence, since we keep up to one ballot per credential, the condition $|C| \leq |\mathcal{CU}|$ is verified. Finally, the strong unforgeability also guarantees that if a ballot that uses the credential pk_v of a lazy voter is valid, then it must be a ballot output by $\mathcal{O}_{\mathsf{vote}}$. □

6.6.2 Privacy

Proving the privacy of our voting system is less straightforward than for the verifiability. A first difficulty, discussed in Section 1.3.5, is that there is no notion of privacy which is satisfactory for

Algorithm 96: $\text{Real}^{\text{Priv}}$

Requires: $\lambda, n_T, C_t, n, n_A, n_C, \mathscr{B}, \mathbb{A}$

1 $\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_{i=1}^{n_T}, \Pi^{\mathsf{S}} \leftarrow \mathsf{Setup}(\lambda, n_T, t)$;
2 $(c_i, \pi_i), \Pi^{\mathsf{R}} \longleftarrow \mathsf{Register}(\mathsf{pk}, n)$;
3 $\mathsf{PB} \longleftarrow \Pi^{\mathsf{S}} || \Pi^{\mathsf{R}}$;
4 $A \longleftarrow \mathbb{A}(\mathsf{pk}, \mathsf{PB}, \{s_i \mid i \in C_t\})$;
5 $j, \nu_0, \nu_1 \longleftarrow \mathbb{A}(\{c_i \mid i \in A\})$;
6 (* chooses the voter to observe *);
7 **if** $|A| \neq n_A \vee j \notin [1, n] \backslash A$ **then**
8 **return** 0;
9 $B \xleftarrow{\$} \mathscr{B}([1, n] \backslash A, n_C)$;
10 **for** $(i, \nu_i) \in B$ **do**
11 $\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(i, \mathsf{PB})$;
12 $\mathsf{PB} \longleftarrow \mathsf{PB} || \mathsf{Vote}_{\mathsf{pk}}(\nu_i, c_i)$;
13 $\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(i, \mathsf{PB}, \texttt{"end for"})$;
14 $b \xleftarrow{\$} \{0, 1\}$;
15 $\mathsf{PB} \longleftarrow \mathsf{PB} || \mathsf{Vote}_{\mathsf{pk}}(\nu_b, c_j)$;
16 $\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(\mathsf{PB})$;
17 $r, \Pi \longleftarrow \mathsf{Tally}^{\mathbb{A}}(\mathsf{PB}, \{s_i\})$;
18 $b' \longleftarrow \mathbb{A}()$;
19 *if* $\nu_0, \nu_1 \in [1, n_C] \wedge b' == b$ **then** **return** 1 **else return** 0;

Algorithm 97: $\text{Ideal}^{\text{Priv}}$

Requires: $\lambda, n_T, C_t, n, n_A, n_C, \mathscr{B}, \mathbb{A}$

1 ;
2 ;
3 ;
4 $A \longleftarrow \mathbb{A}(\lambda)$;
5 $j, \nu_0, \nu_1 \longleftarrow \mathbb{A}()$;
6 (* chooses the voter to observe *);
7 **if** $|A| \neq n_A \vee j \notin [1, n] \backslash A$ **then**
8 **return** 0;
9 $B \xleftarrow{\$} \mathscr{B}([1, n] \backslash (A \bigcup \{j\}), n_C)$;
10 $(\nu)_{i \in A} \longleftarrow \mathbb{A}(I)$;
11 $B \longleftarrow B || (i, \nu_i)_{i \in A, \nu_i \in [1, n_C]}$;
12 ;
13 ;
14 $b \xleftarrow{\$} \{0, 1\}$;
15 $B \longleftarrow B || (j, \nu_b)$;
16 ;
17 $r \longleftarrow \mathsf{tally}(B)$;
18 $b' \longleftarrow \mathbb{A}(r)$;
19 **if** $\nu_0, \nu_1 \in [1, n_C] \wedge b' == b$ **then** **return** 1 **else return** 0;

Figure 19: Definition of privacy, λ is the security parameter, n_T the number of talliers, t the threshold, C_t the set of the corrupted talliers, n the number of voters, n_A the number of corrupted voters, n_C the number of voting options (excluding abstention) and $\mathscr{B}$ the distribution.

our specific case, where the counting function does not have the partial tally property and where we want to consider some fully corrupted talliers. For this reason, we introduced Definition 9 in Section 1.3.4. To improve readability, we reproduce the corresponding experiments above and we recall that, to prove privacy, we need to prove that for all PPT adversary $\mathbb{A}_0$ for the real game, there exists an adversary $\mathbb{B}$ for the ideal game such that, when interacting with $\mathbb{A}_0$, $\mathbb{B}$ wins the ideal game with the same probability as $\mathbb{A}_0$ wins the real game (with up to a negligible difference). We now give Theorem 8, and conclude the section with a proof of this result.

Theorem 8. *Assuming an ideal broadcast channel, under the DDH assumption an in the ROM, TH-voting has privacy as of Definition 9.*

Proof. We proceed by game hops and construct a succession of games $G_1, \cdots, G_4$ where G_4 is the ideal game. For each of these games, we construct an adversary $\mathbb{A}_i$ and we denote S_i the probability that $\mathbb{A}_i$ wins G_i.

Game 1: In this game, the adversary $\mathbb{A}_1$ is no longer able to take part in the tally process. Instead, we consider a trusted party $\mathcal{F}_{\mathsf{Tally}}$ which gets the shares of the participants and computes the result r of the tally as well as the output Π^Z of each conditional gate, by running the protocol Tally itself, when all the participants are honest. At line 18, $\mathbb{A}_1$ gets r, Π^Z and must output its guess b' from this.

To construct $\mathbb{A}_1$, we use Theorem 6 which states that Tally SUC-securely computes $\mathcal{F}_{\mathsf{Tally}}$ in the $\mathcal{F}$-hybrid model, with $\mathcal{F} = \mathcal{F}_{RO}, \mathcal{F}_B$. Hence, there exists a simulator $\mathcal{S}$ such that, for all environment $\mathcal{Z}$, $|\mathrm{Real}^{\mathcal{F}}_{\mathsf{Tally},\mathbb{A}_0,\mathcal{Z}}(\lambda,0) - \mathrm{Ideal}_{\mathcal{F}_{\mathsf{Tally}},\mathcal{S},\mathcal{Z}}(\lambda,0)|$ is negligible. In particular, we consider the environment $\mathrm{Real}^{\mathsf{Priv}}$, so that $\mathrm{Real}^{\mathcal{F}}_{\mathsf{Tally},\mathbb{A}_0,\mathcal{Z}} = S_0$. Then, $\mathbb{A}_1$ can interact with $\mathbb{A}_0$ by simulating the real game using $\mathcal{S}$, so that $\mathrm{Ideal}_{\mathcal{F}_{\mathsf{Tally}},\mathcal{S},\mathcal{Z}}(\lambda,0) = S_1$. Hence, $|S_1 - S_0|$ is negligible.

Game 2: In this game, $\mathbb{A}_2$ is no longer given Π^Z and is only given r.

We construct $\mathbb{A}_2$ that interacts with $\mathbb{A}_1$ by simulating Π^Z. For this purpose, $\mathbb{A}_2$ uses uniformly random ciphertexts.

To argue the validity of this transition, we construct an adversary $\mathbb{B}$ for DDH as follows. First, $\mathbb{B}$ gets the challenge tuple (g_1, g_2, g_3, g_4) from the DDH game and sets $\mathsf{pk} = (g_1, g_2)$. To run the setup, $\mathbb{B}$ recovers the set S of the corrupted participants from $\mathbb{A}_1$, and picks $s_i \in \mathbb{Z}_q$ at random for all $i \in S$. It the completes S into I by picking some additional $s_i \in \mathbb{Z}_q$ at random for all $i \in I \backslash S$, where $I \subset [1, n_T]$ is a set of size t that contains S, and n_T is the number of talliers. For $i \in I$, it computes $h_i = g_1^{s_i}$ and, for $i \in [1, n_T] \backslash I$, it deduces h_i with Lagrange interpolation.

It then runs the remaining of Game 2 honestly, but each time $\mathbb{A}_1$ casts a ballot, $\mathbb{B}$ extracts the corresponding voting option from $\mathbb{A}_1$'s proof of knowledge. In the ROM, this is possible in polynomial time, as a consequence of the forking lemma (see for instance Theorem [BPW12, Theorem 1]). This way, $\mathbb{B}$ can compute the result r of the tally without knowing the secret key sk. Finally, since $\mathbb{B}$ knows the cleartexts of the ballots to tally, $\mathbb{B}$ can run the tally protocol "on the cleartexts", *i.e.* it can compute the cleartext of each of the outputs of each conditional gate, since it is the product of two cleartexts. To simulate the output of a conditional gate, $\mathbb{B}$ "encrypts" the corresponding cleartext z by choosing two random $\rho_1, \rho_2 \in \mathbb{Z}_q$ and computing $Z = (g_1^{\rho_1} g_3^{\rho_2}, g_1^z g_2^{\rho_1} g_4^{\rho_2})$. Finally, if $\mathbb{A}_1$ wins the game, $\mathbb{B}$ states that the challenge was a DDH tuple; otherwise, it states that is was a random tuple. Remark that if (g_1, g_2, g_3, g_4) is a DDH tuple, then $\mathbb{B}$ played a perfect simulation of Game 1 to $\mathbb{A}_1$ and hence wins with probability S_1. On the other hand, if the challenge tuple is a random tuple, $\mathbb{B}$ played $\mathbb{A}_2$'s simulation of Game 1 and wins with probability $1 - S_2$. Yet, under the DDH assumption, $\mathbb{B}$'s advantage in the DDH game must be negligible, hence $|S_1 - S_2|$ is negligible.

Game 3: In this game, whenever a honest voter cast a ballots, a random ballot is added to the board instead of a ballot that encrypts the chosen voting option.

To argue that $|S_3 - S_2|$ is negligible, we use a hybrid argument (the hybrid lemma is stated in Theorem 2, Section 3.1.2). Technically, this is not required since the number of voters is not chosen by the adversary but is a parameter fixed by the experiment. However, giving a hybrid argument shows that the difference in probability $|S_3 - S_2|$ scales linearly with respect to n_V, which is certainly reassuring. For this purpose, we denote Game 2 G_1 and Game 3 G_2. We construct a succession of games hop $(H_i)_{\mathbb{N}}$ such that, for all i, H_i is game G_2 except that for the first i honest voters, the real ballot is added to the board instead of a random ballot. This way, $G_2 = H_0$, which is the first condition of the lemma. In addition, for all adversary $\mathbb{A}$, there exists a polynomial $n_{\mathbb{A}} = n_V$ such that $H_{n_{\mathbb{A}}} = G_1$; hence, for all $\lambda \in \mathbb{N}$, $\Pr(H_{n_{\mathbb{A}}}(\lambda, \mathbb{A}) = 1) = \Pr(G_1(\lambda, \mathbb{A}) = 1)$, which is the second condition of the lemma. Now, the third condition is that there exists a polynomial P such that for all $\mathbb{A}'_{i+1}$ for game H_{i+1}, there exists $\mathbb{A}'_i$ for game H_i which makes at most P transitions. Conversely, given $\mathbb{A}'_i$ for H_i, we need to construct $\mathbb{A}'_{i+1}$ for H_{i+1} that makes at most P transitions. In our case, $\mathbb{A}'_{i+1}$ is given an additional ballot while A'_i is given a random fake ballot instead. However, since the ballot is

encrypted anyway, it must be indistinguishable from a random. Hence we use $A'_i = \mathbb{A}'_{i+1}$ and $P = 0$.

The fourth condition is that we need a decisional game which is considered hard. For this purpose, we use the IND-PA0 game (see Algorithm 23, that we reproduce in Algorithm 98 below). Indeed, by Lemmas 4, 1 and 3, the encryption scheme Gen, Vote, Extract is IND-PA0 , where Gen is the generation algorithm for the ElGamal encryption scheme, Vote is the voting algorithm and Extract is the algorithm that verifies the ZKP of the ballot, outputs $\perp$ if it is invalid, decrypts it and outputs the corresponding voting option if it is valid.

Finally, the last condition is that there exists a uniform reduction to IND-PA0. We construct the required PPT $\mathbb{B}$ for the IND-PA0 game as follows. First, $\mathbb{B}$ is given the public key pk in the IND-PA0 game. Given i, it interacts with an adversary $\mathbb{A}'_{i+1}$ for H_{i+1} by simulating H_{i+1}. For this purpose, $\mathbb{B}$ gets the set of the corrupted talliers and generates their secret shares at random to simulate the setup as in Game 2. Then, it runs a perfect simulation of H_{i+1} by picking a random $b \in \{0, 1\}$ and sampling the distribution B at random from $\mathscr{B}$. However, for the $i+1$th honest voter, instead of creating a ballot for the corresponding voting option ν, it chooses a random voting option ν' and plays the pair ν, ν' in the IND-PA0 game. Finally, when $\mathbb{B}$ needs to output the result of the tally to $\mathbb{A}'_{i+1}$, $\mathbb{B}$ decrypts the valid ballots cast by $\mathbb{A}'_{i+1}$ by querying them to the IND-PA0 game, which allows $\mathbb{B}$ to compute the result of the tally. If $\mathbb{A}'_{i+1}$ correctly guesses the bit b, $\mathbb{B}$ states that the IND-PA0 game encrypted ν; otherwise, it states that it encrypted ν'. Now, remark that when the IND-PA0 game encrypt ν, $\mathbb{B}$ plays a perfect simulation of H_{i+1}. However, when the IND-PA0 game encrypts ν', $\mathbb{B}$ plays a perfect simulation of H_i. Hence, the last condition of the hybrid lemma is met and there exists $\mathbb{A}_3$ such that $|S_2 - S_3|$ is negligible. In addition, since we took $\mathbb{A}'_{i+1} = \mathbb{A}'_i$ for all i, we have $\mathbb{A}_3 = \mathbb{A}_2$.

Algorithm 98: $\mathrm{Exp}^{\mathsf{ind\text{-}pa0}}(\lambda, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;
$\nu_0, \nu_1 \longleftarrow \mathbb{A}(\mathsf{pk})$;
$b \xleftarrow{\$} \{0, 1\}$;
$C \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(m_b)$;
$\boldsymbol{C} \longleftarrow \mathbb{A}(C)$;
$\boldsymbol{m} \longleftarrow (\mathrm{Dec}_{\mathsf{sk}}(y))_{y \in \boldsymbol{C} \setminus \{C\}}$;
$b' \longleftarrow \mathbb{A}(\boldsymbol{m})$;
if $b = b'$ **then return** 1 **else return** 0;

Game 4: This game is the ideal game.

Finally, we construct $\mathbb{A}_4$ that interacts with $\mathbb{A}_3$ by simulating Game 3. First, $\mathbb{A}_4$ runs the setup honestly by generating a random secret key sk and acting as the trusted dealer. Then, it also runs the registration honestly and get the set of the corrupted voters A from $\mathbb{A}_3$, that it plays in the ideal game. Then it gives to $\mathbb{A}_3$ the credentials of the corrupted voters and gets j, ν_0, ν_1 in return, that it plays in the ideal game. Afterwards, it simulates the voting phase using I by emulating the public board as follows. For $i \in I$, $\mathbb{A}_4$ calls $\mathbb{A}_3$ with the input i and the current (simulated) public board PB. Whenever $\mathbb{A}_3$ casts a valid ballot using $\mathcal{O}_{\mathsf{cast}}$, by the strong unforgeability of the signature scheme, the ballot must use the credential of a a corrupted voter j. Also, by the computational soundness of the ZKP, the ballot must encrypt some valid voting option. Hence $\mathbb{A}_4$ can decrypt the ballot using sk and update ν_j using the corresponding

voting option. Finally, whenever a voter (including the honest voter) casts a ballot, $\mathbb{A}_4$ adds a random ballot in the public board, just as in Game 3. Finally, after the last call of $\mathcal{O}_{\mathsf{cast}}$ by $\mathbb{A}_3$, $\mathbb{A}_4$ plays $(\nu_j)_{j\in A}$ in the ideal game and gets the result r in return, that it forwards to $\mathbb{A}_3$. Finally, it outputs $\mathbb{A}_3$'s guess.

Clearly, except if $\mathbb{A}_3$ forges a valid ZKP for an invalid ballot or forges a signature, $\mathbb{A}_4$ plays a perfect simulation of Game 3 to $\mathbb{A}_3$, so that $|S_3 - S_4|$ is negligible.

Conclusion. By the triangular inequality, this shows that for all PPT adversary $\mathbb{A}$ for the real game, there exists a PPT adversary $\mathbb{B}$ for the ideal game that wins with the same probability, with up to a negligible difference. □

6.7 Lessons learned

Our study shows that it is possible to compute the result of an election without leaking any additional information about the original ballots, often at a realistic cost. In this thesis, we provide a toolbox that can be used for this purpose, and apply it to several counting methods. During the process, we made some interesting findings that we give here as the lessons learned.

Think ElGamal. While Paillier is the Swiss-Army knife for MPC, our study has shown that ElGamal can often suffice, even when encrypted integers need to be added or multiplied. This can be a big advantage in terms of efficiency and availability of software libraries.

Rethink the encoding of ballots. The encoding of a ballot can have a huge impact on the cost of the rest of the procedure. For example, encoding integers in their bit representation adds an initial cost that can later save a lot of computation. It is often necessary in the ElGamal setting. The encoding of ballots also offers different tradeoffs in terms of load balance between the voters and the talliers, as seen in the case of the Condorcet methods.

A proof of a shuffle is a versatile tool. The typical use of a proof of a shuffle is inside a mixnet, where some ciphertexts are mixed and re-randomized. However, a proof of a shuffle can also be used to prove the validity of a ballot, in the context of preferential voting. From this remark, we proposed an original usage of the proof of a shuffle in the context of STV and Condorcet voting, which both give two very efficient voting procedures. In the case of Condorcet voting, this gives an efficient solution on the voter-side that is compatible with a homomorphic tally.

Consider the full algorithmic toolbox. When designing an MPC protocol, the constraints are rather not standard. The worst case complexity always needs to be considered, and all the branches need to be visited, just as in the circuit complexity model. In fact, this circuit point of view is highly relevant, and we borrowed some designs from the hardware literature. The depth of the circuit is related to the number of communication rounds; but limits on the fan-in or fan-out of a gate are irrelevant.

Some rather advanced algorithms like the MJ counting functions or the Floyd-Warshall shortest path algorithm can be translated rather easily. On the other hand, some basic tasks can be surprisingly expensive. For instance, many classical algorithms assume that accessing the i^{th} value of an array $T[i]$ takes a constant time, even when i is a computed value. In MPC, this requires a linear time to pass through all the values of T in order to hide the value of i. Another example is the addition of encrypted integers, where the carry propagation can generate a chain of dependencies that translates into a linear number of communication rounds. Breaking the chain of carries as done in hardware circuits allows to reduce this to a logarithmic number of rounds.

Part III

Coercion resistance

Coercion is a common security concern in electronic voting. It occurs when an attacker, the coercer, asks a voter to vote in a specific way, using a threat or a reward. This phenomenon is known to exist in real-world elections, with traditional voting at polling stations. However, an electronic voting system which is not designed to tackle coercion could allow the attacker to coerce a larger number of voters, or to gain a more convincing evidence that the coerced voters actually obeyed. Also, since Internet voting is a remote voting process, this introduces new attacks compared to polling station voting. For instance, the coercer can ask the voter to give all the voting materials that they received. The classical verifiability mechanisms will then provide a proof to the coercer that the voter did not cheat.

In the literature, the most notable approach to address coercion is that of Juels, Catalano and Jakobsson [JCJ05], who gave a formalization of the notion of coercion-resistance as well as the first *coercion-resistant* protocol. Their definition and their protocol, now known as the JCJ protocol, remain the reference for the research on coercion-resistance in electronic voting. In this thesis, we disclose that the original protocol of JCJ is not perfectly coercion-resistant when revoting is allowed. This is because of an issue in the definition of coercion-resistant, which does not allow to properly take revoting into account. In Chapter 7, we present the vulnerability of the JCJ scheme, we evaluate its impact and we propose a new definition of coercion-resistance which better models revoting. Since the JCJ protocol is the basis of most of the academic protocols that aim at achieving coercion-resistance, a large majority of the existing protocols are also concerned with the vulnerability that exists in JCJ. In Chapter 8, we present CHide, which is a variant of the JCJ protocol which achieves our definition of coercion-resistance, and corrects the vulnerability of JCJ.

Chapter 7

Is the JCJ voting system really coercion-resistant?

7.1 The JCJ family

A prominent strategy to address coercion is that of the JCJ family, which is based on the *fake credential* paradigm. The idea is that whatever the coercer might ask the voter to do, it can do it itself when given the credential of the voter. Hence, the strategy of JCJ is to provide a way for the voter to give a *fake* credential to the coercer. The coercer, who votes with the provided credential, has no way to detect whether the latter is valid or not. In order to guarantee that, during the voting phase, the ballots are accepted in the ballot box regardless of their credentials; those which use an invalid or a duplicate one are removed later, during a *cleansing* phase. The output of this cleansing phase is a set of ballots that is tallied in the usual way. The main security feature is that, given a credential and all the publicly available information, the coercer is unable to tell whether the credential is real or fake. At the same time, for the legitimate voters, verifiability is preserved.

7.1.1 Presentation of the JCJ protocol

In the case of the JCJ family, we define a voting system as usual, but we also consider that the voting system must provide a polynomial time probabilistic algorithm Fakecred that allows a voter under coercion to produce a fake credential. Hence, a voting system is a tuple (Setup, Register, Vote, Check, Valid, Fakecred, Tally, Verify). More precisely, the protocol must actually provides an *evasion strategy*, which is a list of instructions that the voter can follow to evade coercion; *i.e.* to be able to deceive the coercer and still vote for the desired voting option. The JCJ voting system consists of the following phases.

Setup. The talliers jointly generate an ElGamal encryption key pk for a group G of prime order q. The resulting public information, such as $\mathsf{pk}, (h_i)_{i=1}^{n_T}$, is published in the public board.

Registration. The registrars jointly compute n_V random credential $c_1, \cdots, c_{n_V}$, where n_V is an eligible voter. For each credential i, they generate a random encryption C_i of c_i, and form the *public roster* $\Pi^R = (C_1, \cdots, C_{n_V})$. Then, whenever an eligible voter authenticates themselves with the registrars, the latter privately send the voter one available credential at random, possibly with designated zero-knowledge proofs (DVZKP) that guarantees the voters that their credential is valid [JSI96]. This DVZKP is such that it can only convince the voter, and a voter under coercion can forge a fake DVZKP for any statement.

Voting. To cast a ballot, a voter encrypts their voting option ν with the public key pk, which gives the ciphertext V (or list / matrix of ciphertexts, depending on the expected format of a ballot). In addition, they also encrypt their credential c, which gives a ciphertext C. They prove the knowledge of ν and c using a PoK, and prove in zero knowledge that ν is a valid voting option, yielding an overall proof π. The resulting ballot $B = (V, C, \pi)$ is sent anonymously to the bulletin board.

Tallying. The tally phase consists of four steps.

1. Ballots with duplicated credentials are detected using Plaintext Equivalence Tests [JJ00] (PET). (We present the PET in Section 2.4.4.) At most one ballot (typically, the last) is kept per credential.

2. The trustees shuffle the remaining ballots, using a mixnet.

3. PET are used again to remove the ballots with invalid credential, that is, whose credential is not present in an encrypted form in Π^R.

4. Finally, each remaining ballot is decrypted so that the result can be computed. (Alternatively, any other tally process could be deployed.)

Each step includes a zero-knowledge proof that the correct operations are performed.

Evading coercion. In the JCJ voting system, a voter under coercion generates a random, fake credential c' (*i.e.* Fakecred is an algorithm that produces a uniformly random credential) and hands this over to the coercer, pretending that it is the real credential obtained during the registration phase. Afterwards, the voter under coercion votes once for the desired voting option (or abstain, depending on their personal preference). Note that if the coercer casts a ballot using the fake credential c', the ballot will be removed at Step **3.** of the tally phase. However, thanks to the mixnet, the coercer is unable to learn that the suppressed was the one it cast with the credential c'.

7.1.2 Some variants of the JCJ voting system

The JCJ scheme was impactful in the literature, and most of the subsequent schemes that addressed coercion-resistance were aimed at improving its scalability, or at least were largely inspired by the JCJ protocol. Civitas [CCM08] is one of the most notable examples, and is widely considered as an important step towards a practical version of JCJ. Among other things, it introduces the notion of ballot blocks, that mitigates the quadratic cost of the cleansing phase. Other attempts were made to improve the efficiency of JCJ. In [SKHS11], Spycher *et al.* claim a linear time cleansing, but this comes with a deterioration of the coercion-resistance. Later on, the same authors proposed other schemes with a clear trade-off between efficiency and coercion-resistance, thanks to anonymity sets [SKHS12]. Other improvements include [AFT08], where Araújo *et al.* propose the AFT scheme to perform the cleansing phase in linear time, and [CH11], which introduces the idea of *over-the-shoulder* coercion-resistance. Note that the AFT scheme was itself the subject of many iterations, see for instance [ARR+10, ABBT16].

In any case, the JCJ protocol and the fake credential paradigm is always the underlying idea behind those proposals.

7.2 Unveiling a shortcoming in JCJ

Despite the JCJ protocol being a central protocol in the literature, we discovered a shortcoming that occurs when revoting is allowed. In the context of coercion-resistance, allowing revoting is a natural counter-measure that can address, for instance, a coercion from a family member or an employer. The idea is that the voter first complies with the coercer, but then revotes using the desired voting option when given a moment of privacy. In particular, the Estonian electronic voting system entirely relies on revoting to mitigate coercion [MM06]. Hence, we consider that it is natural for a coercion-resistant protocol to allow revoting. Note that in [IRRR17, Section 4.4], it was already mentioned that some problems may arise due to revoting in the JCJ scheme.

7.2.1 Leakage in case of revoting

For a verifiable voting system, it seems unavoidable to leak the number of received ballots in the public board. The number of ballots that use a valid credential is also leaked unless a more sophisticated tally methods is used, such as tally-hiding. However, the JCJ protocol leaks the following additional information:

- n_B, the total number of received ballots;
- n_V, the total number of valid (and counted) ballots;
- n_R, the total number of revotes;
- the complete distribution of revotes per (encrypted) credential (hence, for all k, the number of credentials used to revote k times).

This can be exploited by a coercer to detect when a coerced voter disobeys. Indeed, there is no reason to assume that revoting is independent from the choice of the candidate. On the contrary, revoting is often due to voters changing their mind between candidates, for instance due to some late announcements in the press.

An attack against coercion-resistance. To illustrate that the leakage can indeed be exploited in some cases, we consider an extreme case, with two candidates A and B. Suppose that voters voting for A do not revote while those voting for B always revote, exactly once. We denote r_A (resp. r_B) the number of votes for A (resp. B), and we suppose that those information can be deduced by the result of the tally, which is often the case except if a tally-hiding strategy is used. Due to the considered revoting behaviors, the number of revotes n_R corresponds to the number of votes for B sent by the honest voters.

Assume now that Alice wants to vote for B but is instructed by her coercer to vote for A.

- If Alice obeys, the coercer will observe $r_B = n_R$.
- If Alice disobeys and casts one ballot for B, the coercer will observe that $r_B = n_R + 1$.

Hence the coercer will detect that Alice has disobeyed, which breaks coercion-resistance.

One could argue that Alice should follow a different evasion strategy and cast one ballot if she votes for A and two if she votes for B. This does not work either. Indeed, assume now that Alice wants to vote for A, but is instructed to vote for B.

- If she obeys, she gives her real credential c to her coercer. The latter then casts *exactly one* ballot for B using c.

- Otherwise, she provides a fake credential c', that the coercer uses to vote for B. Alice then votes for A using c.

In the first case, $r_B = n_R + 1$ but in the second case, $r_B = n_R$. Once again, the coercer is able to detect that Alice disobeyed and coercion-resistance is lost. More generally, it seems difficult to come up with a simple evasion strategy that fixes the issue in JCJ; see the discussion in Section 7.6.

7.3 The impact on coercion-resistance

In the previous section, we explained the leakage of the JCJ protocol and we illustrated, on an extreme scenario, how this can be exploited to completely break coercion-resistance. In this section, we estimate the impact of the leakage in more realistic scenarios. For this purpose, we use the framework of [KTV10a] which allows to quantify the coercion level of a voting protocol.

7.3.1 Quantifying coercion-resistance

We consider n_V voters, among which one is under coercion. The others are supposed honest and independent. They choose a voting option among $C+1$ possibilities, which includes abstention and blank voting. We suppose that the choices follow a probability distribution $(P_0, \dots, P_C)$, where P_0 is the probability to abstain. Let α be the voting option corresponding to the intention of the coerced voter, and β be the one that is the instruction of the coercer. The coerced voter either disobeys, gives a fake credential and votes with the option α (the evasion strategy does not imply any revote), or obeys and gives their real credential which the coercer uses to vote with option β. The coercer must decide whether the voter obeyed or not, given only the result.

The ideal result is $\mathsf{R}^{\text{Ideal}} = \overrightarrow{res} = (\mathsf{res}_0, \cdots, \mathsf{res}_C)$, the number of voters who opted for each option. In JCJ, however, the real result R^{Real} is $\overrightarrow{res}$, as well as, for all k, the number of voters who revoted k times. In addition, both results should also include the number of ballots that are removed during the cleansing phase. However, to focus on the leakage of JCJ, we assume that a large and unpredictable number of dummy ballots are cast, so that the adversary cannot gain information by observing this number. This approximation is necessary to use the framework of [KTV10a], which does not model the possibility to cast a ballot with an invalid credential in the ideal setting.

We now instantiate [KTV10a] in our scenario. To simplify the analysis, we assume that a voter revotes at most once, so that $\mathsf{R}^{\text{Real}} = (\overrightarrow{res}, n_R)$, where n_R is the total number of revotes. We also assume that all the parties are honest except for the coercer, and that the cryptography is perfect, so that the coercer does not learn any other information than the result. With these assumptions, we define the real and ideal games, where the behavior of the coerced voter is decided at random (the coerced voter obeys with probability 1/2 and tries to evade coercion with probability 1/2). The coercer wins the real (resp. ideal) game if they correctly guess the behavior of the coerced voter given the real (resp. ideal) result. For $g \in \{\text{Real}, \text{Ideal}\}$ and for a pair (α, β), we denote $W^g_{\alpha,\beta}$ the event when the coercer wins the game. We denote $\delta^g_{\alpha,\beta} = 2|\Pr(W^g_{\alpha,\beta}) - 1/2|$, and $\delta^g = \max_{\alpha,\beta} \delta^g_{\alpha,\beta}$. We call δ^{Real} (resp. δ^{Ideal}) the coercion level of the real (resp. ideal) game.

We denote by $\Pr(\mathsf{R}^g|\alpha)$ (resp. $\Pr(\mathsf{R}^g|\beta)$), the probability that the result R^g is obtained, assuming the voter votes for α (resp. obeys the coercer and votes for β). According to [KTV10a], the best strategy for the coercer is to assume that the voter obeyed if and only if $\Pr(\mathsf{R}^g|\beta) \geq$

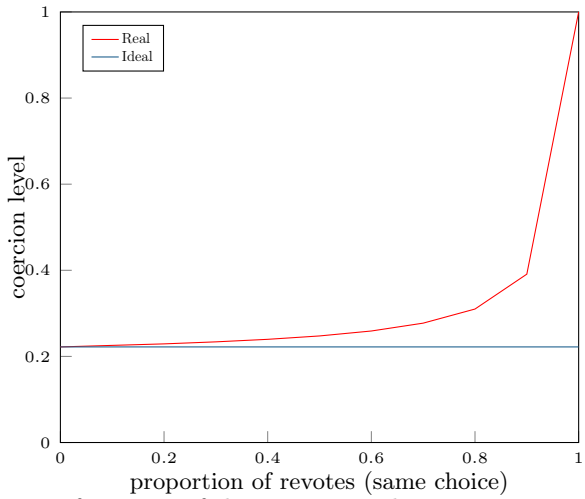


Figure 20: Coercion levels as a function of the revote, with 20 voters, 2 candidates, 30% abstention and a 50%-50% distribution of votes between the candidates.

$\Pr(\mathsf{R}^g|\alpha)$. This gives a close formula for the coercion level which can be written as

$$\delta^g = \max_{(\alpha,\beta)} \sum_{\mathsf{R}^g \in M_{\alpha,\beta}} \Pr(\mathsf{R}^g|\beta) - \Pr(\mathsf{R}^g|\alpha), \tag{3}$$

where $M_{\alpha,\beta}$ is the set of all possible results R^g such that $\Pr(\mathsf{R}^g|\beta) \geq \Pr(\mathsf{R}^g|\alpha)$.

In the remaining of this section, we compare δ^{Real} and δ^{Ideal} in two scenarios, where external events provoke many revotes. In these scenarios, we assume that there are two candidates A and B, no blank vote, but the possibility to abstain or to revote once. In Appendix D, we explain in more details how the different figures of this section were obtained.

7.3.2 The technical incident scenario

In general, we can expect revoting to be rare. This is something that is not allowed in classical paper-based elections, so that in a context where electronic voting is recent, voters will not be using this possibility. Even in a country such as Estonia, where revoting is available for Internet voters since 2005, a recent study revealed a revoting rate of about 2% [ESWV22].

However, much more revotes could occur if an announcement reveals a suspicion of a technical incident, and encourages the voters to revote to be on the safe side. In this case, many voters could be inclined to revote with the same voting option, which would seem harmless if they are not aware of the weaknesses of JCJ. Note that the coercer could be the source of such an announcement, and spread fake news about the necessity to revote.

In Fig. 20, we consider this situation where a proportion of voters (who already voted) revote for the same voting option.

We plot the coercion level in both the real and ideal settings when the proportion x of revotes ranges from 0 to 1. When $x = 0$, both coercion levels are the same since there is no revote. However, when $x = 1$, there is no coercion-resistance in JCJ because the coerced voter would be the only one to cast a ballot without revoting. Note that the ideal coercion level remains constant since the overall probability to choose each voting option is unaffected by x.

7.3.3 A discredit in the press

We still consider two candidates A and B, and in this scenario, we assume that during the period of the voting phase, the candidate A is discredited by an announcement in the press. As a consequence, some proportion of the voters who initially voted for A will change their mind and revote for B. For simplicity, we assume that no revote occurs that is not due to this event.

Such discredits have happened in the past. For instance, Dominique Strauss-Kahn, a former IMF managing director, was highly expected to become the next French president in 2012. However, due to an accusation of sexual assault, his political party chose to support another candidate. This occurred before the time of the election, and no electronic voting was involved. We can also mention the 2022 Tory leadership election for the succession of Boris Johnson; the voters could vote by Internet, and revote was initially authorized (before a security concern forced the organizers to forbid it). The duration was more than a month, which is more than enough for a discrediting event to occur (for this election, it did not occur).

To study the potential impact of JCJ's leakage in such a scenario, we first, we fix a small number of voters, so that the effect is more visible, and we study the influence of the other parameters.

In Fig. 21, we plot the real and ideal coercion levels as the proportion x of voters who change their mind from A to B ranges from 0 to 1. When $x = 0$, there is no difference since there is no revote. When $x = 1$, there is no difference either since nobody votes for A anymore, so that there is no coercion-resistance in both the real and ideal games (note that this is because there are only two candidates; if there are more candidates, there would be no coercion-resistance in the real game but still the coercion level of the ideal game would remain reasonable: this is the scenario of Section 7.2.1). However, a non-negligible difference can be observed for the intermediate values of x.

In Fig. 22, we plot the real and ideal coercion levels with a fixed value of $x = 0.3$ (*i.e.* 30% of the voters who voted for A revote for B) and we let the initial proportion p in favor of A vary from 0 to 1. When $p = 0$, everyone votes for B so that there is no coercion-resistance. When p is large, we get close to the scenario presented in Section 7.2.1, so that there is no coercion-resistance in the real game while the ideal game still offers some coercion-resistance.

In Fig. 23, we plot the real and ideal coercion levels with fixed $x = 0.3$ and $p = 0.7$ and we let the abstention rate P_0 range between 0 and 1. When $P_0 = 0$, there is no coercion-resistance because forced-abstention attacks are trivial; similarly, there is no coercion-resistance when $P_0 = 1$. However, a non-negligible difference can be observed for the intermediate values.

Finally, in Fig. 24, we plot the real and ideal coercion levels with fixed $x = 0.3$, $p = 0.7$ and $P_0 = 0.3$, for a number of uncoerced honest voters equals to $16, 32, 64, 128, 256, 512$ and 1024. This shows that the difference between both coercion levels remains non-negligible even when the number of voters is large. An asymptotic analysis (see *e.g.* [Dav22]) reveals that the coercion level decreases in $1/\sqrt{n_V}$.

7.4 Defining coercion-resistance

One of the reasons why the weakness of JCJ was not discovered so far is because the original definition of coercion-resistance is flawed, and does not consider revoting. In this section, we present the original definition of JCJ, we mention another existing definition and we propose a new definition of coercion-resistance.

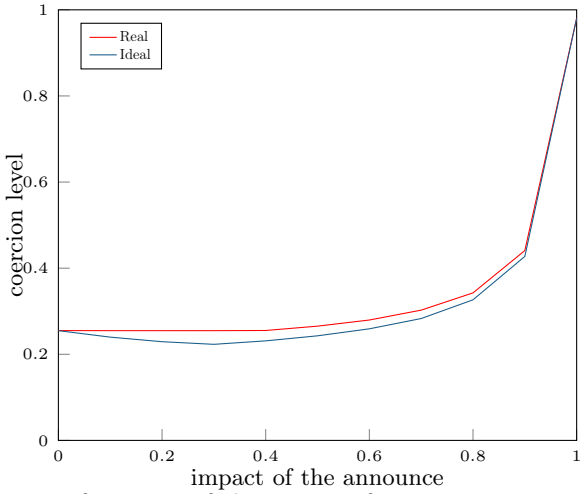


Figure 21: Coercion levels as a function of the impact for 20 voters, 2 candidates, 30% abstention and a 70%-30% distribution between the candidates.

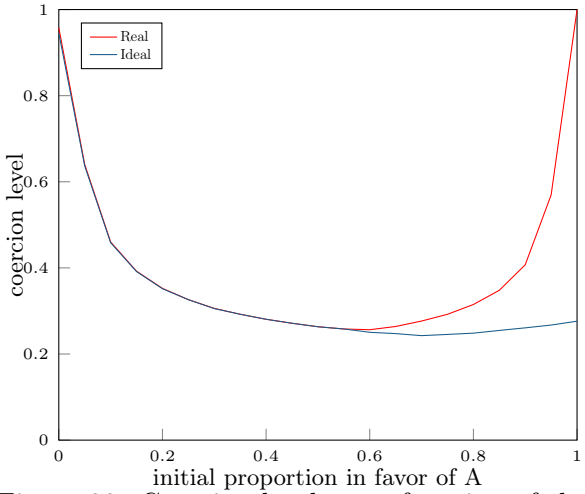


Figure 22: Coercion levels as a function of the proportion in favor of A for 20 voters, 2 candidates and 30% abstention.

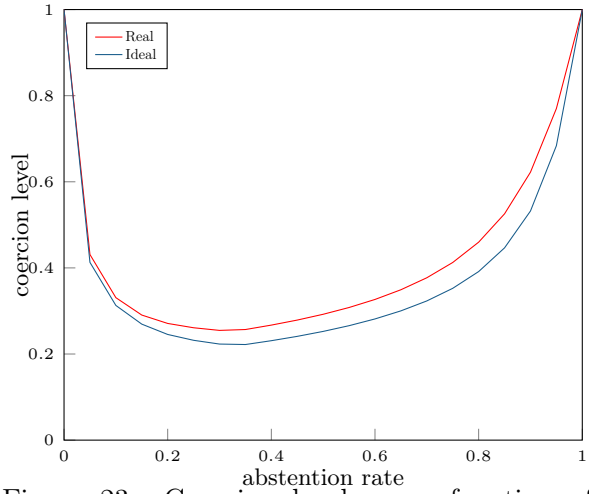


Figure 23: Coercion levels as a function of the abstention for 20 voters, 2 candidates, 21% revotes and a distribution of 70%-30% between the candidates.

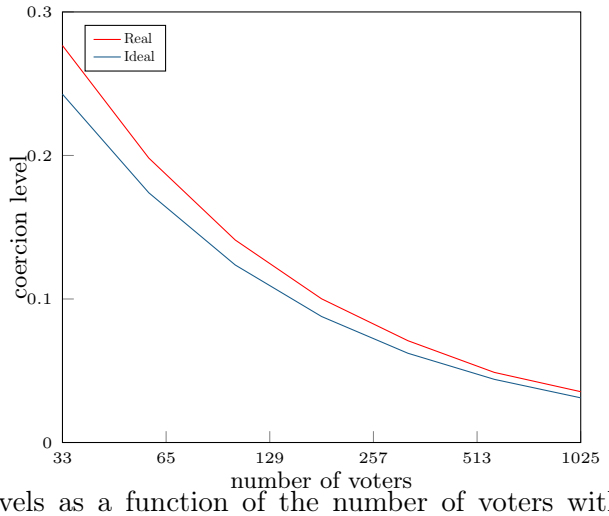


Figure 24: Coercion levels as a function of the number of voters with 30% abstention, 21% revotes and a distribution of 70%-30% between the candidates.

7.4.1 The original definition of JCJ

The intuition of the JCJ definition of coercion-resistance is that an adversary must not be able to guess whether a coerced voter obeyed or evaded coercion. When the voter obeys ($b = 1$ in the definition), they give their real credential and abstain from doing any other action. Note that a coercer may ask the voter to cast some specific vote or to perform some specific computations, but this is not considered in the definition as the adversary might as well do it itself, with the given credential. When the voter evades ($b = 0$ in the definition), they give a fake credential and cast a single vote for the desired voting option (or abstain, depending on their personal choice).

This yields the game $\mathsf{Real}^{\mathsf{CR}}_{\mathsf{JCJ}}$ presented in Algorithm 116. Voting choices are represented as integers between 1 and n_C, and ϕ represents the choice to abstain. During this game, the adversary selects the set of corrupted voters. It is given the corresponding private credentials as well as all the public roster Π^R (*i.e.* the encrypted credentials in the JCJ protocol, which are included in the transcript Π^R of the registration). It then chooses (j, α), where j denotes the voter under coercion and α their desired voting option. The evasion strategy is modeled in lines 10 and 12: when the voter disobeys, they create a fake credential and cast a vote for α (or abstain if $\alpha = \phi$). Otherwise, they give their real credential.

In the definition of JCJ, the honest voters vote according to a distribution which depends on the number of options n_C and returns a value that may be:

- any valid vote $\nu \in [1, n_C]$;
- ϕ, which represents abstention;
- κ, which represents casting a vote with a fake credential.

We extend the Vote function to votes equal to κ as follows.

$$\mathsf{Vote}_{\mathsf{pk}}(\kappa, c) = \mathsf{Vote}_{\mathsf{pk}}(\nu, \tilde{c}),$$

where $\tilde{c} = \text{Fakecred}(c)$ and ν is sampled from $[1, n_C]$.

It is worth noting that the advantage of an adversary in game $\mathsf{Real}^{\mathsf{CR}}_{\mathsf{JCJ}}$ will always be non negligible since one can compare the result of the tally with the expected result, given the

Algorithm 99: $\mathsf{Real}^{\mathsf{CR}}_{\mathsf{JCJ}}$

Requires: $\mathbb{A}, \lambda, n_T, t, n_V, n_A, n_C, \mathcal{D}$

$\mathsf{pk}, (s_i, h_i)_{i=1}^{n_T}, \Pi^S \longleftarrow \mathsf{Setup}(\lambda, n_T, t)$;
$(c_i, \pi_i), \Pi^R \longleftarrow \mathsf{Register}(\mathsf{pk}, n_V)$;
$\mathsf{PB} \longleftarrow \Pi^S || \Pi^R$;
$A \longleftarrow \mathbb{A}(\mathsf{PB})$;
$(j, \alpha) \longleftarrow \mathbb{A}(\{c_i; i \in A\})$;
if $|A| \neq n_A \vee j \notin [1, n_V] \backslash A \vee \alpha \notin [1, n_C] \bigcup \{\phi\}$ **then return** 0 ;
$b \xleftarrow{\$} \{0, 1\}$;
$\tilde{c} \longleftarrow c_j$;
if $b = 0$ **then**
  $\tilde{c} \longleftarrow \text{Fakecred}(c_j)$;
  if $\alpha \neq \phi$ **then**
    $\mathsf{PB} \longleftarrow \mathsf{PB} \bigcup \{\mathsf{Vote}_{\mathsf{pk}}(c_j, \alpha)\}$;
for $i \in [1, n_V] \backslash (A \bigcup \{j\})$ **do**
  $\nu_i \longleftarrow \mathcal{D}_{n_C}()$;
  if $\nu_i \neq \phi$ **then**
    $\mathsf{PB} \longleftarrow \mathsf{PB} \bigcup \{\mathsf{Vote}_{\mathsf{pk}}(c_i, \nu_i)\}$
$\mathsf{PB} \longleftarrow \mathsf{PB} \bigcup \mathbb{A}(\tilde{c}, \mathsf{PB})$;
;
;
$X, \Pi \longleftarrow P^{\mathbb{A}}_{\mathsf{tally}}(\mathsf{PB}, \Pi^R, \mathsf{pk}, \{h_i, s_i\}, t)$;
$b' \longleftarrow \mathbb{A}()$;
if $b' = b$ **then return** 1 **else return** 0;

Algorithm 100: $\mathsf{Ideal}^{\mathsf{CR}}_{\mathsf{JCJ}}$

Requires: $\mathbb{A}, \lambda, n_V, n_A, n_C, \mathcal{D}$

;
;
$D \longleftarrow \varnothing$;
$A \longleftarrow \mathbb{A}(\lambda)$;
$(j, \alpha) \longleftarrow \mathbb{A}()$;
if $|A| \neq n_A \vee j \notin [1, n_V] \backslash A \vee \alpha \notin [1, n_C] \bigcup \{\phi\}$ **then return** 0 ;
$b \xleftarrow{\$} \{0, 1\}$;
;
if $b = 0 \wedge \alpha \neq \phi$ **then**
  $D \longleftarrow D \bigcup \{\alpha\}$;
;
;
for $i \in [1, n_V] \backslash (A \bigcup \{j\})$ **do**
  $\nu_i \longleftarrow \mathcal{D}_{n_C}()$;
  if $\nu_i \notin \{\phi, \lambda\}$ **then**
    $D = D \bigcup \{\nu_i\}$
$(\nu_i)_{i \in A}, \beta \longleftarrow \mathbb{A}()$;
if $b = 1 \wedge \beta \in [1, n_C]$ **then**
  $D \longleftarrow D \bigcup \{\beta\}$;
$X \leftarrow \mathsf{count}(D \bigcup \{\nu_i \mid i \in A, \nu_i \in [1, n_C]\})$;
$b' \longleftarrow \mathbb{A}(X)$;
if $b' = b$ **then return** 1 **else return** 0;

Figure 25: JCJ definition of coercion resistance. λ is the security parameter, n_T the number of talliers, t the threshold, n_V the number of voters, n_A the number of corrupted voters, n_C the number of voting options and $\mathcal{D}$ the distribution of votes.

distribution $\mathcal{D}$ of the voting intentions. For example, if the adversary wants to cast a vote for a very unlikely candidate, they may observe cases where the latter does not get a single vote in the result, which is a clear indication that the coerced voter disobeyed. Hence, the JCJ definition compares the advantage of an adversary in game $\mathsf{Real}^{\mathsf{CR}}_{\mathsf{JCJ}}$ with the one in an ideal game $\mathsf{Ideal}^{\mathsf{CR}}_{\mathsf{JCJ}}$, where there is no other information than what is unavoidably leaked, that is, the result. The game $\mathsf{Ideal}^{\mathsf{CR}}_{\mathsf{JCJ}}$ is presented in Algorithm 117. Compared to the original definition, we present a slightly modified version that reasons on the clear votes only. This simplifies the understanding by focusing on the information given to the adversary. All our claims and remarks hold on the original definition as well.

Definition 20 (adapted from [JCJ05])**.** *A voting system is* JCJ-coercion resistant *if for all PPT adversary* $\mathbb{A}$*, for all parameters* n_T, t, n_V, n_A, n_C*, and for all distributions* $\mathcal{D}$*, there exists a PPT adversary* $\mathbb{B}$ *and a negligible function* μ *such that*

$$|\Pr(\mathsf{Ideal}^{\mathsf{CR}}_{\mathsf{JCJ}}(\mathbb{B}, \lambda, n_V, n_A, n_C, \mathcal{D}) = 1) - \Pr(\mathsf{Real}^{\mathsf{CR}}_{\mathsf{JCJ}}(\mathbb{A}, \lambda, n_T, t, n_V, n_A, n_C, \mathcal{D}) = 1)| \leq \mu(\lambda).$$

As noticed in [HS19], this definition cannot be realized by a scheme which uses a public board. Indeed, in the real game, the adversary observes the length n_B of the board which corresponds to the total number of ballots cast by non-corrupted voters. Then the adversary learns the result and in particular its size $|r|$, that is the number of valid ballots counted. Hence the total number $\Delta = n_B - |r|$ of ballots discarded can be deduced, which is not available in the ideal game $\mathsf{Ideal}^{\mathsf{CR}}_{\mathsf{JCJ}}$. The value of Δ can be compared with its expected number, according to the distribution $\mathcal{D}$. Since there is an additional ballot discarded (the one of the coercer) when the voter evades coercion, the adversary has a non-negligible advantage in the real game. For instance, if $\mathcal{D}$ is such that no voters cast a ballot with an invalid credential, either $n_B = |r|$, which means that the adversary's ballot has been counted, or $n_B = |r| + 1$, meaning that the adversary's ballot has been discarded and that the voter has disobeyed. Of course, the same issue applies to the JCJ definition as stated in [JCJ05].

The authors of [HS19] proposed a patch to the issue they discovered: the length of the board should be given to the adversary in the ideal game as well. Intuitively, this corresponds to rewriting line 17 of Algorithm 117 as $(\nu_i)_{i\in V}, \alpha \longleftarrow \mathbb{A}(|D|)$. However, this still does not allow to detect the leakage of the JCJ protocol during the tally. Indeed, the distribution $\mathcal{D}$ fails to model several aspects:

- First, the addition of a ballot with an invalid credential only happens when a honest voter sacrifices their own vote. This is unlikely in practice, and does not model ballots sent by non-eligible voters (for instance, by the authorities).

- Second, revoting is not considered at all in $\mathcal{D}$, which explains why the leakage of the JCJ protocol was not detected.

A final remark about the definition of JCJ concerns its underlying trust assumptions, which are slightly different from the usual. For clarity, we recall them here. First of all, it is assumed that all the registrars are honest and that the adversary is inactive during the registration phase (or, alternatively, the registration is untappable, which means that the communications between the registrars and the voter leave no trace). Second, the adversary can only corrupt a minority of decryption authorities. Also, ballots are cast through anonymous channels. Finally, the bulletin board is honest.

7.4.2 Our definition of coercion-resistance

Apart from the definition of JCJ, there are other definitions in the literature (see [HS19] for a survey). In particular, we already used the framework of [KTV10a] (KTV) in Section 7.3. In this framework, a *quantitative* definition of coercion-resistance is proposed, where the notion of δ-coercion-resistance comes with two conditions: first, the coerced voter must have a strategy to meet their objective with overwhelming probability; second, the adversary cannot decide, with an advantage greater than δ, whether the voter used this strategy or forwarded all received messages (including their credential).

The KTV definition is abstract. To use it, it is necessary to model the voting protocol, its participants and the evasion strategy. In addition, it does not say much about how ballots sent with an invalid credential should be handled since the honest participants are assumed to vote following a fixed distribution of valid voting options. Finally, it does not tell if a specific δ is acceptable or not. To address this, we propose our own definition which can be seen as an instantiation of KTV, where δ is shown to be minimal, that is, not greater than that of an ideal protocol.

If we compare the advantage of the adversary in the real game with its advantage in the ideal one, we need to cover a large family of vote distributions. Otherwise, we may miss security flaws. In particular, we need to cover cases explicitly planned by the protocol such as revote and addition of ballots with fake credentials.

Therefore, given a set S of unique identifiers and the number n_C of voting options (excluding abstention), we consider a distribution $\mathcal{B}(S, n_C)$ of sequences of pairs of the form (j, ν) where $\nu \in [1, n_C]$ represents a vote and j represents either a valid voter (when $j \in S$) or a fake voter, with a fake credential. Typically, if A is the set of corrupted voters, $S = [1, n_V] \backslash A$. To avoid collisions with identifiers which may be in A, we consider that any $j \notin S$ holds a negative value. The distribution $\mathcal{B}$ captures the abstention of a voter j with the absence of a couple of the form $(j, *)$. It models both revoting and the addition of fake ballots, typically by authorities:

- revoting is reflected in $\mathcal{B}$ by the fact that a voter may appear several times in the same sequence;
- fake ballots are modeled by pairs (j, ν) where $j \notin S$. They may be added by authorities or voters. Note that $\mathcal{B}$ also models the case of a revote with a fake credential.

For example, in the sequence $(1,1),(2,1),(1,2),(-1,2),(1,1)$ with $n_V = 3$, we have three voters V_1, V_2 and V_3. V_1 first votes 1, V_2 votes 1, then V_1 revotes for 2, then a fake vote for 2 is added, then V_1 changes back her vote to 1. V_3 chooses to abstain.

Our $\mathsf{Real}^{\mathsf{CR}}$ game, defined in Algorithm 101, is similar to $\mathsf{Real}^{\mathsf{CR}}_{\mathsf{JCJ}}$. Votes are drawn according to $\mathcal{B}([1, n_V] \backslash A, n_C)$, yielding a sequence B. It typically contains pairs (i, ν) with $i < 0$, which corresponds to the addition of ballots with fake credentials. For such a pair, we therefore generate a fake credential at lines 10-11. Just as in the definition of JCJ, the adversary must guess a bit b. If $b = 1$, the coerced voter j obeys, hence any vote from j is removed from B and the real credential is provided to the adversary. If $b = 0$, the voter follows the evasion strategy, namely they cast one vote for β (if $\beta \neq \phi$) and provides a fake credential. Hence the votes from j in $\mathcal{B}$ are replaced by a single vote for β (if $\beta \neq \phi$). Then ballots are added according to $\mathcal{B}$. They correspond either to real or fake votes (or revotes). Compared to the original JCJ definition, we also slightly improve the power of the adversary by letting them observe the board after each vote and add ballots if they want to, which better reflects the reality. Also, recall that the notation $\mathsf{Tally}^{\mathbb{A}}$ is used to capture the fact that the adversary is active during the tally phase, and can impersonate the corrupted talliers.

Algorithm 101: $\mathsf{Real}^{\mathsf{CR}}$

Requires: $\mathbb{A}, \lambda, n_T, C_t, n_V, n_A, n_C, \mathcal{B}$

$\mathsf{pk}, (s_i, h_i)_{i=1}^{n_T}, \Pi^S \longleftarrow \mathsf{Setup}(\lambda, n_T, t)$;
$(c_i, \pi_i), \Pi^R \longleftarrow \mathsf{Register}(\mathsf{pk}, n_V)$;
$\mathsf{PB} \longleftarrow \Pi^S || \Pi^R$;
$A \longleftarrow \mathbb{A}(\mathsf{PB}, \{s_i \mid i \in C_t\})$ (* corrupt voters *);
$(j, \alpha) \longleftarrow \mathbb{A}(\{c_i; i \in A\})$;
(* coerces j who has the intention α *)
if $|A| \neq n_A \vee j \notin [1, n_V] \backslash A \vee \alpha \notin [1, n_C] \bigcup \{\phi\}$ **then return** 0 ;
$B \longleftarrow \mathcal{B}([1, n_V] \backslash A, n_C)$;
(* samples a sequence of pairs (i, ν_i) with $i \in ([1, n_V] \backslash A) \bigcup \{n \mid n < 0\}$ *)
for $(i, *) \in B$, $i \notin [1, n_V]$ **do**
    $c_i \longleftarrow$ Fakecred();
    (* this captures dummy ballots *)
$b \xleftarrow{\$} \{0, 1\}$;
$\tilde{c} \longleftarrow c_j$;
if $b = 1$ **then** Remove all $(j, *) \in B$;
else
    Remove all $(j, *) \in B$ but the last, which is replaced by (j, α) if $\alpha \neq \phi$ and removed otherwise;
    $\tilde{c} \longleftarrow \mathrm{Fakecred}(c_j)$;
$\mathbb{A}(\tilde{c})$ (* $\mathbb{A}$ learns $\tilde{c}$ *);
for $(i, \nu_i) \in B$ *(in this order)* **do**
    $\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(\mathsf{PB})$ (* casts valid ballots *);
    $\mathsf{PB} \leftarrow \mathsf{PB} \bigcup \{\mathsf{Vote}_{\mathsf{pk}}(c_i, \nu_i)\}$;
$\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(\mathsf{PB}, \texttt{"end for"})$;
$X, \Pi \leftarrow \mathsf{Tally}^{\mathbb{A}}(\mathsf{PB}, \mathsf{pk}, \{s_i\})$;
$b' \longleftarrow \mathbb{A}()$;
if $b' = b$ **then return** 1 **else return** 0;

Algorithm 102: $\mathsf{Ideal}^{\mathsf{CR}}$

Requires: $\mathbb{A}, \lambda, n_V, n_A, n_C, \mathcal{B}$

;
;
;
$A \longleftarrow \mathbb{A}(\lambda)$ (* corrupt voters *);
$(j, \alpha) \longleftarrow \mathbb{A}()$;
(* coerces j who has the intention α *);
if $|A| \neq n_A \vee j \notin [1, n_V] \backslash A \vee \alpha \notin [1, n_C] \bigcup \{\phi\}$ **then return** 0 ;
$B \longleftarrow \mathcal{B}([1, n_V] \backslash A, n_C)$;
(* samples a sequence of pairs (i, ν_i) with $i \in ([1, n_V] \backslash A) \bigcup \{n \mid n < 0\}$ *);
;
;
;
$b \xleftarrow{\$} \{0, 1\}$;
;
if $b = 1$ **then** Remove all $(j, *) \in B$;
else
    Remove all $(j, *) \in B$ but the last, which is replaced by (j, α) if $\alpha \neq \phi$ and removed otherwise;
;
;
$(\nu_i)_{i \in A}, \beta \longleftarrow \mathbb{A}(|B|)$;
if $(b = 1) \wedge (\beta \in [1, n_C])$ **then**
    $B \longleftarrow B \bigcup \{(j, \beta)\}$;
$B \longleftarrow B \bigcup \{(i, \nu_i) \mid i \in A, \nu_i \in [1, n_C]\}$;
$X \longleftarrow \mathsf{count}(\mathsf{cleanse}(B))$;
$b' \longleftarrow \mathbb{A}(X)$;
if $b' = b$ **then return** 1 **else return** 0;

Figure 26: Definition of coercion-resistance. λ is the security parameter, n_T the number of talliers, t the threshold, C_t the set of the corrupted talliers, n_V the number of voters, n_A the number of corrupted voters, n_C the number of voting options and $\mathcal{B}$ the distribution.

Again, the advantage of the adversary in the real game is compared with its advantage in an ideal game $\mathsf{Ideal}^{\mathsf{CR}}$ (see Algorithm 119), where the adversary can only observe the number of ballots and the result. The latter is computed from the considered counting function count, but also from a function $\mathsf{cleanse}$ that removes votes from invalid voters $j \notin [1, n_V]$ and that takes care of revotes according to the policy (typically, the last vote is kept).

Definition 21. *A voting system is* coercion resistant *if for all PPT adversary $\mathbb{A}$, for all parameters n_T, t, n_V, n_A, n_C, for all subset $C_t \subset [1, n_T]$ of size at most t and for all distribution $\mathcal{B}$, there exists a PPT adversary $\mathbb{B}$ and a negligible function μ such that*

$$|\Pr(\mathsf{Ideal}^{\mathsf{CR}}(\mathbb{B}, \lambda, n_V, n_A, n_C, \mathcal{B}) = 1) - \Pr(\mathsf{Real}^{\mathsf{CR}}(\mathbb{A}, \lambda, n_T, C_t, n_V, n_A, n_C, \mathcal{B}) = 1)| \leq \mu(\lambda).$$

The main difference between our definition and the original one is that we consider a larger family of distributions, which allows to analyze a protocol in the context of revotes and fake ballots.

Another difference is that the adversary shall not gain any advantage for *any* distribution $\mathcal{B}$, while the JCJ definition defines coercion-resistance *with respect to* a particular distribution. This is preferable since a protocol should be as secure as the ideal one, whatever the considered distribution. It is counter-intuitive to design a cryptographic protocol that resists only for particular distributions. Of course, it makes sense to analyze the exact advantage in the ideal game for a particular distribution, and devise whether voters are reasonably protected in that case of not. But the cryptographic protocol itself should be as solid as the ideal one nevertheless.

7.5 A description of the leakage in JCJ

Because of a leakage during the tally phase, the JCJ protocol does not verify Definition 21 and is not fully coercion-resistant. In Section 7.3.1, we quantified the impact of the leakage in some realistic scenarios; however, it is also interesting to qualify its exact.

In Definition 22, we define a weaker notion of coercion-resistance, where the votes are no longer considered perfectly anonymous. Rather, we consider that the adversary is able to detect when a ballot is a revote or not. More precisely, in the ideal game, we generate a pseudonym c_i for each voter i occurring in B. The first voter who votes is given pseudonym 1, the second one is given pseudonym 2, and so on. The rest of the game is left unchanged except that at the end (line 25 of Algorithm 119), the adversary is given an additional information, that we denote $\boldsymbol{I}$, which tells which ballots correspond to the same credential (invalid or not). More formally, $\boldsymbol{I}$ is a sequence of pseudonyms, each corresponding to the voter that has voted at this step. This is exactly what can be observed in JCJ. See Algorithm 120 for a description of the ideal game $\mathsf{Ideal}_W^{\mathsf{CR}}$. As for the real game, it is left unchanged.

Definition 22. *A voting system is* weakly coercion resistant *if for all PPT adversary $\mathbb{A}$, for all parameters n_T, t, n_V, n_A, n_C, and for all distribution $\mathcal{B}$, there exists a PPT adversary $\mathbb{B}$ and a negligible function μ such that*

$$|\Pr(\mathsf{Ideal}_W^{\mathsf{CR}}(\mathbb{B}, \lambda, n_V, n_A, n_C, \mathcal{B}) = 1) - \Pr(\mathsf{Real}^{\mathsf{CR}}(\mathbb{A}, \lambda, n_T, t, n_V, n_A, n_C, \mathcal{B}) = 1)| \leq \mu(\lambda)\,.$$

Assuming that the cryptography used in JCJ is perfect (*i.e.* that the tally protocol is SUC-secure), it is possible to prove that JCJ satisfies this relaxed version of coercion-resistance under JCJ's trust assumptions: the registration is untappable, the voting channel is anonymous, up to a threshold t of talliers is corrupted, and the registrars are honest. In reality, the tally protocol

Algorithm 103: $\mathsf{Ideal}_W^{\mathsf{CR}}$

Requires: $\mathbb{A}, \lambda, n_V, n_A, n_C, \mathcal{B}$

$k \longleftarrow 1$;
$A \longleftarrow \mathbb{A}(\lambda)$ (* corrupt voters *);
$(j, \alpha) \longleftarrow \mathbb{A}(\{c_i; i \in A\})$;
(* coerce a voter j who has the intention α *);
if $|A| \neq n_A \vee j \notin [1, n_V] \backslash A \vee \alpha \notin [1, n_C] \bigcup \{\phi\}$ **then return** 0 ;
$B \longleftarrow \mathcal{B}([1, n_V] \backslash A, n_C)$;
(* samples a sequence of pairs (i, ν_i) with $i \in ([1, n_V] \backslash A) \bigcup \{n \mid n < 0\}$ *)
for $(i, *) \in B$ **do**
    if $c_i = \perp$ **then**
        $c_i \longleftarrow k$; $k \longleftarrow k + 1$;
$b \xleftarrow{\$} \{0, 1\}$;
;
if $b = 1$ **then** Remove all $(j, *) \in B$;
else
    Remove all $(j, *) \in B$ but the last, which is replaced by (j, α) if $\alpha \neq \phi$ and removed otherwise;
$(\nu_i)_{i \in A}, \beta \longleftarrow \mathbb{A}(|B|)$;
$\boldsymbol{I} \longleftarrow \{c_i; (i, *) \in B\}$ (* in this order, with duplicates *) ;
if $(b = 1) \wedge (\beta \in [1, n_C])$ **then** $B \longleftarrow B \bigcup \{(j, \beta)\}$;
$B \longleftarrow B \bigcup \{(i, \nu_i) \mid i \in A, \nu_i \in [1, n_C]\}$;
$X \longleftarrow \mathsf{count}(\mathsf{cleanse}(B))$;
$b' \longleftarrow \mathbb{A}(X, \boldsymbol{I})$;
if $b' = b$ **then return** 1 **else return** 0;

used in JCJ is not SUC-secure since it uses a reencryption mixnet (there is currently no known UC-secure reencryption mixnet). Also, the PET originally used in JCJ was not verifiable when all the participants are corrupted [MPT20]. While the fix proposed in [MPT20] is not proven SUC-secure, we consider that proving the security of JCJ's cryptographic primitives is out of scope for this thesis.

Theorem 9. *Under the DDH assumption and in the ROM, if the* Tally *protocol of JCJ is SUC-secure, then the JCJ protocol is weakly coercion-resistant.*

In Chapter 8, we present a variant of the JCJ protocol that is coercion-resistant, and prove its coercion-resistance in Section 8.2.1. The proof of Theorem 9, is extremely similar to the proof of Section 8.2.1. The main difference is the beginning of the proof, where the ideal result is $\boldsymbol{X}, \boldsymbol{I}$ instead of just $\boldsymbol{X}$. Apart from that, we use the exact same transitions and arguments. Therefore, we only provide a short proof where we reproduce the main transitions. See Section 8.2.1 for a complete proof where each transition is justified.

Proof sketch. Consider an adversary $\mathbb{A}_0$ for the real game. We construct a succession of game hops. For all i, we construct an adversary $\mathbb{A}_i$ and we denote S_i the probability that $\mathbb{A}_i$ wins Game i.

Game 1: In this game, the adversary no longer takes part into the tally process at line 24 but instead is given the output of the protocol, computed by a trusted party (*i.e.* the result of each PET, the result of the reencryption mixnet and the result of the decryption).

Since we assumed that the cryptographic primitives of JCJ are perfect (*i.e.* SUC-secure), there exists an adversary $\mathbb{A}_1$ such that $|S_1 - S_0|$ is negligible.

Game 2: In this game, the adversary no longer has access to the output of the PET, the reencryption mixnet and the final decryptions; instead it is given $\boldsymbol{X}, \boldsymbol{I}$, the final result of the tally.

To construct $\mathbb{A}_2$, it is sufficient to show that one can simulate the missing outputs, given $\boldsymbol{X}$ and $\boldsymbol{I}$. First, using $\boldsymbol{I}$, the adversary not only learns the number of duplicates n_d, but also the pairwise results of the PETs for all the ballots of the board. Indeed, if (C_1, C_2, π) and (C'_1, C'_2, π') are two ballots of respective indexes $j > i$ in the board, $\mathsf{PET}(C_2, C'_2) = 1$ if and only if $\mathsf{id}_j = \mathsf{id}_i$, where id_i is the pseudonym given to the ith ballot in $\boldsymbol{I}$. As for the mixnet, its output can easily be simulated by using $|\mathsf{PB}| - n_d$ random encryptions. By the IND-CPA property of the ElGamal encryption under the DDH assumption, this leads to a computationally indistinguishable simulation. Afterwards, to simulate the last phase where PETs are used again, the adversary can return $|\boldsymbol{X}|$ 1s at some random positions, the remaining outputs being 0s.

Game 3: In this game, the honest voters only use their real credential for their last (re)vote. For their previous vote, they use a random (and fake) credential instead. This, however, does not change $\boldsymbol{I}$, which is constructed depending on the identity of the voters and not how they formed their ballots. This is the same transition as Game 4 in Section 8.2.1. Using the same argument, we construct an adversary $\mathbb{A}_3$ such that $|S_3 - S_2|$ is negligible.

Game 4: In this game, the adversary no longer has access to the roster Π^R at line 4.

Game 5: In this game, before computing the tally, we decrypt every valid ballot sent by the adversary at lines 21 and 23. If one of these ballots uses the same credential as a ballot sent by a honest voter (*i.e.* a ballot added to the board at line 22 for some (i, ν_i) with $i \in [1, n_V] \backslash (A \bigcup \{j\})$, we abort the game and output a random bit.

Game 6: In this game, we remove line 21 so that the adversary can no longer insert its own ballots between two honest ballots. In other words, the adversary must send all its ballots at the end, after every honest voter has voted.

Game 7: The final game is the ideal game.

We construct an adversary $\mathbb{A}_7$ which interacts with $\mathbb{A}_6$ by simulating Game 6. For this purpose, $\mathbb{A}_7$ runs the setup and the registration honestly, by generating the secret key and the credentials. For $\tilde{c}$, it uses a uniformly random credential. Then, when given $|B|$ in the ideal game, it forwards it to $\mathbb{A}_6$ which answers with a sequence of calls to $\mathcal{O}_{\mathsf{cast}}$. To deduce the corresponding voting options $(\nu_i)_{i\in A}$ and β, $\mathbb{A}_7$ creates a hashmap with the keys $\{c_i; i \in A\}$ and $\tilde{c}$, and values $(\nu_i)_{i\in A}$ and β which are initially ϕ (for abstention). For each valid ballot cast by $\mathbb{A}_6$, $\mathbb{A}_7$ decrypts the ballot using the secret key and deduces (ν, c). Since the ballot is valid, by the soundness of the ZKP, c consists of λ bits and ν is a valid voting option. If c is a key of the hashmap, it changes the corresponding value to ν. (Otherwise, it ignores the ballot.) It plays the obtained values in Game 7 and receives the result of the tally which it forwards to $\mathbb{A}_6$. Finally, it outputs $\mathbb{A}_6$'s output. Remark that $\mathbb{A}_7$ played a perfect simulation of *Game* 6, so that $S_7 = S_6$.

Conclusion. With all the above transitions, we showed that for all adversary $\mathbb{A}$ for the real game, there exists an adversary $\mathbb{B}$ for the weakened ideal game which wins with the same probability (up to a negligible difference). By definition, this shows that JCJ is weakly coercion-resistant. □

7.5.1 Generalization

Our notion of weak coercion-resistance can be adapted, in principle, to emphasize the various qualities of coercion-resistance provided by JCJ-like schemes.

We start with the scheme presented by Araújo, Foulle and Traoré (AFT) in [AFT08]. Its main feature is that it has a linear time complexity for the cleansing and tallying phases. While they use different cryptographic primitives from JCJ, their scheme has a similar structure: voters are given credentials to vote with, and can provide a fake credential to a coercer. Assuming that the cryptography is perfect, we can analyze their leakage and compare it with that of JCJ.

During the tally, both the number of duplicates and the number of ballots which use a fake credential are revealed, just as in JCJ. However, it is possible to deduce, by observing the board, how many revotes each ballot has. In JCJ, this information is only available during the tally, when it is no longer possible for the adversary to submit a ballot. In the AFT scheme, this information is available on the fly, during the whole voting phase, and the adversary may exploit it to submit ballots in a specific way. Consequently, the AFT scheme provides a coercion-resistance level which is similar to Definition 22, but where $\boldsymbol{I}$ is given to $\mathbb{A}$ at line 16 instead of line 21 in the ideal game. This is slightly (but strictly) weaker.

Another interesting example is Civitas [CCM08], a scheme considered as an implementation of JCJ which has a similar level of security regarding coercion-resistance. Among the few differences that could have an impact, we concentrate on the leakage during the cleansing and tallying phases. Interestingly, Civitas actually leaks more information than JCJ. First, it provides the same leakage as the AFT protocol: the number of revotes for each ballot can be directly deduced from the board. Furthermore, in order to reduce the (quadratic) number of PETs, Civitas proposes to group voters by blocks: each credential is publicly assigned to one block, and the voter indicates their block in clear when casting their ballot. Compared to JCJ, the adversary still learns how many revotes each ballot has and how many invalid ballots there is, but also has access to this information block by block. Modeling the exact security of Civitas would

require to weaken the coercion-resistance definition compared to the one we sketched for AFT. In particular, the definition would have to take the number of blocks as a parameter, so that the ideal game could leak a list of K information sets similar to $\boldsymbol{I}$.

Finally, for voting schemes that are not based on JCJ, the adaptation is less immediate. For instance, in the VoteAgain system [LQT20], the paradigm for coercion-resistance is different, since the voters are assumed to be able to vote after the coercer. The idea of revoting is key to the security and needs to be reflected in the definition of coercion-resistance by preventing the adversary to vote at any time. Even though the situation is too different from what we have presented in our work to be applied directly, the amount of information revealed during the cleansing phase should also be carefully assessed when analyzing its resistance to coercion.

7.6 Discussion

The leakage in practice. The scenario considered in Section 7.2.1 is extreme, but illustrates that the JCJ protocol does not provide coercion-resistance in some cases. In addition, this scenario can occur if A and B are two candidates for similar political parties, but A benefits for way more support than B, so that there is actually no one that votes for B instead of A (however, there can be some other candidates). Nevertheless, A can suffer from a discredit in the press, which can lead some of the voters to change their mind and vote for B instead, so that all the voters that vote for B revote exactly once. Conversely, assuming that revoting is not a well-spread behavior, none of the voters that vote for A would revote.

In general, the distribution of revotes is not independent from the final choices of the voters, so that the coercer learns some information by observing the leakage in JCJ, and hence detect when a voter disobeys with some non-negligible advantage. In Section 7.3, we analyze the impact of the leakage for some realistic scenarios, and reveal that it can be significant in not-so-extreme cases.

One could argue that in Definition 21, the adversary is supposed to know a perfect description of the distribution $\mathscr{B}$, which can be considered too much. This is a conservative assumptions where we consider that any information which is not a secret should be available to the adversary. In addition, to obtain the non-negligible advantages that we exhibit in Section 7.3, the adversary does not need to know the full distribution $\mathscr{B}$, but only the expected distribution of votes and revotes, as well as their dependencies. This can be deduced by exit poll or the analysis of the social media, where some voters indicate whether they revoted or not and what was the final choice.

More noise is needed. A known issue of JCJ is that an unpredictable number of fake, *dummy* ballots (*i.e.* some ballots that use an invalid credential) should be added, in order to hide to a coercer that their ballot has been removed. Indeed, if it is usual that absolutely no ballot with a fake credential is removed during the cleansing phase, then a coercer, who observes that exactly one ballot is removed, would suspect that the coerced voter has provided a fake credential.

In JCJ, this "noise" comes from honest voters sending dummy ballots, but this source alone may not be sufficient, and it is unnatural to expect that a *honest* voter would send a *fake* ballot. A natural approach is to have the authorities add a random number of dummies. For instance, [SKHS11] uses this to mitigate a leakage during the tally. This noise made of fake ballots should however be calibrated carefully since the computation overhead is important. In a context where revoting is a well spread behavior, it could be judicious to rely on revoting, at least partially, as an additional source of noise. This is not possible in JCJ where a dummy can be distinguished from a revote, but becomes a possibility if our solution from Section 8.1 is used.

Considering other evasion strategies. One possibility to correct JCJ's flaw would be to define other evasion strategies in case of revoting. Indeed, if Alice wants to vote, JCJ's evasion strategy instructs her to do so exactly once. Consequently, if it is usual for everyone to revote several times, the leakage in JCJ allows the coercer to detect that a single person voted once without revoting, and thus that Alice disobeyed. However, it seems very hard to instruct voters to use revoting, according to a certain distribution, when they are under coercion. As seen above, the natural way to proceed does not work and the task is made even harder by the fact that the strategy may evolve depending on new events that could change the revoting distribution for the honest voters.

Hence, we propose another option (see Section 8.1) that consists in reinforcing JCJ in case of revoting, such that there is no leakage besides the total number of ballots and the number of valid ballots. For our proposed protocol, we prove coercion-resistance with the original evasion strategy of JCJ. We acknowledge that the latter is not perfect; in particular, it does not allow a voter under coercion to change their mind and revote. However, modeling a wide variety of behaviors for the coerced voter is too complex for the time being.

Chapter 8

CHide: a cleansing-hiding variant of JCJ

We propose a modification of JCJ that provides full coercion-resistance. During the tally phase, the trustees perform the same tasks of cleansing, mixing and decrypting as in JCJ, but in a hidden way, so that the coercer (or anyone) does not learn how many ballots were deleted because they correspond to revotes or to invalid credentials. For this purpose, we propose a new cleansing algorithm based on the tally-hiding toolbox.

8.1 Description of the protocol

We design CHide to be as close to the JCJ protocol as possible. In particular, we consider that the final result of the tally consists of the list of the voting options chosen by the voters, in some random order, which means that we do not consider Italian attacks. For the sake of the SUC security, we propose to use a UC-secure decryption mixnet instead of the Terelius-Wikström mixnet; see for instance [Wik04]. Alternatively, any other tally protocol can be applied. Also, we use the same trust assumptions as in JCJ.

Setup. The setup is the same as in JCJ.

Registration. Just as for JCJ, the registrars generate n_V random credentials, where n_V is the number of eligible voters. However, the credentials are generated bit by bit; in other words, for all $1 \leq i \leq n_V$, the registrars generates λ random bits $(c_{i,1}, \cdots, c_{i,\lambda})$ that constitute a valid credential c_i. In addition, they generate the corresponding public encryptions $\boldsymbol{R_i} = (R_{i,1}, \cdots, R_{i,\lambda})$, which are added to the board as a part of the roaster $\Pi^R = (\boldsymbol{R_i})_{i=1}^{n_V}$. Apart from that, the registration is the same as in JCJ: we consider a perfect and untappable registration, where the registrars privately send one unique valid credential to each voter.

Voting phase. In order to cast a vote for the option ν (encoded as a group element), a voter computes C_1, an encryption of ν and $\boldsymbol{C_2}$, a bitwise encryption of the credential. The neutral element 1_G (the encoding g^0 of the zero bit) should not represent any voting option as it will be used to encode the invalid voting options. The voter also produces a PoK π_1 that proves the knowledge of ν and c_j for all j. To ensure a strong Fiat-Shamir transformation [BPW12], the computation of the challenge from the commitment of the Σ-protocol must include all public informations in the hash, such as g, pk, C_1 and $\boldsymbol{C_2}$. Finally, a ZKP π_2 that ν is a valid voting option and that $c_1, \ldots, c_\lambda$ are bits must be added to prevent forced-abstention attacks which use write-ins. The ballot $(C_1, \boldsymbol{C_2}, \pi_1, \pi_2) = \mathsf{Vote}_{\mathsf{pk}}(\nu, c)$ is sent to the public board, using an anonymous channel. The voters check that their ballot is present on the board; this defines the

verification step Check. The auditors verify that the ZKP are valid and that there is no other ballot on the board with the same $(C_1, \boldsymbol{C_2})$; this defines the verification Valid.

Cleansing phase. Just as in JCJ, only one ballot is kept per valid credential. However, the ballots that use an invalid or a duplicate credential are not actually *removed*; instead, the talliers use the CSZ protocol to replace the corresponding C_1 with an encryption of 0, which represents an invalid voting option. This way, no one knows whether a specific ballot is removed or not, let alone for which reason (either because the credential was invalid or used multiple times). For this purpose, the talliers need to first compute an encrypted validity boolean for each ballot. Using the tally-hiding toolbox, it is possible to directly adapt the JCJ cleansing phase in MPC, using Eq instead of PET. However, the quadratic cost of JCJ would result to a large number of CSZ protocol, which would make the MPC protocol impractical. Instead, we propose a quasi linear approach, which relies on sorting. This means that CHide is more scalable than JCJ; however, for an election of a typical size (*i.e.* about 1000 to 10000 voters), it is less efficient.

First, the talliers create a list of pairs of encrypted data $(V_i, \boldsymbol{K_i})_{i=1}^{n_B+n_V}$, where n_B is the number of valid ballots on the board and n_V the number of voters.

The first pairs come from the public board: for all $1 \leq^q i \leq n_B$, if $(C_1^i, \boldsymbol{C_2^i}, \pi_1^i, \pi_2^i)$ is the ith valid ballot on the board, then $V_i = C_1$ and $\boldsymbol{K_i} = (\boldsymbol{K_i^\perp}, \boldsymbol{K_i^\top})$, where $\boldsymbol{K_i^\perp}$ is a bitwise encryption of the order of appearance on the public board and $\boldsymbol{K_i^\top} = \boldsymbol{C_2}$. The order of appearance is numbered between 0 and $n_B - 1$; however, since we also want to encrypt the upper bound n_B, $\boldsymbol{K_i^\perp}$ is encrypted using $\lceil \log(n_B + 1) \rceil$ encrypted bits. For the sake of verifiability, those encryptions use the randomness 0.

Afterwards, the remaining pairs come from the public encryptions of the valid credential, *i.e.* the public roster: for all $n_B + 1 \leq i \leq n_B + n_V$, if $\boldsymbol{R_i}$ is the $(i - n_B)$th entry of the roster, $V_i = E_0$, a trivial encryption of 0, and $\boldsymbol{K_i} = (\boldsymbol{K_i^\perp}, \boldsymbol{K_i^\top})$ with $\boldsymbol{K_i^\perp} = {n_B}^{\text{bits}}$ and $\boldsymbol{K_i^\top} = \boldsymbol{R_i}$.

The talliers then use the OddEvenMergeSort protocol to sort the list of the $(V_i, \boldsymbol{K_i})$, in increasing order. This has the following effect:

- First, the elements of the list are sorted according to $\boldsymbol{K_i^\top}$, in increasing order. Hence the ballots that use the same credential are grouped together;
- Second, the elements that have the same $\boldsymbol{K_i^\top}$ (*i.e.* the ballots that use the same credential) are sort in increasing order of $\boldsymbol{K_i^\perp}$. Hence, the entry coming from the roster (if any) appears at the end of the group and the last valid ballot cast (if any) is moved just behind.

After this step, an entry comes from a ballot with a valid credential if and only if the two following conditions are met: 1) its $\boldsymbol{K_i^\top}$ part is the same as the one from its successor in the sorted list; 2) the $\boldsymbol{K_i^\perp}$ part of its successor encodes n_B. These tests can be efficiently implemented with the MPC toolbox and we need only a linear number of them.

The P_{tally} protocol is more precisely presented as follows.

1. Discard all the ballots marked as invalid by the Valid procedure. Let $(C_1^i, \boldsymbol{C_2^i})_{i=1}^{n_B}$ be the remaining ballots, without the ZKPs. We denote $\ell = \lceil \log(n_B + 1) \rceil$.
2. For all $1 \leq i \leq n_B$, set $V_i = C_1^i$ and $\boldsymbol{K_i} = {(i-1)}^{\text{bits}} || \boldsymbol{C_2^i}$, where ${(i-1)}^{\text{bits}}$ is a trivial bit-wise encryption of $i - 1$ that uses ℓ bits (least significant bit first).
3. For all $n_B + 1 \leq i \leq n_B + n_V$, set $\boldsymbol{K_i} = {n_B}^{\text{bits}} || = \boldsymbol{R_{i-n_B}}$ and $V_i = E_0$.
4. Sort the $(V_i, \boldsymbol{K_i})$ in increasing order, using the keys $\boldsymbol{K_i}$. This produces a result $(V_i', \boldsymbol{K_i'})_{i=1}^{n_B+n_V}$ and a transcript Π^{Sort}.

5. For all $1 \leq i < n_B + n_V$, compute $D_i = \mathrm{Eq}(\boldsymbol{K'}_{\boldsymbol{i}}^{\top}, \boldsymbol{K'}_{\boldsymbol{i+1}}^{\top})$, where $\boldsymbol{K'}_{\boldsymbol{i}}^{\top}$ refers to the λ most significant (encrypted) bits of $\boldsymbol{K'_i}$. This produces the transcript $\Pi_{i,1}^{\mathrm{Eq}}$.

6. For all $1 \leq i < n_B + n_V$, compute $F_i = \mathrm{EqKnown}(\boldsymbol{K'}_{\boldsymbol{i+1}}^{\perp}, {n_B}^{\mathrm{bits}})$, where $\boldsymbol{K'}_{\boldsymbol{i+1}}^{\perp}$ refers to the ℓ least significant bits of $\boldsymbol{K'_{i+1}}$. This produces the transcript $\Pi_{i,2}^{\mathrm{Eq}}$.

7. For all $1 \leq i < n_B + n_V$, replace V'_i by $\mathrm{CSZ}(V'_i, \mathrm{And}(D_i, F_i))$.

8. Apply a decryption mixnet on the $(V'_i)_{i=1}^{n_B+n_V-1}$. This produces the result of the election as well as a verification transcript Π^{Mixnet}.

Each step produces a transcript, published on the board, and verified by the auditors.

Evading coercion. As mentioned previously, we keep the same evasion strategy as in JCJ: to evade coercion, a voter generates a random credential which consists of λ uniformly random bits and gives this to the coercer. Latter, the voter uses the legitimate credential to vote (once) for the desired voting option (or abstain).

8.1.1 Efficiency considerations

In terms of computational and communication costs, CHide is less efficient than JCJ, mainly because the encrypted credentials are now formed by λ ciphertexts instead of a single one.

For the talliers, the cleansing phase is more complex but scales better with the number of submitted ballots. While JCJ is quadratic, we propose a quasi-linear tally protocol based on sorting. Another difference is that, due to the MPC toolbox, the number of communication rounds between them is no longer constant, but slightly depends on the number of ballots and the security parameter. Nevertheless, the task is highly parallelizable and remains affordable for medium-size elections. Also, we mentioned that we use the mixnet of [Wik04] instead of the one from [Wik09, TW10]. This is so that the whole tally protocol remains SUC-secure. The complexity of [Wik04] is different from that of [Wik09, TW10] and can be more expensive; however, it is still linear with respect to the number of ciphertexts to shuffle. Since there are $n_B + n_V - 1$ such ciphertexts to decrypt, the factor λ is no longer present and the mixing phase is going to be way cheaper than the remaining of the cleansing phase. Hence, it is safe to consider that the cost of the decryption mixnet is negligible compared to that of applying the toolbox.

For the voters, the computational load increases but the total cost for realistic parameters is around a thousand exponentiations, which should be a matter of seconds with a standard implementation in JavaScript running within a modern browser.

In Table 21, we give estimates of the number of exponentiations and of the transcript size for both JCJ and CHide. For this purpose, we consider a number of $n_T = 3$ talliers with a threshold $t = 2$, a security parameter of $\lambda = 128$ and a number of $n_C = 2$ voting options. We also give the corresponding running times, based on an estimate of 5000 exponentiations per second on the client side, and 10000 per second on the server side. This reveals that the CHide voting system is still a realistic option for under 10000 submitted ballots, since computing the tally would take about 9 hours if each tallier uses 128 CPU cores.

8.2 Security proofs for CHide

In this section, we prove that CHide is coercion-resistant, ensures privacy and is universally verifiable. For this purpose, we use the same trust assumptions as in JCJ: the registrars are supposed honest, up to a threshold of t talliers can be corrupted and the public board is honest.

Table 21: Number of exponentiations and transcript size in JCJ and CHide, with $\lambda = 128$.

# voters	# exp.		estimated CPU time		transcript	
any	JCJ	CHide	JCJ	CHide	JCJ	CHide
(Vote)	27	1.4k	5.4ms	0.28s	1.1kB	58kB
10	JCJ	CHide	JCJ	CHide	JCJ	CHide
(Tally)	4.26k	4.1M	0.43s	6.8min	170kB	65MB
100	JCJ	CHide	JCJ	CHide	JCJ	CHide
(Tally)	380k	120M	38s	3.3h	16.0MB	1, 9GB
1000	JCJ	CHide	JCJ	CHide	JCJ	CHide
(Tally)	37.5M	2.4G	1.0h	2.8d	1.59GB	39GB
10000	JCJ	CHide	JCJ	CHide	JCJ	CHide
(Tally)	3.75G	41G	4, 3d	48d	158GB	668GB
100000	JCJ	CHide	JCJ	CHide	JCJ	CHide
(Tally)	375G	658G	1.2y	2.1y	15.8TB	10.6TB
1000000	JCJ	CHide	JCJ	CHide	JCJ	CHide
(Tally)	37.5T	9.4T	1.2×10^2y	30y	1.58PB	152TB

8.2.1 Proof of coercion-resistance

The definition of coercion-resistance is given in Definition 21, based on the comparison of two games that we reproduce below. Just as in the JCJ paper, we consider that the registration is perfect, that the voting channel is anonymous, that up to a threshold t of talliers is corrupted, and that the registrars are honest. Note that in Appendix E, we also give a proof of privacy for CHide, which is similar to that of coercion-resistance.

Theorem 10. *Under the DDH assumption and in the ROM, assuming a SUC-secure decryption mixnet, CHide is coercion-resistant.*

Proof. We give a succession of games such that Game 0 is the real game and Game 9 is the ideal game. We consider a PPT $\mathbb{A}_0$ for Game 0. For Game i, we construct a PPT adversary $\mathbb{A}_i$ for this game and we denote S_i the probability that $\mathbb{A}_i$ wins this game. (To ease the notation, we drop the dependency in λ when the context is clear.) For all i, we show that $|S_{i+1} - S_i|$ is negligible, which proves that $|S_0 - S_9|$ is also negligible.

Game 1: In this game, the adversary no longer takes part into the whole tally process at line 24, but only in the decryption mixnet process. Instead, it is given the result of all the conditional gates, computed by a trusted party. With a similar argument as in Theorem 6, we can show that the cleansing phase up to the decryption mixnet is SUC-secure, so that there exists an adversary $\mathbb{A}_1$ such that $|S_1 - S_0|$ is negligible.

Game 2: In this game, the adversary no longer takes part in the decryption mixnet and is instead given the result at line 25, computed by a trusted party. Since the decryption mixnet is supposed SUC-secure, we can similarly construct an adversary $\mathbb{A}_2$ such that $|S_2 - S_1|$ is negligible.

Game 3: In this game, the adversary is no longer given the output of the conditional gates. Just as in the transition to Game 2 in the proof of Theorem 8, under the DDH assumption and in the ROM, there exists $\mathbb{A}_3$ such that $|S_3 - S_2|$ is negligible.

Algorithm 104: $\mathsf{Real}^{\mathsf{CR}}$

Requires: $\mathbb{A}, \lambda, n_T, C_t, n_V, n_A, n_C, \mathcal{B}$

$\mathsf{pk}, (s_i, h_i)_{i=1}^{n_T}, \Pi^S \longleftarrow \mathsf{Setup}(\lambda, n_T, t)$;
$(c_i, \pi_i), \Pi^R \longleftarrow \mathsf{Register}(\mathsf{pk}, n_V)$;
$\mathsf{PB} \longleftarrow \Pi^S || \Pi^R$;
$A \longleftarrow \mathbb{A}(\mathsf{PB}, \{s_i \mid i \in C_t\})$ (* corrupt voters *);
$(j, \alpha) \longleftarrow \mathbb{A}(\{c_i; i \in A\})$;
(* coerces j who has the intention α *)
if $|A| \neq n_A \vee j \notin [1, n_V] \backslash A \vee \alpha \notin [1, n_C] \bigcup \{\phi\}$ **then return** 0 ;
$B \longleftarrow \mathcal{B}([1, n_V] \backslash A, n_C)$;
(* samples a sequence of pairs (i, ν_i) with $i \in ([1, n_V] \backslash A) \bigcup \{n \mid n < 0\}$ *)
for $(i, *) \in B$, $i \notin [1, n_V]$ **do**
  $c_i \longleftarrow$ Fakecred();
  (* this captures dummy ballots *)
$b \xleftarrow{\$} \{0, 1\}$;
$\tilde{c} \longleftarrow c_j$;
if $b = 1$ **then** Remove all $(j, *) \in B$;
else
  Remove all $(j, *) \in B$ but the last, which is replaced by (j, α) if $\alpha \neq \phi$ and removed otherwise;
  $\tilde{c} \longleftarrow \mathrm{Fakecred}(c_j)$;
$\mathbb{A}(\tilde{c})$ (* $\mathbb{A}$ learns $\tilde{c}$ *);
for $(i, \nu_i) \in B$ *(in this order)* **do**
  $\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(\mathsf{PB})$ (* casts valid ballots *);
  $\mathsf{PB} \leftarrow \mathsf{PB} \bigcup \{\mathsf{Vote}_{\mathsf{pk}}(c_i, \nu_i)\}$;
$\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(\mathsf{PB}, \texttt{"end for"})$;
$X, \Pi \leftarrow \mathsf{Tally}^{\mathbb{A}}(\mathsf{PB}, \mathsf{pk}, \{s_i\})$;
$b' \longleftarrow \mathbb{A}()$;
if $b' = b$ **then return** 1 **else return** 0;

Algorithm 105: $\mathsf{Ideal}^{\mathsf{CR}}$

Requires: $\mathbb{A}, \lambda, n_V, n_A, n_C, \mathcal{B}$

;
;
;
$A \longleftarrow \mathbb{A}(\lambda)$ (* corrupt voters *);
$(j, \alpha) \longleftarrow \mathbb{A}()$;
(* coerces j who has the intention α *);
if $|A| \neq n_A \vee j \notin [1, n_V] \backslash A \vee \alpha \notin [1, n_C] \bigcup \{\phi\}$ **then return** 0 ;
$B \longleftarrow \mathcal{B}([1, n_V] \backslash A, n_C)$;
(* samples a sequence of pairs (i, ν_i) with $i \in ([1, n_V] \backslash A) \bigcup \{n \mid n < 0\}$ *);
;
;
;
$b \xleftarrow{\$} \{0, 1\}$;
;
if $b = 1$ **then** Remove all $(j, *) \in B$;
else
  Remove all $(j, *) \in B$ but the last, which is replaced by (j, α) if $\alpha \neq \phi$ and removed otherwise;
;
;
$(\nu_i)_{i \in A}, \beta \longleftarrow \mathbb{A}(|B|)$;
if $(b = 1) \wedge (\beta \in [1, n_C])$ **then**
  $B \longleftarrow B \bigcup \{(j, \beta)\}$;
$B \longleftarrow B \bigcup \{(i, \nu_i) \mid i \in A, \nu_i \in [1, n_C]\}$;
$X \longleftarrow \mathsf{count}(\mathsf{cleanse}(B))$;
$b' \longleftarrow \mathbb{A}(X)$;
if $b' = b$ **then return** 1 **else return** 0;

Figure 27: Definition of coercion-resistance. λ is the security parameter, n_T the number of talliers, t the threshold, C_t the set of the corrupted talliers, n_V the number of voters, n_A the number of corrupted voters, n_C the number of voting options and $\mathcal{B}$ the distribution.

Game 4: In this game, we modify the sequence B so that the honest voters no longer revote. Instead, for all honest voter x, we replace all but the last occurrence of the form (x, ν) in B by an occurrence of the form $(\tilde{x}, \nu)$ which uses a fresh and unique $\tilde{x} < 0$. This way, the last vote remains the same but the previous votes are replaced by a vote with a fresh, random (and fake) credential. Note that this modification happens on the sequence B, before the voters actually cast their votes according to this sequence.

Let n_R be the number of revotes. We set $\mathbb{A}_4 = \mathbb{A}_3$ and argue that $|S_4 - S_3|$ is negligible. For this purpose, we give a succession of hops $H_0, \cdots, H_{n_R}$ such that in game H_i, we replace the last i revotes as described above. This way, H_0 is Game 3 and H_{n_r} is Game 4. For each of these games, we denote W_i the probability that $\mathbb{A}_4$ wins this game.

Now, let i be some index. We construct an adversary $\mathbb{B}$ for IND-PA0 as follows. First, $\mathbb{B}$ receives pk from the IND-PA0 game. It simulates game H_i by giving this pk to $\mathbb{A}_4$ and generating the credentials at random. It then gets (j, α) from H_i and chooses b at random. Afterwards, $\mathbb{B}$ generates B from the distribution $\mathcal{B}$, (j, α) and b. It then continues the simulation of H_i by replacing the i last revotes as necessary. However, for the next remaining revote, it looks up for the previous vote (x, ν) with the same x and generates a random, fresh credential c. It plays $(\nu, c_x), (\nu, c)$ in the IND-PA0 game and gets an encrypted ballot C which it plays in the simulation of H_i instead of a honestly generated ballot for (ν, c_x) (therefore, C is added on the board). Finally, to compute the tally, $\mathbb{B}$ plays the concatenation of all the valid ballots sent by $\mathbb{A}_4$ in the IND-PA0 game, which returns a decryption of these ballots. Remark that due to the nature of Valid which uses a form of weeding, a valid ballot is not already on the board. Therefore, none of the valid ballots sent by $\mathbb{A}_4$ can be equal to C, so that the IND-PA0 will indeed accept to decrypt them. $\mathbb{B}$ computes the result of the tally from the plaintexts and gives it to $\mathbb{A}_4$ which answers with some bit b'. This is possible because every valid ballot of the board has a valid ZKP π_2, whose soundness guarantees that the corresponding plaintext (ν, c) is such that c is a λ-bit credential and ν a valid voting option. If $b = b'$, $\mathbb{B}$ states that the IND-PA0 game encrypted (c_x, ν), and (c, ν) otherwise.

Clearly, when the IND-PA0 game encrypts (ν, c_x) (resp. (ν, c)), $\mathbb{B}$ plays a perfect simulation of game H_i (resp. H_{i+1}) so that $b = b'$ with probability W_i (resp. W_{i+1}). Hence, $\mathbb{B}$'s advantage in the IND-PA0 game is $|\frac{1}{2}(W_i + 1 - W_{i+1}) - \frac{1}{2}| \leq \varepsilon_{\mathsf{PA0}}$. By the triangular inequality,

$$|S_4 - S_3| \leq 2n_R\varepsilon_{\mathsf{PA0}}.$$

Game 5: In this game, the adversary no longer has access to the roster Π^R, which contains the encryptions of the valid credentials. We construct $\mathbb{A}_5$ which interacts with $\mathbb{A}_4$ by simulating Game 4. For this purpose, it generates $n_V\lambda$ ElGamal encryptions of random bits and uses it to simulate the roster. For the remaining of the game, it can play a perfect simulation since both games are identical. To argue that $|S_5 - S_4|$ is negligible, we construct a succession of hop $H_0, \cdots, H_{n_V}$ such that in game H_i, the last i elements of Π^R are replaced by a random encryption. This way, H_0 is Game 4 and H_{n_V} is Game 5. For each of these games, we denote W_i the probability that $\mathbb{A}_4$ wins this game.

Now, let i be some index. We construct an adversary $\mathbb{B}$ for IND-PA0 as follows. $\mathbb{B}$ gets pk from the IND-PA0 game and plays this pk to $\mathbb{A}_4$ to simulate H_i. It generates the credentials and replaces the last i elements of the roster by encryptions of random bits just as in H_i. However, for the $(n_V - i)$th element of Π^R, it generates a second random credential c and plays the pair $(\nu, c_{n_V - i}), (\nu, c)$ in the IND-PA0 game, where ν is some valid voting option. The IND-PA0 game answers with some encrypted ballot of the form $(C_1, \boldsymbol{C_2}, \pi)$. $\mathbb{B}$ retrieves the ElGamal bitwise

encryption $\boldsymbol{C_2}$ of the credential and uses it as the $(n_V - i)$th element of Π^R instead of an honest bitwise encryption of c_{n_V-i}. Afterwards, $\mathbb{B}$ continues the simulation of H_i (see the transition to Game 4 for more details) and outputs 1 if and only if $\mathbb{A}_4$ wins the game.

Clearly, when the IND-PA0 encrypts (ν, c_{n_V-i}) (resp. (ν, c)), $\mathbb{B}$ plays a perfect simulation of game H_i (resp. H_{i+1}) to $\mathbb{A}_4$, so that the advantage of $\mathbb{B}$ in the IND-PA0 game is $|\frac{1}{2}(W_i + 1 - W_{i+1}) - \frac{1}{2}| \leq \varepsilon_{\mathsf{PA0}}$. By the triangular inequality,

$$|S_5 - S_4| \leq 2n_V \varepsilon_{\mathsf{PA0}}.$$

Game 6: In this game, before computing the tally, we decrypt every valid ballot sent by the adversary at lines 21 and 23. If one of these ballots uses the same credential as a ballot sent by a honest voter (*i.e.* a ballot added to the board at line 22 for some (i, ν_i) with $i \in [1, n_V] \backslash (A \bigcup \{j\})$, we abort the game and output a random bit.

Now, we set $\mathbb{A}_6 = \mathbb{A}_5$ and, to argue that $|S_6 - S_5|$ is negligible, we remark that $|S_6 - S_5| = \varepsilon/2$, where ε is the probability that we abort in Game 6. Let E be the event of an abortion. We construct an adversary $\mathbb{B}$ for IND-PA0 which wins with a non-negligible advantage whenever E occurs and wins with probability 1/2 otherwise, which shows that ε is negligible.

First, $\mathbb{B}$ gets pk from the IND-PA0 game and forwards it to $\mathbb{A}_5$. It simulate Game 5 as in the transition to Game 4. However, it chooses a random honest voter x that would send a ballot (if no honest voter votes, E cannot happen) and, for this voter, generates a fresh, second random credential c. When this voter votes (which happens at most once due to the transition to Game 4), $\mathbb{B}$ plays the pair $(\nu, c_x), (\nu, c)$ in the IND-PA0 game, where ν is the voting option chosen by x. It gets back an encrypted ballot C, which it uses in the simulation instead of the ballot from x. Afterwards, $\mathbb{B}$ gets the list of all valid ballots sent by $\mathbb{A}_5$ in the simulation and plays them in the IND-PA0 game to get their decryption. If there is a ballot which uses the credential c_x (resp. c), $\mathbb{B}$ states that the IND-PA0 encrypted (ν, c_x) (resp. (ν, c)). If there is no such ballot, a ballot which uses c and a ballot which uses c_x or if no honest voter votes, $\mathbb{B}$ guesses at random.

Now, suppose that $b = 0$ (resp. 1) in the IND-PA0 game; in other words, that C is an encryption of (ν, c_x) (resp. $(\nu, c,)$). Let q be the number of valid ballots sent by $\mathbb{A}_5$. With probability ε, $\mathbb{A}_5$ managed to produce a ballot which uses the same credential as a ballot sent by some honest voter. In this case, with probability at least $1/n_H$, one of the concerned honest voter is x. Then, when $\mathbb{B}$ gets the decryption from the IND-PA0 game, there is a ballot of the form (γ, c_x) (resp. (γ, c)). In addition, $\mathbb{A}_5$ has no information about c (resp. c_x) so that with probability at least $1 - q/2^\lambda$, there is no ballot of the form (γ, c) (resp. (γ, c_x)). Hence $\mathbb{B}$ wins with probability at least $1 - q/2^{\lambda+1}$. Otherwise, no ballot uses the credential c_x (resp. c) and since the adversary has no information about c (resp. c_x), the probability that a ballot uses the credential c (resp. c_x) is at most $q/2^\lambda$. Therefore, the probability that $\mathbb{B}$ wins the IND-PA0 game is at least $(1 - q/2^\lambda)/2$. Overall, $\mathbb{B}$'s probability to win is at least

$$\frac{\varepsilon}{n_H}(1 - q/2^{\lambda+1}) + (1 - \frac{\varepsilon}{n_H})(1 - q/2^\lambda)/2 = \frac{1}{2} + \frac{\varepsilon}{2n_H} - \frac{q}{2^{\lambda+1}}.$$

Therefore, we have

$$\frac{\varepsilon}{2n_H} - \frac{q}{2^{\lambda+1}} \leq \mathsf{Adv}^{\mathsf{IND-PA0}}_{\mathbb{B}} \leq \varepsilon_{\mathsf{PA0}},$$

hence $|S_6 - S_5| = \varepsilon/2 \leq n_H \varepsilon_{\mathsf{PA0}} + \frac{qn_H}{2^{\lambda+1}}$, where q is the number of valid ballots sent by the adversary.

Game 7: In this game, we remove line 21 so that the adversary can no longer insert its own ballots between two honest ballots. In other words, the adversary must send all its ballots at the end, after every honest voter has voted. We construct $\mathbb{A}_7$ which interacts with $\mathbb{A}_6$ by simulating Game 6. For this purpose, $\mathbb{A}_7$ gets $\mathsf{PB} = (B_1, \cdots, B_n)$ at line 23 and creates a fake empty ballot box PB'. Then, in the kth iteration of the for loop, it appends to PB' the valid ballots output by $\mathbb{A}_6$ and then B_k. The remaining of the simulation is similar to that of the transition to Game 4.

Clearly, $\mathbb{A}_7$ plays a perfect simulation of Game 6 if the result of the tally is the same. Besides, the latter can only differ if the credential of a ballot sent by $\mathbb{A}_6$ is the same as the credential of a ballot sent by some honest voter. In this case, both games abort with a random output and $\mathbb{A}_7$'s probability to win is the same as $\mathbb{A}_6$'s in Game 6. Consequently, $S_7 = S_6$.

Game 8: In this game, the adversary has no longer access to the ballot box PB at line 23 but instead has $|B|$ (which is equal to $|\mathsf{PB}|$). With a similar argument as in Game 5, we construct $\mathbb{A}_8$ and we have

$$|S_8 - S_7| \leq |\mathsf{PB}|\varepsilon_{\mathsf{PA0}}.$$

Game 9: The final game is the ideal game.

We construct an adversary $\mathbb{A}_9$ which interacts with $\mathbb{A}_8$ by simulating Game 8. For this purpose, $\mathbb{A}_9$ runs the setup and the registration honestly, by generating the secret key and the credentials. For $\tilde{c}$, it uses a uniformly random credential. Then, when given $|B|$ in the ideal game, it forwards it to $\mathbb{A}_8$ which answers with a sequence of calls to $\mathcal{O}_{\mathsf{cast}}$. To deduce the corresponding voting options $(\nu_i)_{i\in A}$ and β, $\mathbb{A}_9$ creates a hashmap with the keys $\{c_i; i \in A\}$ and $\tilde{c}$, and values $(\nu_i)_{i\in A}$ and β which are initially ϕ (for abstention). For each valid ballot cast by $\mathbb{A}_8$, $\mathbb{A}_9$ decrypts the ballot using the secret key and deduces (ν, c). Since the ballot is valid, by the soundness of the ZKP, c consists of λ bits and ν is a valid voting option. If c is a key of the hashmap, it changes the corresponding value to ν. (Otherwise, it ignores the ballot.) It plays the obtained values in Game 9 and receives the result of the tally which it forwards to $\mathbb{A}_8$. Finally, it outputs $\mathbb{A}_8$'s output. Remark that $\mathbb{A}_9$ played a perfect simulation of *Game* 8, so that $S_9 = S_8$.

Conclusion. With all the above transitions, we showed that for all adversary $\mathbb{A}$ for the real game, there exists an adversary $\mathbb{B}$ for the ideal game which wins with the same probability (up to a negligible difference. By definition, this shows that CHide is coercion-resistant. □

8.2.2 Proof of verifiability

To formalize the notion of universal privacy, we use the approach of [CGGI14] which is presented in Section 1.2.2. This, however, comes with a difficulty when revoting is allowed. Indeed, if a voter's votes are, in order, (ν_1, ν_2, ν_3) and if the voter does not verify that each of the corresponding ballots appear on the board when casting them, then the adversary might drop any of those ballots and have the voter abstain, or have their vote counted as ν_1, ν_2 or ν_3, which technically breaks the definition since the adversary should only be able to have the vote be counted as ν_3 or nothing. Most of the existing electronic voting schemes are actually vulnerable to this when revoting is allowed, and specific measures to enforce that the definition of [CGGI14] is verified are yet to appear in the literature. This problem, which concerns individual verifiability, is independent from the tally phase and whether JCJ of CHide is used. Therefore, we consider a weaker definition where every voter systematically verifies that their ballot appears on the board.

This gives Definition 23, based on the verifiability game given in Fig. 28. In this game, the setup and the registration are run honestly and the adversary is given the public transcripts

Π^S, Π^R, which contains the public key pk, the public commitments $(h_i)_{i=1}^{n_T}$ on the secret shares of the talliers and the public roster Π^R. Also, the adversary is given the secret shares of the corrupted talliers. Afterwards, it can corrupt a subset A of the voters. Then, given the roster and the credentials of the corrupted voters, the adversary must generate a valid bulletin board. For this purpose, it has access to the oracle $\mathcal{O}_{\mathsf{cast}}$ which takes as input a ballot B and adds it to the public board if the ballot is valid. The adversary also has access to $\mathcal{O}_{\mathsf{vote}}$, which models the ballots sent by the honest voters. However, $\mathcal{O}_{\mathsf{vote}}$ modifies two inner tables H and HV. The first one model the fact that a honest voter is supposed to check that their ballot appears on the board. If a honest voter vote but does not check or if their check fails, they become unhappy. The second tabular represent the state of a voter, and contains their ballot, their credential and their chosen voting option. The adversary can have a voter i check their vote by calling $\mathcal{O}_{\mathsf{check}}(i)$ which causes the voter to initiate the Check procedure with the current state HV_i of the voter and the current bulletin board. The adversary can call $\mathcal{O}_{\mathsf{cast}}$, $\mathcal{O}_{\mathsf{vote}}$ and $\mathcal{O}_{\mathsf{check}}$ any (polynomial) number of times, in any order. It then takes part into the tally to produce the result $\boldsymbol{X}$ as well as the transcript Π. The goal of the adversary is to produce a valid $\boldsymbol{X}, \Pi$ which is different from any result obtained with the sequence $(i, \mathsf{HV}_i)_i$ concatenated with another sequence $(\mathsf{cor}_i, \alpha_i)_i$, where the cor_i's are corrupted voters. In addition, all honest voter who have voted must be happy.

We acknowledge that, in our definition, we assume that the registrar is honest, and that up to t decryption trustees can be corrupted. Usually, one would want verifiability even if all the talliers and the registrar are corrupted. In [CGGI14], for instance, it is assumed that either the registrar or the bulletin board is honest. However, in JCJ-like voting systems, the registrar knows the credential of the voter and can therefore break eligibility by voting instead of the abstaining voters (ballot stuffing), or even change the choice of any voter by revoting with their credential afterwards. Similarly, since an encryption of the credentials is published in the roster Π^R, if more than a threshold t of talliers is corrupted, they can decrypt the credentials and perform the same attacks. Therefore, it is necessary to assume that the registrar is honest and that up to a threshold of talliers can be corrupted. Those shortcomings are already present in JCJ.

Definition 23 (Verifiability)**.** *We say that the a voting system* $(\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{Check}, \mathrm{Valid}, \mathrm{Fakecred}, \mathsf{Tally}, \mathrm{Verify})$ *is verifiable if, for all adversary* $\mathbb{A}$*, for all parameters* (n_T, t, n_C) *and for all subset* $C_t \subset [1, n_T]$ *of size at most* t*, there exists a negligible function* μ *such that* $\Pr(\mathsf{Ver}(\mathbb{A}, \lambda, n_T, C_t, n_C) = 1) \leq \mu(\lambda)$.

We now prove that CHide has verifiability. Despite the difference between coercion-resistance and verifiability, there are arguments in common in the two proofs and some parts were reproduced *verbatim*.

Theorem 11. *Under the DDH assumption and in the ROM, CHide is verifiable as of Definition 23.*

Proof. We give a succession of games such that Game 0 is the Ver game and Game 6 is a game which outputs 1 with negligible probability. We consider a PPT $\mathbb{A}_0$ for Game 0. For Game i, we construct a PPT adversary $\mathbb{A}_i$ for this game and we denote S_i the probability that $\mathbb{A}_i$ wins this game. (To ease the notations, we drop the dependency in λ when the context is clear.) For all i, we show that $|S_{i+1} - S_i|$ is negligible, which proves that $|S_0 - S_6|$ is also negligible.

Game 1: In this game, the adversary no longer takes part into the whole tally process at line 24, but only in the decryption mixnet process. Instead, it is given the result of all the conditional gates, computed by a trusted party. With a similar argument as in Theorem 6, we

Ver

Inputs: $\mathbb{A}, \lambda, n_T, C_t, n_C$

$\mathsf{pk}, (s_i, h_i)_{i=1}^{n_T}, \Pi^S \longleftarrow \mathsf{Setup}(\lambda, n_T, t)$;
$1^{n_V} \longleftarrow \mathbb{A}(\Pi^S, \{s_i \mid i \in C_t\})$;
$\{c_i; i \in [1, n_V]\}, \Pi^R \leftarrow \mathsf{Register}(\mathsf{pk}, n_V)$;
$\mathsf{PB} \longleftarrow \Pi^S || \Pi^R$;
$A \longleftarrow \mathbb{A}(\mathsf{PB})$;
for $i \in [1, n_V] \backslash A$ **do**
  $\mathsf{HV}_i \longleftarrow \perp; \mathsf{H}_i \longleftarrow 1$;
$\mathbb{A}^{\mathcal{O}_{\mathsf{vote}}, \mathcal{O}_{\mathsf{cast}}, \mathcal{O}_{\mathsf{check}}}(\{c_i; i \in A\})$;
if $\exists i \notin A \mid \mathsf{H}_i < 1$ **then return** 0;
$r, \Pi \longleftarrow P^{\mathbb{A}}_{\mathsf{tally}}(\mathsf{PB}, \{s_i\})$;
if $\mathrm{Verify}(\mathsf{PB}, \Pi, r) = 0$ **then return** 0;
if $\exists (\mathsf{cor}_i, \alpha_i)_{i=1}^{n} \in (A \times [1, n_C]) \mid$
$r = \mathsf{count}(\mathsf{cleanse}(\{\!\{(i, \mathsf{HV}_i[0]) \mid i \in [1, n_V] \backslash A, \mathsf{HV}_i \neq \perp\}\!\} \uplus \{\!\{(\mathsf{cor}_i, \alpha_i) \mid i \in [1, n]\}\!\}))$ **then return** 0 **else return** 1

$\mathcal{O}_{\mathsf{vote}}$ (updates the tabulars HV and H)

Inputs: i, ν

if $i \in [1, n_V] \backslash A \wedge \nu \in [1, n_C]$ **then**
  $\mathsf{H}_i \longleftarrow \mathsf{H}_i - 1$;
  $B \longleftarrow \mathsf{Vote}_{\mathsf{pk}}(\nu, c_i)$;
  $\mathsf{HV}_i \longleftarrow (\nu, c_i, B)$;
  return B;

$\mathcal{O}_{\mathsf{cast}}$ (updates the bulletin board PB)

Inputs: B

if $\mathrm{Valid}(\mathsf{pk}, \mathsf{PB}, B) = 1$ **then**
  $\mathsf{PB} \longleftarrow \mathsf{PB} \bigcup \{B\}$

$\mathcal{O}_{\mathsf{check}}$ (updates HV and H)

Inputs: i

$\nu, c_i, B \longleftarrow \mathsf{HV}_i$;
if $\mathsf{Check}(\nu, c_i, B, \mathsf{PB}) = 1$ **then**
  $\mathsf{H}_i \longleftarrow \mathsf{H}_i + 1$;
  $\mathsf{HV}_i \longleftarrow \nu, c_i, \perp$;

Figure 28: Definition of verifiability, λ is the security parameter, n_T the number of talliers, t the threshold, C_t the set of the corrupted talliers, n_V the number of voters, n_A the number of corrupted voters and n_C the number of voting options (excluding abstention)

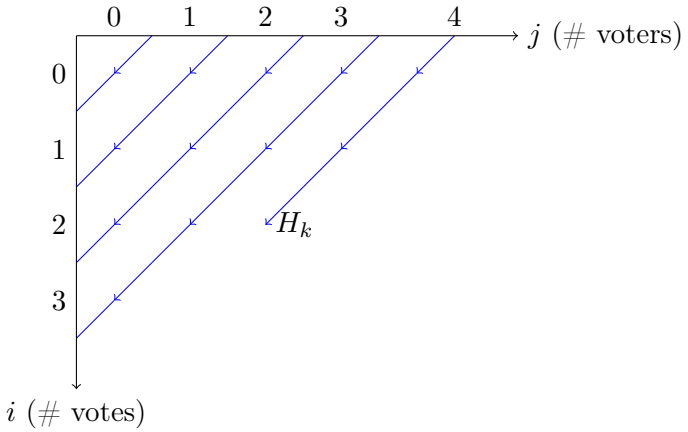


Figure 29: A bijection from $\mathbb{N}$ to $\mathbb{N} \times \mathbb{N}$

can show that the cleansing phase up to the decryption mixnet is SUC-secure, so that there exists an adversary $\mathbb{A}_1$ such that $|S_1 - S_0|$ is negligible.

Game 2: In this game, the adversary no longer takes part in the decryption mixnet and is instead given the result at line 25, computed by a trusted party. Since the decryption mixnet of [Wik04] is proven UC-secure, we can similarly construct an adversary $\mathbb{A}_2$ such that $|S_2 - S_1|$ is negligible.

Game 3: In this game, the adversary is no longer given the output of the conditional gates. Just as in the transition to Game 2 in the proof of Theorem 8, under the DDH assumption and in the ROM, there exists $\mathbb{A}_3$ such that $|S_3 - S_2|$ is negligible.

Game 4: In this game, we create a secret internal state Cred_i for each voter. It contains a sequence of credentials which is initially $[c_i]$, where c_i is i's credential. In addition, we also modify $\mathcal{O}_{\mathsf{vote}}$ so that it uses the last element of Cred_i instead of c_i. It also generates a fresh random credential c which is added at the end of Cred_i, so that the next instance of $\mathcal{O}_{\mathsf{vote}}$ with the same voter will use this new credential instead of the old one. (In case if c is a collision with another voter, we pick another random credential.) Finally, we change the way the result of the tally r is computed. Now, we decrypt each valid ballot of PB into a couple (ν, c). For each such couple, we look for a voter i such that c appears in the sequence Cred_i and we set i's vote as ν (since there is no collision, there may be up to one such voter). Finally, we deduce r by evaluating count on the voters' votes.

To construct $\mathbb{A}_4$ and show that $|S_4 - S_3|$ is negligible, we use a hybrid argument with G_1 as *Game* 3 and G_2 as *Game* 4. This hybrid argument needs to cover two dimensions since the adversary chooses both the number of revotes and the number of voters, which can be at most polynomial. Hence, we use a bijection from $\mathbb{N}$ to $\mathbb{N}^2$ that travels the grid $\mathbb{N} \times \mathbb{N}$ diagonal by diagonal, as illustrated in Fig 29. For $k \in \mathbb{N}$, we denote $i(k), j(k)$ the corresponding elements in $\mathbb{N} \times \mathbb{N}$. Now, we define H_0 as G_2, so that the first condition of the hybrid lemma is verified. Then, for $k \geq 1$, we define H_k as H_{k-1}, except that the $i(k)$th vote of the $j(k)$th voter uses a fresh new credential, as explained above (the subsequent votes use the last generated credential). Remark that, if n_V is the number of voters and n_R the maximum number of times a voter revotes, for $n_{\mathbb{A}} = (n_V + n_R)^2$, we have that $H_{n_{\mathbb{A}}}$ is perfectly indistinguishable from G_1, which means that the second condition of the hybrid lemma is verified.

Now, for all adversary $\mathbb{A}_{k+1}$ for H_{k+1}, we consider the same adversary $\mathbb{A}_k$ for H_k; similarly, for all $\mathbb{A}_k$ for H_k, we consider the adversary $\mathbb{A}_{k+1} = \mathbb{A}_k$ for H_{k+1}, so that the third condition of the lemma is met. For the fourth condition, we will consider the IND-PA0 game.

Finally, we construct the adversary $\mathbb{B}$ for IND-PA0 as follows. Let k be some index and $\mathbb{A}_{k+1}$ be an adversary for game H_{k+1}. We denote $i = i(k)$ and $j = j(k)$. The adversary $\mathbb{B}$ gets pk from the IND-PA0 game and uses this to simulate an instance of H_{k+1} to $\mathbb{A}_{k+1}$. For this purpose, it generates $\{s_i \mid i \in C_t\}$ at random, computes the corresponding h_i and generates $(h_i)_{i \notin C_t}$ by Lagrange interpolation, so that $\mathsf{pk}, (s_i, h_i)_{i=1}^{n_T}$ follows the same distribution as in the verification game. Then, $\mathbb{B}$ runs the registration honestly and creates $\mathsf{PB}, \mathsf{HV}, \mathsf{H}$ and Cred, that it will update according to the answers of $\mathbb{A}_{k+1}$. It will also simulate the oracles $\mathcal{O}_{\mathsf{vote}}, \mathcal{O}_{\mathsf{cast}}, \mathcal{O}_{\mathsf{check}}$.

The ith time $\mathcal{O}_{\mathsf{vote}}$ is called with voter j (let ν be the corresponding voting option), $\mathbb{B}$ retrieves the last credential c used by j from Cred_j and generates a new, fresh credential c' that it adds to Cred_j as in H_{k+1}. However, instead of computing $\mathsf{Vote}_{\mathsf{pk}}(\nu, c')$, $\mathbb{B}$ plays $(\nu, c), (\nu, c')$ in the IND-PA0 game and uses the answer $C^\star$ as the output of $\mathcal{O}_{\mathsf{vote}}$. At some point, $\mathbb{A}_3$ terminates and $\mathbb{B}$ must decide whether the IND-PA0 game encrypted (ν, c) or (ν, c'). For this purpose, $\mathbb{B}$ uses PB, the list of the ballots cast by $\mathbb{A}_{k+1}$. If $C^\star \in \mathsf{PB}$, $\mathbb{B}$ removes it from PB but keeps in memory its index. For the remaining ballots, $\mathbb{B}$ decrypts them using the decryption oracle of the IND-PA0 game. Thanks to the result of the decryptions, $\mathbb{B}$ can decide whether $\mathbb{A}_{k+1}$ won the game or not. In this case, $\mathbb{B}$ states that (ν, c) was encrypted; otherwise, it states that (ν, c') was encrypted.

Clearly, when (ν, c) is encrypted, $\mathbb{B}$ plays a perfect simulation of H_k to $\mathbb{A}_{k+1}$ and, when (ν, c') is encrypted, $\mathbb{B}$ plays a perfect simulation of H_{k+1}. Therefore, by the hybrid argument, we deduce that there exists an adversary $\mathbb{A}_4 = \mathbb{A}_3$ such that $|S_4 - S_3|$ is negligible.

Game 5: In this game, the public roster roster π^R is no longer added to the public board, and therefore no longer given to the adversary at line 5.

To construct $\mathbb{A}_5$ and argue that $|S_5 - S_4|$ is negligible, we use another hybrid argument. This time, the idea is to remove the ciphertexts of the public roster one by one. More precisely, we define G_1 as *Game* 4, G_2 as *Game* 5, and, for $i \in \mathbb{N}$, H_i as G_2, except that the i first entries of the public roster (an entry consists of λ ciphertexts, where λ is the security parameter) are given to the adversary. If $\mathbb{A}_{i+1}$ is an adversary for H_{i+1}, we construct $\mathbb{A}_i$ as an adversary which forwards the i first entries of the public roster that it gets from H_i, and uses λ random encryption of random bits as the $(i+1)$th entry of the roster. This way, $\mathbb{A}_{i+1}$ makes P additional transitions, where P is a polynomial that corresponds to the cost of generating λ ciphertexts.

Now, we construct the adversary $\mathbb{B}$ for IND-PA0 as follows. Given i and $\mathbb{A}_{i+1}$, $\mathbb{B}$ gets the public key pk from the IND-PA0 game and uses this to simulate H_{i+1} to $\mathbb{A}_{i+1}$. However, for the $(i+1)$th entry of the public roster, instead of a honestly generated bitwise encryption of the credential c, $\mathbb{B}$ generates a random voting option ν, a random credential c' and plays the couple $(\nu, c'), (\nu, c)$ in the IND-PA0 game, where c' is c. It gets a challenge ciphertext $C^\star$ from which it extracts the $\boldsymbol{C_2}$ part, which it uses as the $(i+1)$th entry of the public roster. At the end of the simulation, $\mathbb{B}$ uses the decryption oracle of the IND-PA0 game to decrypt the ballots sent by $\mathbb{A}_{i+1}$, and learn whether $\mathbb{A}_{i+1}$ wins the simulation or not. It outputs 1 if and only if $\mathbb{A}_{i+1}$ wins the simulation.

Remark that, when the IND-PA0 game encrypts (ν, c), $\mathbb{B}$ plays a perfect simulation of game H_{i+1} to $\mathbb{A}_{i+1}$. In addition, when the IND-PA0 game encrypts $(\_nu, c')$, $\mathbb{B}$ played $\mathbb{A}_i$'s simulation of H_{i+1} to $\mathbb{A}_{i+1}$. Once again, all the conditions of the hybrid lemma are met.

Game 6: In this game, before computing the result, we decrypt every valid ballot of PB which is not the output of some $\mathcal{O}_{\mathsf{vote}}$. If one of these ballots uses the same credential as a ballot output by $\mathcal{O}_{\mathsf{vote}}$, we abort the game and output a random bit.

Now, we set $\mathbb{A}_6 = \mathbb{A}_5$ and, to argue that $|S_6 - S_5|$ is negligible, we remark that $|S_6 - S_5| = \varepsilon/2$, where ε is the probability that we abort in Game 6. Let E be the event of an abortion. We construct an adversary $\mathbb{B}$ for IND-PA0 which wins with a non-negligible advantage whenever E occurs and wins with probability 1/2 otherwise, which shows that ε is negligible.

First, $\mathbb{B}$ gets pk from the IND-PA0 game and uses it to simulate *Game* 5 to $\mathbb{A}_5$. However, it chooses a random instance of $\mathcal{O}_{\mathsf{vote}}$ (let (x, ν) be its inputs) and, for this instance, generates two random credential $c, \tilde{c}$. $\mathbb{B}$ plays the pair $(\nu, c), (\nu, \tilde{c})$ in the IND-PA0 game, and gets back an encrypted ballot B, which it uses in the simulation as the output of $\mathcal{O}_{\mathsf{vote}}$. Afterwards, $\mathbb{B}$ removes from PB any ballot output by $\mathcal{O}_{\mathsf{vote}}$ (including B) and plays PB in the IND-PA0 game to get the decryption of the remaining ballots. If one uses the credential c (resp. $\tilde{c}$), $\mathbb{B}$ states that the IND-PA0 encrypted (ν, c) (resp. $(\nu, \tilde{c})$). If there is no such ballot or if there is a ballot which uses c and a ballot which uses $\tilde{c}$, $\mathbb{B}$ guesses at random.

Now, suppose that $b = 0$ (resp. 1) in the IND-PA0 game; in other words, that B is an encryption of (ν, c) (resp. $(\nu, \tilde{c})$). Let q_v be the number of (valid) calls to $\mathcal{O}_{\mathsf{vote}}$ and q_c be the number of ballots from PB which are not an output of $\mathcal{O}_{\mathsf{vote}}$. With probability ε, $\mathbb{A}_5$ managed to produce a ballot which uses the same credential as a ballot output by $\mathcal{O}_{\mathsf{vote}}$. In this case, with probability at least $1/q_v$, one of the concerned ballot is B. Then, when $\mathbb{B}$ gets the decryption from the IND-PA0 game, there is a ballot of the form (γ, c) (resp. $(\gamma, \tilde{c})$). In addition, $\mathbb{A}_5$ has no information about $\tilde{c}$ (resp. c) so that with probability at least $1 - q_c/2^\lambda$, there is no ballot of the form $(\gamma, \tilde{c})$ (resp. (γ, c)). Hence $\mathbb{B}$ wins with probability at least $1 - q_c/2^{\lambda+1}$. Otherwise, no ballot uses the credential c (resp. $\tilde{c}$) and since the adversary has no information about $\tilde{c}$ (resp. c), the probability that a ballot uses the credential $\tilde{c}$ (resp. c) is at most $q_c/2^\lambda$. Therefore, the probability that $\mathbb{B}$ wins the IND-PA0 game is at least $(1 - q_c/2^\lambda)/2$. Overall, $\mathbb{B}$'s probability to win is at least

$$\frac{\varepsilon}{q_v}(1 - q_c/2^{\lambda+1}) + (1 - \frac{\varepsilon}{q_v})(1 - q_c/2^\lambda)/2 = \frac{1}{2} + \frac{\varepsilon}{2q_v} - \frac{q_c}{2^{\lambda+1}}.$$

Therefore, we have

$$\frac{\varepsilon}{2q_v} - \frac{q_c}{2^{\lambda+1}} \leq \mathsf{Adv}^{\mathsf{IND-PA0}}_{\mathbb{B}} \leq \varepsilon_{\mathsf{PA0}},$$

hence $|S_6 - S_5| = \varepsilon/2 \leq q_v \varepsilon_{\mathsf{PA0}} + \frac{q_c q_v}{2^{\lambda+1}}$, where q_v is the number of calls to $\mathcal{O}_{\mathsf{vote}}$ and q_c the number of ballots in PB which are not an output of $\mathcal{O}_{\mathsf{vote}}$.

Conclusion. Now, remark that due to the nature of $\mathcal{O}_{\mathsf{vote}}$ which overwrite HV, it is clear that the adversary must call $\mathcal{O}_{\mathsf{check}}(i)$ between each call of $\mathcal{O}_{\mathsf{vote}}$ with the same voter i, otherwise $\mathsf{H}_i < 1$ at line 9. Also, it is also necessary for the adversary to call $\mathcal{O}_{\mathsf{cast}}(B)$ before $\mathcal{O}_{\mathsf{check}}(i)$, where B is the output of $\mathcal{O}_{\mathsf{vote}}(i, \nu)$ (otherwise $\mathcal{O}_{\mathsf{check}}$ does not modify HV. Note that due to the randomness involved in Vote, except with negligible probability μ, the adversary cannot call $\mathcal{O}_{\mathsf{cast}}(B)$ before $\mathcal{O}_{\mathsf{vote}}(i, \nu)$ (at which point it has no information about B). Therefore, the order of the revotes of each voter is enforced in PB and, because of the last transition and the nature of the tally, we readily have that the condition at line 12 always result in a 0 output. Therefore, $|S_6| \leq \mu$, which concludes the proof. □

8.3 Conclusion

We showed that it is possible to correct the flaw of the JCJ protocol and to achieve coercion-resistance as defined in Definition 21. Coercion typically appears in a high stake election, with a politically-binding result that can durably affect the future of a country. For such an election, the priority is the security, and it is not a possibility to tolerate a flaw such as that of JCJ, especially when it can be devastating in some cases. The CHide protocol, although less efficient than the typical electronic voting protocol, allows to address the vulnerability of JCJ. Since coercion is considered as an important threat for many governments, it is not far-fetched that, for a high-stake election, a government may be willing to pay the cost of CHide in order to achieve coercion-resistance. Indeed, CHide's efficiency, although not better than JCJ's, is still practical as it is possible to obtain the result within a day, for less than 3$ per voter. In addition, just as many subsequent schemes focused on improving the scalability of JCJ, it is possible that CHide's efficiency may be improved in future works.

The efficiency considerations, as important as they are, are not the main reason why no coercion-resistant mechanism has been deployed for a politically binding election, other than the revoting paradigm. For the time being, the JCJ protocol, as well as CHide, are still academic proposals that are difficult to apply in practice. For instance, they suppose that the voters are able to correctly handle their credential and use the evasion strategy when under coercion. In addition, the trust assumptions where all the registrars are supposed honest can be considered not acceptable: to extend the verifiability and the privacy in a case where some (but not all) registrars can be corrupted, the usual strategy is that of Civitas [CCM08], which requires the voters to have a well-identified public key and to be able to generate a DVZKP when under coercion. Overall, those "practical" difficulties are far more concerning than the fact that CHide requires a bit more computational power.

Chapter 9

Traceable encryption for verifiable receipt-free electronic voting

In previous chapters, we studied the notion of coercion-resistance, where the adversary can ask the voter to vote in a specific way, using a threat or a reward. A related security notion is that of receipt-freeness, where the voter actively tries to convince a third party (typically the adversary) that they voted in a specific way. In receipt-freeness, it is usual to consider that the adversary might be a vote buyer, *i.e.* that it gives the voter some specific instructions to follow, just as in coercion-resistance. However, the main difference is that we do not consider forced-abstention attacks and that, in receipt-freeness, the adversary cannot ask the voter to give away their credential. Hence, receipt-freeness is often considered as a weaker version of coercion-resistance, but that still addresses the threat of vote buying.

There are various approaches to achieve receipt-freeness. One of the first strategies was, for each voter and each voting option, to prepare an encryption in advance so that the voter cannot use a specific randomness to obtain a receipt [SK95, HS00]. However, this requires a lot of precomputation. In particular, when there are too many voting options, as this is the case with preferential voting, this solution may not be practical. A second approach is based on *deniable revoting*, where the voter can prove that their ballot contains a specific voting option, but is also given the possibility to revote. This way, the vote buyer does not know whether the ballot was canceled by a subsequent ballot or not. In this context, it is important to hide the number of revotes for each voter. Otherwise, the vote buyer can use a strategy known as the "1009 attack", which consists of instructing the voter to vote 1009 times (or any unlikely number, which may be different for each voter). Then, if the voter revotes to cancel the last vote, the attacker may notice that no one actually voted 1009 times. Interestingly, we showed in Chapter 7 that it is also important to conceal the number of revotes per voter, in the context of coercion-resistance. Schemes that prevent the 1009 attack are, for instance, [LHK16] and VoteAgain [LQT20]. Finally, the *rerandomization* paradigm consists of letting the voters cast their ballot as usual. However, the ballot is sent to a *rerandomization server*, which is trusted for the purpose of receipt-freeness. The server rerandomizes the ballot, so that it becomes indistinguishable from a random ballot. This way, even if the ballot was created maliciously, it is no longer possible to prove that the ballot contains a specific voting option. Nevertheless, the voter still has a guarantee that the content of the ballot has not been modified. Some examples of academic proposals based on this strategy are, for instance [Hir10], [BFPV11] and BeleniosRF [CCFG16].

In [DPP22b], Devillez, Pereira and Peters introduce the notion of *traceable encryption*, which augments the notion of encryption scheme with some additional properties related to rerandom-

ization. When those properties are verified, they show that it is possible to achieve receipt-freeness with very few assumptions about the voting protocol. This allows considering registration and eligibility independently from receipt-freeness, which gives more modularity to the protocol design compared to BeleniosRF. In particular, it means that a more generic purpose secret key can be used for the sake of eligibility. For instance, the voters may identify themselves through their health insurance or an electronic identity card, such as the one used in Estonia. In this context, it is plausible that a voter might not be willing to give away their secret credential, as it might be used for other purposes than voting (for instance, to get a loan or a mortgage). By contrast, it is more difficult to argue that the voter might not be willing to give away a short term secret such as the one used in BeleniosRF.

The present chapter is a follow-up of [DPP22b], made in collaboration with Henri Devillez, Olivier Pereira and Thomas Peters. We remark that the construction of [DPP22b], compared to that of [CCFG16], allows vote buying: a vote buyer can give some *instructions* that the voter can follow to produce a convincing receipt. We come to the conclusion that the definition of receipt-freeness used in [DPP22b] has a shortcoming and does not properly model vote buying. Therefore, we propose a new definition of receipt-freeness, that is presented in Section 9.1. Just as the definition of [DPP22b], ours considers that the registration phase and the eligibility mechanism are independent from receipt freeness. To satisfy our definition, we adapt the voting scheme from [DPP22b]; the new voting scheme is presented in Section 9.6. Compared to the solution of [DPP22b], our construction does not allow vote buying, even if the adversary can give an arbitrary instruction to the voter. It also uses a new traceable encryption scheme, which is compatible with 0/1 proofs (as it is often required in electronic voting) and uses a public coin setup protocol, that needs fewer trust assumptions compared to that of [DPP22b]. Compared to the encryption scheme used in BeleniosRF, ours has an encryption and a decryption algorithm which have a linear complexity with respect to the bitlength ℓ of the plaintext (*e.g.*, the number of choices in a multiple choices question), while the decryption algorithm used in BeleniosRF requires an exponential number of group operations (with respect to ℓ). Our traceable encryption scheme is presented in Section 9.4. Finally, we also investigate the possibility to adapt the voting scheme to allow cast-a-intended verification, using the Benaloh challenge (see Section 9.7). In this collaboration, my main contribution was during the design of the new definition of receipt-freeness, the proposed voting protocol and the corresponding security proofs. By contrast, I was not involved in the design of the new traceable encryption scheme, nor in the implementation available at [tre].

Contents

9.1 Our definition of receipt-freeness 205
9.1.1 Existing definitions 205
9.1.2 Modeling vote buying 206
9.2 Introduction to traceable encryptions 209
9.2.1 Definition 209
9.2.2 Security notions for verifiable receipt-free voting 210
9.3 Building blocks 211
9.3.1 Bilinear pairings 211
9.3.2 Linearly Homomorphic Structure-Preserving Signatures 212
9.3.3 The Groth-Sahai proof system 214
9.4 Construction of a traceable encryption scheme 216
9.5 Security proofs for our traceable encryption scheme 218

9.5.1 Verifiability . . . 218
9.5.2 Traceability . . . 221
9.5.3 TCCA security . . . 224
9.6 Application to verifiable receipt-free electronic voting . . . 228
9.6.1 A voting scheme based on a traceable encryption . . . 228
9.6.2 Implementation . . . 230
9.6.3 Receipt-freeness . . . 231
9.7 Adapting the scheme to provide cast-as-intended verification . 233
9.7.1 Adapting our scheme for the Benaloh challenge . . . 234
9.7.2 On the fly cast-as-intended verification . . . 236
9.8 Conclusion . . . 237

9.1 Our definition of receipt-freeness

Our first contribution is to provide a new definition of receipt-freeness. Compared to existing definitions, it is closer to the notion of coercion-resistance and better addresses the threat of vote-buying.

9.1.1 Existing definitions

There are numerous definitions of receipt-freeness in the literature. For instance, [KZZ15] gives a game-based definition to capture both privacy and receipt-freeness. However, this definition considers that the voter uses the expected algorithm to cast a ballot, so that the Helios voting protocol can be considered receipt-free as of their definition. By contrast, if the voter uses a malicious algorithm to cast a ballot in Helios, they can keep in memory the randomness used to encrypt the vote, which can be used as a receipt to prove to a third party that their ballot contains a specific voting option.

In BeleniosRF [CCFG16], which proposes to adapt the BPRIV definition from [BCG+15b] to account for receipt-freeness, no assumption is made about how the voter may send their ballot. On the contrary, it considers that the ballot is provided by the adversary, so that the latter may know the randomness used to encrypt the ballot. However, compared to the definition of Kiayias *et al.*, the definition of Chaidos *et al.* does not account for the possibility that the voter may be given a receipt during the voting phase (for instance through the individual verifiability mechanism). In addition, the definition supposes that the adversary can only provide a ciphertext to the voter, and not just any instruction. This may be restrictive, as it rules out schemes such as [SK95, HS00], where the voters do not create the ciphertexts themselves (rather, the ciphertext is provided by the voting authorities).

In [DPP22b], a variant of the definition of BeleniosRF is used, which makes fewer assumptions on the registration phase. However, it makes more assumptions on the instructions provided by the adversary, which is questionable. In particular, even if their definition is verified, it is possible for the adversary to provide a specific ballot to the voter (that contains the desired voting option), so that the adversary knows exactly if the voter casts this ballot or another ballot. Intuitively, the definition of [DPP22b] captures a less generic version of receipt-freeness, which considers a purely passive adversary, that does not give any instruction to the voter.

Finally, we mention that there exists formal definitions (*e.g.*, [DKR06]), or simulation-based definitions (*e.g.*, [MN06]). However, we prefer to focus on game-based definitions.

- In receipt-freeness, the adversary does not monitor or interact with the voter;
- In particular, coercion-resistance considers forced-abstention attacks and usually requires an anonymous channel, which is not the case for receipt-freeness;
- In receipt-freeness, we do not consider that the adversary threatens the voter: a voter that fails to convince the adversary will not be punished;
- In particular, the voter may be willing to fool the adversary in order to gain money, as they have nothing to lose;
- In receipt-freeness, we do not consider that the adversary may ask the voter to provide their voting credential; instead, it may give some explicit instructions to follow.

Figure 30: The main differences between coercion-resistance and receipt-freeness.

9.1.2 Modeling vote buying

To model vote buying, we consider a situation where an adversary, the vote buyer, runs the website voteselling.onion, where a voter can download (or read) some instructions, say instructions.exe (or instructions.txt). The instructions may be different for each voter, and generated after the voter registers in the website by providing a username and a password. In addition, since the adversary can wait until all the public information are available before releasing the website, the instructions can depend, for instance, on the public encryption key. To simplify the study, we consider that the instructions can be modeled as a deterministic *Turing machine* I. In addition, we also consider that I is compatible with the voting protocol, *i.e.* that the voter can use I to successfully cast a ballot. In other words, the voter can use $I()$ to get the first message to send during the Vote protocol; then, given the answer a of the server, the voter can use $I(a)$ to get the second message to send and so on. We assume that, this way, all the messages sent by the voter are considered valid by the server. When the voter follows the instructions, they get a receipt s, which is an arbitrary string of polynomial size (for instance, the randomness used to encrypt the ballot). Typically, s can be seen as the final output of I, and can depend on any feedback that the voter gets from the voting protocol.

Once the voter gets s, they can upload it on voteselling.onion. Then, given s and the access to the public ballot box, the vote buyer decides whether the voter actually followed the instructions or not. If so, they reward them (how exactly is not relevant).

Our scenario models the threat of vote buying and is close to that of coercion that we studied in the previous chapters. However, there are some differences that we highlight in Fig. 30.

We now give our definition of receipt-freeness, which is designed to capture the above scenario. This definition is adapted from the BPRIV definition [BCG+15b], which means that it requires the voting system to have strong-correctness and strong consistency (see Section 1.3.2). In particular, there exists an efficient algorithm Extract such that, for all honestly generated ballot B of the form $\mathsf{Vote}_{\mathsf{pk}}(\nu, \mathsf{id})$ (where ν is a voting option and id is the identity of the voter), we have $\mathrm{Extract}_{\mathsf{sk}}(B) = (\mathsf{id}, \nu)$. Note that to match the notations of Bernhard *et al.*, we consider that the voting process takes as input the voting option and the identity rather than the voting option and the credential. This means that we consider that receipt-freeness must be achieved independently of the eligibility mechanism, and explains why the definition does not feature a

registration protocol.

To capture receipt-free voting systems that are based on the rerandomization paradigm, we consider that the voting protocol is an interactive protocol between the voter, the server (denoted RS) and the public board. If V is the process of the voter id, we use the notation $s, B \longleftarrow \mathsf{Vote}(\mathsf{id}, V, \mathsf{RS}, \mathsf{PB})$ to express that the voting protocol resulted in the production of the ballot B and the receipt s for the voter (s can be the empty string, a string maliciously obtained by the voter, or any evidence that the voting protocol ended successfully). Note that B might have been rerandomized by the server. If ν is a valid voting option, we use the notation $s, B \longleftarrow \mathsf{Vote}(\mathsf{id}, \nu, \mathsf{RS}, \mathsf{PB})$ to express that the voter honestly followed the specifications of the voting protocol to vote for ν.

Finally, as explained above, we consider that the adversary can give some instructions to the voter, that we model as a Turing machine I. However, I must be *compatible* with the voting protocol, which means that if the voter runs I instead of V, the messages sent by the voter during the voting protocol are valid. We say that the adversary $\mathbb{A}$ is *non-restrictive* if they only give compatible instructions.

Definition 24 (Receipt-freeness)**.** *A voting system* $(\mathsf{Setup}, \mathsf{Vote}, \mathsf{Check}, \mathrm{Valid}, \mathsf{Tally}, \mathrm{Verify})$ *that has strong-correctness and strong-consistency has* receipt-freeness *if there exists two algorithms* SimSetup, SimProof*, a PPT* $\mathcal{D}$ *and a negligible function* μ *such that:*

- *for parameters* n_T, t *and all non-restrictive PPT* $\mathbb{A}$*, we have*

$$\left|\Pr\left(\mathrm{Exp}^{\mathsf{rf\text{-}0}}(\lambda, \mathbb{A}) = 1\right) - \Pr\left(\mathrm{Exp}^{\mathsf{rf\text{-}1}}(\lambda, \mathbb{A}) = 1\right)\right| \leq \mu(\lambda),$$

  where $\mathrm{Exp}^{\mathsf{rf\text{-}b}}$ *is defined in Figure 31;*

- *for all voter* id*, for all compatible instruction* I *(i.e. s.t. if* $s, B \longleftarrow \mathsf{Vote}(\mathsf{id}, \mathsf{I}, \mathsf{RS}, \mathsf{PB})$*, then* $\mathrm{Valid}(B, \mathsf{PB}) = 1$*) and all voting option* ν*, if* $(s, B) \longleftarrow \mathsf{Vote}(\mathcal{D}^{\mathsf{I}}(\nu), \mathsf{RS}, \mathsf{PB})$*, then* $\mathrm{Valid}(B, \mathsf{PB}) = 1$ *and* $\mathrm{Extract}_{\mathsf{sk}}(B) = (\mathsf{id}, \nu)$*.*

Just as in [CCFG16], we define receipt-freeness using an experiment $\mathrm{Exp}^{\mathsf{rf\text{-}b}}$, where the bit b must be guessed by the adversary. The intuition is that when $b = 0$, the protocol is ran honestly and the voter follows the instructions of the vote buyer. When $b = 1$, however, the voter votes according to their own preference but still gives the adversary a receipt. This is possible thanks to the deceiving algorithm $\mathcal{D}$, which allows the voter to not only produce a ballot that looks like the one cast when following the instructions, but also to generate a deceiving receipt. Hence, if the adversary is unable to guess b with a non-negligible advantage, it means that the voter cannot convince the adversary that they followed the instruction instead of applying the deceiving strategy. Alternatively, it also means that the voter might as well vote with the desired voting option and still try to convince the vote buyer in order to gain the reward.

A key element in this definition is that, when $b = 1$, the tally is simulated so that the result of the election is the same whatever the value of b, just as in the BPRIV definition. Consequently, during the experiment, the adversary cannot use any information from the tally to infer whether the voter obeyed or not. To simulate the tally, we stick to the setting of [BCG+15b] and we require that there exists a couple of PPT $(\mathrm{SimProof}, \mathrm{SimSetup})$. In this thesis, we mostly rely on the ROM, so that SimSetup is not required to simulate the proofs. In the standard model, however, it is usual that the ZKP can only be simulated using a trapdoor τ generated during the setup.

During the experiment, the adversary has access to several oracles. $\mathcal{O}_{\mathsf{cast}}$ allows the adversary to cast a valid ballot B in both boards. In the BPRIV definition, the validity is only verified

$\mathrm{Exp}^{\mathsf{rf\text{-}b}}(\lambda, \mathbb{A})$

$\mathsf{PB}_1 \longleftarrow \varnothing$; $\mathsf{PB}_2 \longleftarrow \varnothing$;
$\mathsf{pk}, \mathsf{sk}, \tau \longleftarrow \mathcal{O}_{\mathsf{init}}(\lambda, n_T, t)$;
$\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}, \mathcal{O}_{\mathsf{board}}, \mathcal{O}_{\mathsf{voteLR}}, \mathcal{O}_{\mathsf{receiptLR}}}(\mathsf{pk})$;
$r, \Pi \longleftarrow \mathcal{O}_{\mathsf{tally}}()$;
$b' \longleftarrow \mathbb{A}(r, \Pi)$;
if $b' = b$ **then return** 1 **else return** 0;

$\mathcal{O}_{\mathsf{init}}(\lambda, n_T, t)$

$\tau \longleftarrow \bot$;
if $b = 0$ **then**
  $\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_i, \_ \longleftarrow \mathsf{Setup}(\lambda, n_T, t)$;
else
  $\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_i, \tau \longleftarrow \mathrm{SimSetup}(\lambda, n_T, t)$;
return $\mathsf{pk}, \mathsf{sk}, \tau$;

$\mathcal{O}_{\mathsf{cast}}(B)$

if $\mathrm{Valid}(B, \mathsf{PB}_0) = 1$ **and** $\mathrm{Valid}(B, \mathsf{PB}_1) = 1$ **then**
  $\mathsf{Append}(\mathsf{PB}_0, B)$; $\mathsf{Append}(\mathsf{PB}_1, B)$;

$\mathcal{O}_{\mathsf{board}}()$

return PB_b;

$\mathcal{O}_{\mathsf{voteLR}}(\mathsf{id}, \nu_0, \nu_1)$

if $\nu_0 \notin \mathcal{V}$ or $\nu_1 \notin \mathcal{V}$ **then return** $\bot$;
$s_0, B_0 \longleftarrow \mathsf{Vote}(\mathsf{id}, \nu_0, \mathsf{RS}, \mathsf{PB})$;
$s_1, B_1 \longleftarrow \mathsf{Vote}(\mathsf{id}, \nu_1, \mathsf{RS}, \mathsf{PB})$;
$\mathsf{Append}(\mathsf{PB}_0, B_0)$; $\mathsf{Append}(\mathsf{PB}_1, B_1)$;

$\mathcal{O}_{\mathsf{receiptLR}}(\mathsf{id}, \mathsf{I}, \nu)$

if $\nu \notin \mathcal{V}$ **then return** $\bot$;
$s_0, B_0 \longleftarrow \mathsf{Vote}(\mathsf{id}, \mathsf{I}, \mathsf{RS}, \mathsf{PB})$;
$s_1, B_1 \longleftarrow \mathsf{Vote}(\mathsf{id}, \mathcal{D}^{\mathsf{I}}(\nu), \mathsf{RS}, \mathsf{PB})$;
$\mathsf{Append}(\mathsf{PB}_0, B_0)$; $\mathsf{Append}(\mathsf{PB}_1, B_1)$;
return s_b;

$\mathcal{O}_{\mathsf{tally}}()$

$r_0, \Pi_0 \longleftarrow \mathsf{Tally}(\mathsf{PB}_0, \mathsf{sk})$;
$\Pi_1 \longleftarrow \mathrm{SimProof}(\mathsf{PB}_1, r_0, \tau)$;
return r_0, Π_b;

Figure 31: The receipt-free experiment and its oracles, where $\mathcal{V}$ is the set of the voting options.

with respect to the current board PB_b; however, this leads to a definition glitch as discussed in Section 1.3.2.

The oracle $\mathcal{O}_{\mathsf{voteLR}}(\mathsf{id}, \nu_0, \nu_1)$ is similar to that of the BPRIV definition, and causes the honest voter id to vote with the voting option ν_0 when $b = 0$ and ν_1 when $b = 1$.

The oracle $\mathcal{O}_{\mathsf{receiptLR}}(\mathsf{id}, \mathsf{I}, \nu)$ takes as input a honest voter id, a compatible instruction I and a voting intent ν. When $b = 0$, id follows the instruction I which leads to the creation of a ballot B_0 with the receipt s_0. Note that since we only consider compatible instructions, B_0 is necessarily a valid ballot. Also, recall that the ballot is created in interaction with the server, and might be a rerandomized version of a ballot sent by the voter. Hence, when the voter follows the instructions, they actually act as a dummy router that forwards the messages produced by I (seen as a Turing machine) to RS and the other way around. When $b = 1$, however, the voter runs the deceiving algorithm $\mathcal{D}$ in interaction with I, and votes with the voting option ν.

Finally, the oracle $\mathcal{O}_{\mathsf{tally}}$ is similar to that of the BPRIV definition: it computes the result (r, Π) of the tally with respect to the board PB_0 and return r, Π_b where Π_b is either Π when $b = 0$ or a simulated transcript when $b = 1$.

9.2 Introduction to traceable encryptions

To achieve receipt-freeness, we propose to use the notion of *traceable encryption*, introduced in [DPP22b]. Indeed, a traceable encryption scheme can provide various security properties that are useful in the context of verifiable receipt-free voting. In this section, we give all the definitions related to traceable encryptions.

9.2.1 Definition

A traceable encryption is a public key encryption scheme $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$, augmented with the additional algorithms $(\mathrm{LGen}, \mathrm{LEnc}, \mathrm{Trace}, \mathrm{Rand}, \mathrm{Ver})$, where:

- $\mathrm{LGen}(\mathsf{pk})$ is the link generation algorithm, which generates a random link key lk.
- $\mathrm{LEnc}_{\mathsf{pk}}(\mathsf{lk}, m, r)$ is the linked encryption algorithm, which outputs an encryption C of the plaintext m, obtained with the randomness r.
- $\mathrm{Trace}_{\mathsf{pk}}(C)$ is the tracing algorithm, which outputs the *trace* of a ciphertext C.
- $\mathrm{Rand}_{\mathsf{pk}}(C, r)$ is the rerandomization algorithm, which outputs a rerandomization C' of the ciphertext C, obtained with the randomness r.
- $\mathrm{Ver}_{\mathsf{pk}}(C)$ is a verification algorithm, which outputs either 0 or 1.

Intuitively, when a voter wants to cast a ballot for a voting option m, they first generate a random link key lk using LGen, then use LEnc to encrypt m. This produces a ciphertext C, which is rerandomized into C' by the server. Then, the voter can verify that $\mathrm{Trace}_{\mathsf{pk}}(C') = \mathrm{Trace}_{\mathsf{pk}}(C)$, so that it gains some guarantee that C' contains the same plaintext m, unless the server managed to find lk. Since the voter knows lk, however, the voter is able to forge a ciphertext C_v such that $\mathrm{Trace}_{\mathsf{pk}}(C_v) = \mathrm{Trace}_{\mathsf{pk}}(C)$ for any voting option v, hence this guarantee cannot be used to convince a third party that the ballot contains a specific voting option. Just as any encryption scheme, a traceable encryption must verify some *correctness* properties.

Definition 25 (Correctness)**.** *A tuple* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{LGen}, \mathrm{LEnc}, \mathrm{Trace}, \mathrm{Rand}, \mathrm{Ver})$ *has correctness if:*

- *For all key pair* $(\mathsf{pk}, \mathsf{sk})$ *output by* Gen*, for all plaintext* m *and all randomness* r*, we have* $\mathrm{Dec}_{\mathsf{sk}}(\mathrm{Enc}_{\mathsf{pk}}(m, r)) = m$.
- *The algorithm* $\mathrm{Enc}_{\mathsf{pk}}(m, r)$ *consists of first sampling a random* lk *with* LGen*, then computing* $\mathrm{LEnc}_{\mathsf{pk}}(\mathsf{lk}, m, r)$.
- *For all public key* pk*, for all link key* lk*, for all plaintexts* m_0, m_1 *and all randomness* r_0, r_1, $\mathrm{Trace}_{\mathsf{pk}}(\mathrm{LEnc}_{\mathsf{pk}}(\mathsf{lk}, m_0, r_0)) = \mathrm{Trace}_{\mathsf{pk}}(\mathrm{LEnc}_{\mathsf{pk}}(\mathsf{lk}, m_1, r_1))$. *This is the link traceability property.*
- *For all key pair* $(\mathsf{pk}, \mathsf{sk})$*, for all ciphertext* C*, for all randomness* r*, we have* $\mathrm{Dec}_{\mathsf{sk}}(C) = \mathrm{Dec}_{\mathsf{sk}}(\mathrm{Rand}_{\mathsf{pk}}(C, r))$ *and* $\mathrm{Trace}_{\mathsf{pk}}(C) = \mathrm{Trace}_{\mathsf{pk}}(\mathrm{Rand}_{\mathsf{pk}}(C, r))$. *This is the publicly traceable rerandomization property.*
- *For all public key* pk*, for all plaintext* m *and for all randomness* r*,* $\mathrm{Ver}_{\mathsf{pk}}(\mathrm{Enc}_{\mathsf{pk}}(m, r)) = 1$.

We note that the link traceability property ensures that the trace of a ciphertext does not depend on the message that is encrypted: intuitively, this means that the link key cannot be used as a receipt by a malicious voter.

9.2.2 Security notions for verifiable receipt-free voting

In electronic voting, encrypting the voting options is not sufficient to obtain privacy. Indeed, we already mentioned that the encryption scheme must provide some additional security properties, such as NM-CPA security. Similarly, using a traceable encryption is not enough to obtain receipt-freeness. For this purpose, [DPP22b] introduces several security properties which are verifiability, TCCA security, traceability and strong rerandomization. Compared to the original definition of verifiability, this thesis uses a slightly weaker version, as it is sufficient for our purpose.

Definition 26 (Verifiability)**.** *A traceable encryption is* verifiable *if, for every efficient adversary* $\mathbb{A}$*, the probability* $\Pr\left((\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{Gen}(\lambda); C \leftarrow \mathbb{A}(\mathsf{pk}, \mathsf{sk}); \mathrm{Ver}_{\mathsf{pk}}(C) = 1 \wedge \mathrm{Dec}_{\mathsf{sk}}(C) \notin \mathscr{P}\right)$ *is negligible in* λ*, where* $\mathscr{P}$ *is the plaintext space.*

In the context of electronic voting, the public verifiability of the encryption scheme allows to ensure that an encrypted ballot indeed contains a valid voting option.

Definition 27 (TCCA)**.** *A traceable encryption scheme is secure against traceable chosen-ciphertext attacks (TCCA-secure) if, for all PPT adversary* $\mathbb{A}$*, the advantage of* $\mathbb{A}$ *in the experiment defined in Algorithm 106 is negligible in* λ.

The TCCA-security game can be read as follows. First, a random key pair $(\mathsf{pk}, \mathsf{sk})$ is generated by the challenger, and the adversary is given the public key. In addition, it has access to a decryption oracle, which can decrypt any well-formed ciphertext (*i.e.* an element C s.t. $\mathrm{Ver}_{\mathsf{pk}}(C) = 1$). With this oracle, the adversary must output two valid ciphertexts C_0 and C_1 that share the same trace, and the challenger rerandomizes one of them at random, yielding the challenge ciphertext $C^\star$. At this point, the adversary must guess whether C_0 and C_1 has been rerandomized; for this purpose, it can make queries to the decryption oracle $\mathcal{O}_{\mathrm{Dec}^\star}$. Similarly to $\mathcal{O}_{\mathsf{Dec}}$, this oracle can only decrypt valid ciphertexts; however, it cannot decrypt a ciphertext that has the same trace as $C^\star$.

The intuition is that if the traceable encryption scheme is TCCA, then even if one is instructed to submit a specific ciphertext C_1, the rerandomization $C^\star$ of C_1 is indistinguishable from that of any other ciphertext C_0, provided that C_0 and C_1 have the same trace.

Algorithm 106: $\mathrm{Exp}^{\mathsf{tcca}}(\lambda, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;
$C_0, C_1 \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathrm{Dec}}}(\mathsf{pk})$;
$b \xleftarrow{\$} \{0,1\}$;
if $\mathrm{Trace}_{\mathsf{pk}}(C_0) \neq \mathrm{Trace}_{\mathsf{pk}}(C_1)$ **or** $\mathrm{Ver}_{\mathsf{pk}}(C_0) = 0$ **or** $\mathrm{Ver}_{\mathsf{pk}}(C_1) = 0$ **then return** b;
$r \xleftarrow{\$} \mathscr{R}$;
$C^\star \longleftarrow \mathrm{Rand}_{\mathsf{pk}}(C_b, r)$;
$b' \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathrm{Dec}^\star}}(C^\star)$;
if $b = b'$ **then return** 1 **else return** 0;

Algorithm 107: $\mathrm{Exp}^{\mathsf{trace}}(\lambda, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;
$m \longleftarrow \mathbb{A}(\mathsf{pk}, \mathsf{sk})$;
$r \xleftarrow{\$} \mathscr{R}$;
$C \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(m, r)$;
$C^\star \longleftarrow \mathbb{A}(C)$;
if $m \in \mathscr{P}$ **and** $\mathrm{Ver}_{\mathsf{pk}}(C^\star) = 1$ **and** $\mathrm{Trace}_{\mathsf{pk}}(C) = \mathrm{Trace}_{\mathsf{pk}}(C^\star)$ **and** $\mathrm{Dec}_{\mathsf{sk}}(C^\star) \neq m$ **then return** 1;
else return 0;

Another interesting property is the traceability, which states that given a honestly generated ciphertext, one cannot forge another ciphertext that shares the same trace but does not encrypt the same value. In the context of electronic voting, this property prevents the rerandomization server from modifying the votes.

Definition 28 (Traceability)**.** *A traceable encryption is traceable if, for all PPT adversary* $\mathbb{A}$*, the probability* $\Pr(\mathrm{Exp}^{\mathsf{trace}}(\lambda, \mathbb{A}) = 1)$ *is negligible in* λ*, where* $\mathrm{Exp}^{\mathsf{trace}}$ *is defined in Algorithm 107.*

Finally, the strong rerandomization means that a rerandomized ciphertext follows the same distribution as a fresh ciphertext output by LEnc. In other words, it means that the rerandomization process is not supposed to "add" some entropy: even with no rerandomization, the encryption scheme itself would be sementically secure, so that the rerandomization server does not have to be trusted for the purpose of privacy.

Definition 29 (Strong rerandomization)**.** *A traceable encryption is strongly rerandomizable if, for all plaintext* m*, for all link key* lk*, for all randomness* r_0*, if* $C_0 = \mathrm{LEnc}_{\mathsf{pk}}(\mathsf{lk}, m, r_0)$ *and if* R *is the uniform random variable over the randomness space, we have the following computational indistinguishability:*

$$\mathrm{Rand}_{\mathsf{pk}}(C_0, R) \approx \mathrm{LEnc}_{\mathsf{pk}}(\mathsf{lk}, m, R).$$

9.3 Building blocks

We now provide the building blocks that we used to construct a traceable encryption scheme that achieves all the desired properties. They are the same as the ones proposed in [DPP22b], where an instantiation based on bilinear maps is mentioned.

9.3.1 Bilinear pairings

Let $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ be three groups of prime order q, and g (resp. $\hat{g}$) be a generator of $\mathbb{G}$ (resp. $\hat{\mathbb{G}}$). A *pairing* is a bilinear map $e : \mathbb{G} \times \hat{\mathbb{G}} \longrightarrow \mathbb{G}_T$, and we say that $\mathbb{G}, \hat{\mathbb{G}}$ are *pairing-friendly* if there exists a non-trivial, efficiently computable pairing. On this occasion, recall that we use a multiplicative notation, so that for all $a, b \in \mathbb{G}$, $\hat{c} \in \hat{\mathbb{G}}$ and $\alpha, \beta \in \mathbb{Z}_q$, we have $e(a^\alpha b^\beta, \hat{c}) = e(a, \hat{c})^\alpha e(b, \hat{c})^\beta$. In addition, it is usual to denote $\iota : \mathbb{G} \to \mathbb{G}^2$ the injection that maps $X \in \mathbb{G}$ to $\iota(X) = (X, 1)$.

Respectively, we define the $\hat{\imath} : \hat{\mathbb{G}} \to \hat{\mathbb{G}}^2$ the injection that maps $\hat{X} \in \hat{\mathbb{G}}$ to $\hat{\imath}(\hat{X}) = (\hat{X}, 1)$. Finally, in our construction, we rely on the SXDH assumption.

The SXDH assumption. The SXDH assumption is a classical computational assumption in pairings-friendly groups, defined, for instance, in [BGdMM05]. Intuitively, it states that the DDH problem is hard in both $\mathbb{G}$ and $\hat{\mathbb{G}}$. Note that since there is an efficiently computable pairing, the SXDH assumption is not true if $\mathbb{G} = \hat{\mathbb{G}}$ or, alternatively, if there exists an efficiently computable group isomorphism from $\mathbb{G}$ to $\hat{\mathbb{G}}$ (or the other way around).

9.3.2 Linearly Homomorphic Structure-Preserving Signatures

A central tool for our construction is the linearly homomorphic structure-preserving (LHSP) signature. The structure preserving property, defined, for instance, in [AFG+10], allows to sign messages that respect a vector space structure. The additional linearly homomorphic feature, introduced in [LPJY13], allows to derive a signature on any linear combination of already signed vectors. In what follows, we describe the LHSP signature that we used in our construction.

Description of the LHSP scheme. Just as an encryption scheme, a signature scheme is defined by a tuple of algorithms, *i.e.* $(\mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify})$. In the LHSP scheme that we consider, they are defined as follows.

- $\mathbf{Keygen(pp, \boldsymbol{n})}$**.** Given the dimension n of the vector space and the public parameters pp (here, pp contains two indepent group elements $\hat{g}, \hat{h} \in \hat{\mathbb{G}}$), the key generation algorithm picks $(\chi_i, \gamma_i)_{i=1}^n \in \mathbb{Z}_q$ at random and computes $\hat{g}_i = \hat{g}^{\chi_i} \hat{h}^{\gamma_i}$ for $i = 1$ to n. It returns the pair $(\mathsf{pk}, \mathsf{sk})$, where the private key is $\mathsf{sk} = (\chi_i, \gamma_i)_{i=1}^n$ and the public key is $\mathsf{pk} = (\hat{g}_i)_{i=1}^n$.
- $\mathbf{Sign_{sk}(\boldsymbol{M_1, \cdots, M_n})}$**.** To sign a vector $\overrightarrow{M} \in \mathbb{G}^n$ using $\mathsf{sk} = (\chi_i, \gamma_i)_{i=1}^n$, one returns $\sigma = (Z, R)$ with $Z = \prod_{i=1}^n M_i^{\chi_i}$ and $R = \prod_{i=1}^n, M_i^{\gamma_i}$.
- $\mathbf{Verify_{pk}(\boldsymbol{M_1, \cdots, M_n, \sigma})}$**.** To verify that a signature $\sigma = (Z, R)$ is valid with respect to a public key $\mathsf{pk} = (\hat{g}_i)_{i=1}^n$, one checks the equality

$$e(Z, \hat{g}) e(R, \hat{h}) = \prod_{i=1}^{n} e(M_i, \hat{g}_i). \tag{4}$$

Unforgeability. It is easy to check that the signature scheme is correct, *i.e.* that a honestly generated signature is considered valid by the Verify algorithm. A less trivial property is that of the unforgeability of linearly independent messages. More precisely, we give Definition 30 which is adapted from [LPJY13]. In this definition, we consider some honestly generated public parameters pp and we let the adversary choose $n \geq 1$, the dimension of the vector space. Then the adversary has access to a signing oracle that it can query to obtain valid signatures of some chosen messages. The signing oracle updates the set V of all the plaintexts that the adversary queried, and the goal of the adversary is to output a valid signature of a message M which is not in the vector space spanned by V, denoted $< V >$. In other words, M must be linearly independent from all the signed messages obtained by the adversary. Note that if (Z_1, R_1) and (Z_2, R_2) are two valid signatures of $M_1, \cdots, M_n$ and $N_1, \cdots, N_n$, then, for all $\alpha, \beta \in \mathbb{Z}_q$, $(Z_1^\alpha Z_2^\beta, R_1^\alpha R_2^\beta)$ is a valid signature of $(M_1^\alpha N_1^\beta, \cdots, M_n^\alpha N_n^\beta)$. In particular, $(1, 1)$ is a valid signature of $(1, \cdots, 1)$. Hence, the linear independence is required for the unforgeability.

Definition 30. *A LHSP scheme is secure if, for all PPT adversary $\mathbb{A}$, the probability that $\mathbb{A}$ wins the experiment* $\mathrm{Exp}^{\mathsf{lin\text{-}unf}}$ *(defined in Figure 32) is negligible in λ.*

$\mathrm{Exp}^{\mathsf{lin\text{-}unf}}(\lambda, \mathbb{A})$

Requires: Setup, an algorithm that generates the public parameters

1 $\mathsf{pp} \longleftarrow \mathsf{Setup}(\lambda)$; $V \longleftarrow \varnothing$;
2 $1^n \longleftarrow \mathbb{A}(\mathsf{pp})$;
3 $(\mathsf{pk}, \mathsf{sk}) \longleftarrow \mathsf{Keygen}(\mathsf{pp}, n)$;
4 $M_1, \cdots, M_n, \sigma \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathsf{Sign}}}(\mathsf{pk})$;
5 **if** $\overrightarrow{M} \notin < V >$ **and** $\mathsf{Verify}_{\mathsf{pk}}(M_1, \cdots, M_n, \sigma) = 1$ **then return** 1;
6 **else return** 0;

$\mathcal{O}_{\mathsf{Sign}}(M_1, \cdots, M_n)$

1 $V \longleftarrow V \bigcup \{(M_1, \cdots, M_n)\}$;
2 $\sigma \longleftarrow \mathsf{Sign}_{\mathsf{sk}}(M_1, \cdots, M_n)$;
3 **return** σ

Figure 32: Definition of unforgeability of linearly independent messages

We now prove that the above LHSP scheme is secure under the SXDH assumption. The proof is extremely similar to that of [LPJY13].

Lemma 16. *Under the SXDH assumption, the LHSP signature scheme presented in this section is secure.*

Proof. Let $\mathbb{A}$ be an adversary for $\mathrm{Exp}^{\mathsf{lin\text{-}unf}}$. We construct an adversary $\mathbb{B}$ against SXDH, that interacts with $\mathbb{A}$ as follows. The SXDH assumption states that the DDH problem is hard in both $\mathbb{G}$ and $\hat{\mathbb{G}}$. Hence, we only need to construct an adversary for the DDH game in $\hat{\mathbb{G}}$. First, $\mathbb{B}$ gets a challenge tuple in the DDH game, that we denote $\hat{g}, \hat{h}, \hat{g}_1, \hat{h}_1 \in \hat{\mathbb{G}}$. It plays $\hat{g}, \hat{h}$ in the unforgeability game, as the public parameters pp. Then, $\mathbb{A}$ chooses n and $\mathbb{B}$ picks a random secret key $(\chi_i, \gamma_i)_{i=1}^n$ as well as the corresponding public key $\mathsf{pk} = (\hat{g}_i)_{i=1}^n$, with $\hat{g}_i = \hat{g}^{\chi_i} \hat{h}^{\gamma_i}$ for all i.

$\mathbb{B}$ sends pk to $\mathbb{A}$ and uses sk to simulate the signing oracle. It also updates V, the set of the messages that it signs for $\mathbb{A}$. At some point, $\mathbb{A}$ outputs $M_1, \cdots, M_n, \sigma$, and $\mathbb{B}$ signs $M_1, \cdots, M_n$ to generate another signature σ'.

If $\mathbb{A}$ does not win the unforgeability game, $\mathbb{B}$ guesses at random. If $\mathbb{A}$ wins the unforgeability game, then $\sigma = (Z, R)$ and $\sigma' = (Z', R')$ are two valid signatures for the same message $M_1, \cdots, M_n$, so that

$$e(Z, \hat{g}) e(R, \hat{h}) = \prod_{i=1}^{n} e(M_i, \hat{g}_i) = e(Z', \hat{g}) e(R', \hat{h}).$$

Hence, $e(Z/Z', \hat{g}) = e(R'/R, \hat{h})$ and $\mathbb{B}$ can proceed as follows: if $e(Z/Z', \hat{g}_1) = e(R'/R, \hat{h}_1)$, it states that $\hat{g}, \hat{h}, \hat{g}_1, \hat{h}_1$ was a DDH tuple. Otherwise, it states that it was a random tuple.

Now, let E be the event in which $\mathbb{A}$ wins the unforgeability game and ε the probability that E occurs. When E occurs, (Z, R) and (Z', R') are valid signatures so that $e(Z/Z', \hat{g}) = e(R'/R, \hat{h})$. Hence, if $(\hat{g}, \hat{h}, \hat{g}_1, \hat{h}_1)$ is a DDH tuple, we also have $e(Z/Z', \hat{g}_1) = e(R/R', \hat{h}_1)$ so that $\mathbb{B}$ wins the DDH game with probability 1. In addition, when E occurs, $(M_1, \cdots, M_n) \notin < V >$ so that we can obtain a basis of $< V >$ of size at most $n - 1$. Given V and the corresponding signatures, a computationally unbounded adversary has no more information about the secret key than that

contained in $2n-1$ linear equations: at most $n-1$ equations are given by the validity of the signatures, and are of the form

$$e(\prod_{i=1}^{n} N_i^{\chi_i}, \hat{g}) e(\prod_{i=1}^{n}, N_i^{\gamma_i}, \hat{h}) = \prod_{i=1}^{n} e(N_i, \hat{g}_i)$$

for some N_i's and n equations comes from the public key, and are of the form $\hat{g}_i = \hat{g}^{\chi_i}\hat{h}^{\gamma_i}$. Hence, when E occurs, the probability that $(Z,R) = (Z',R')$ is at most $1/q$. Yet, if $(\hat{g},\hat{h},\hat{g}_1,\hat{h}_1)$ is a random tuple and if $(Z,R) \neq (Z',R')$, then $e(Z/Z',\hat{g}_1) = e(R'/R,\hat{h}_1)$ occurs with probability $1/q$. Therefore, there exists a negligible function μ, that respects $0 \leq \mu \leq \frac{2}{q}$, such that, when E occurs while $(\hat{g},\hat{h},\hat{g}_1,\hat{h}_1)$ is a random tuple, $\mathbb{B}$ wins the DDH game with probability $1-\mu$.

Overall, the probability that $\mathbb{B}$ wins the DDH game is

$$\frac{1}{2}(1-\varepsilon) + \frac{1}{2}\varepsilon(1+1-\mu) = \frac{1}{2} + \frac{1}{2}\varepsilon - \frac{1}{2}\varepsilon\mu.$$

This shows that $\varepsilon \leq 2\varepsilon_{SXDH} + \frac{2}{q}$, where ε_{SXDH} is the advantage of $\mathbb{B}$ in the DDH game. By the SXDH assumption, ε is negligible. □

9.3.3 The Groth-Sahai proof system

In the context of bilinear-friendly groups, the Groth-Sahai proofs were introduce in [GS08] and can be used to prove that quadratic equations are verified on committed values. In what follows, we focus on *pairing product* equations, which have the form

$$\prod_{i=1}^{n} e(X_i, \hat{B}_i) \prod_{j=1}^{m} e(A_j, \hat{Y}_j) \prod_{i=1}^{n}\prod_{j=1}^{m} e(X_i, \hat{Y}_j)^{\gamma_{ij}} = T,$$

where $X_1,\ldots,X_n \in \mathbb{G}$ and $\hat{Y}_1,\ldots,\hat{Y}_m \in \hat{\mathbb{G}}$ are committed secret values; $\hat{B}_1,\ldots,\hat{B}_n \in \hat{\mathbb{G}}$, $A_1,\ldots,A_m \in \mathbb{G}$ and $T \in \mathbb{G}_T$ are public group elements and, for $1 \leq i \leq n$ and $1 \leq j \leq m$, $\gamma_{i,j} \in \mathbb{Z}_q$ are public scalars. For instance, the verification equation (4) of the LHSP signature is a pairing product equation.

The Groth-Sahai proof system provides, among others, several algorithms to commit to secret values and prove that pairing product equations hold on the committed values. In what follows, we do not fully detail the construction of Groth-Sahai proofs based on SXDH, but only what is necessary for our purpose.

- **Gen(pp) :** given the public parameter pp, choose $\vec{u}_1 = (u_{1,1}, u_{1,2})$, $\vec{u}_2 = (u_{2,1}, u_{2,2}) \in \mathbb{G}^2$, $\vec{\hat{v}}_1 = (\hat{v}_{1,1}, \hat{v}_{1,2})$ and $\vec{\hat{v}}_2 = (\hat{v}_{2,1}, \hat{v}_{2,2}) \in \hat{\mathbb{G}}^2$ at random. The common reference string (CRS) of the proof system is given by $(\vec{u}_1, \vec{u}_2, \vec{\hat{v}}_1, \vec{\hat{v}}_2)$.

- **Com(crs, $X, \vec{r}$) :** to commit to an element $X \in \mathbb{G}$ (resp. $\hat{Y} \in \hat{\mathbb{G}}$) with randomness $\vec{r} = (r_1, r_2) \in \mathbb{Z}_q^2$, compute $\vec{C} = \iota(X)\vec{u}_1^{r_1}\vec{u}_2^{r_2}$ (resp. $\vec{\hat{C}} = \hat{\iota}(\hat{Y})\vec{\hat{v}}_1^{r_1}\vec{\hat{v}}_2^{r_2}$).

- **CRand(crs, $\vec{C}, \vec{r'}$) :** to rerandomize a commitment $\vec{C} \in \mathbb{G}$ (resp. $\vec{\hat{C}} \in \hat{\mathbb{G}}$) with randomness $\vec{r'} = (r'_1, r'_2)$, compute $\vec{C}' = \vec{C}\vec{u}_1^{r'_1}\boldsymbol{u}_2^{r'_2}$ (resp. $\vec{\hat{C}}' = \vec{\hat{C}}\vec{v}_1^{r'_1}\vec{v}_2^{r'_2}$).

- **Prove(crs, $E, \mathcal{W}, \rho$) :** Given a pairing product equation E and a set of witnesses that verify the equation $\mathcal{W} = \{(X_i, \vec{r}_i)_{i=1}^m, (\hat{Y}_j, \hat{\vec{r}}'_j)_{j=1}^n\}$ (the $\vec{r}_i$'s and the $\hat{\vec{r}}'_j$'s are the randomness used to produce the respective commitments), the proving algorithm produces a proof $\vec{\pi}, \vec{\hat{\pi}} \in \mathbb{G}^4 \times \hat{\mathbb{G}}^4$ that the witnesses verify the equation. If the equation is linear in $\mathbb{G}$ (resp. $\hat{\mathbb{G}}$), the proof lies in $\hat{\mathbb{G}}^2$ (resp. $\mathbb{G}^2$).

- **PRand(crs, $\vec{C}', \vec{\pi}, \vec{\hat{\pi}}, \vec{r}'$) :** Given the randomness $\vec{r}'$ used to rerandomize the commitment $\vec{C}$ into $\vec{C}'$, it is possible to rerandomize $\vec{\pi}, \vec{\hat{\pi}}$ into $\vec{\pi}', \vec{\hat{\pi}}'$.

- **Verify(crs, $E, \mathcal{C}, \vec{\pi}, \vec{\hat{\pi}}$) :** Given a CRS, an equation, some commitments $\mathcal{C}$ (one commitment in $\mathbb{G}^2$ for each witness in $\mathbb{G}$ and one commitment in $\hat{\mathbb{G}}^2$ for each witness in $\hat{\mathbb{G}}$) and a proof $\vec{\pi}, \vec{\hat{\pi}}$, the verification algorithm returns 1 if the proof is valid and 0 otherwise.

The Groth-Sahai proofs have various well-known interesting properties. For our purpose, we mention the following properties, and we refer to [GS07] for more details.

Correctness. If the witnesses verify the equation, then running Com and Prove generates a valid tuple of proof and commitments. In addition, the rerandomization of a valid tuple of proofs and commitments is also valid.

Linearity. Thanks to the algorithms CRand and PRand, the Groth-Sahai proofs are rerandomizable. More generally, they are known to be malleable [Fuc11, CKLM12]. In our construction, we do not need the generic malleability theory, and we only use the following facts:

1. Let $A_1, A'_1, \cdots, A_{m-1}, A'_{m-1} \in \mathbb{G}$ be some group elements that define the equations (5) and (6), let $\vec{\hat{C}}_1, \cdots, \vec{\hat{C}}_m$ be some commitments on witnesses that verify both equations and let $\vec{\pi}_1, \vec{\pi_2}$ be two valid Groth-Sahai proofs w.r.t. those equations and commitments. Then, for all $\theta' \in \mathbb{Z}_q$, the proof $\vec{\pi}_1 \vec{\pi_2}^{\theta'}$ is a valid proof for Eq. (7) with respect to the commitments $\vec{\hat{C}}_1, \cdots, \vec{\hat{C}}_{m-2}, \vec{\hat{C}}_{m-1} \vec{\hat{C}}_m^{\theta'}$.

$$\prod_{j=1}^{m-2} e(A_j, \hat{Y}_j) = e(A_{m-1}, \hat{Y}_{m-1}) \tag{5}$$

$$\prod_{j=1}^{m-2} e(A'_j, \hat{Y}_j) = e(A_{m-1}, \hat{Y}_m) \tag{6}$$

$$\prod_{j=1}^{m-2} e(A_j A_j'^{\theta'}, \hat{Y}_j) = e(A_{m-1}, \hat{Y}_{m-1}) \tag{7}$$

2. Let $\hat{B}_1, \cdots, \hat{B}_n \in \hat{\mathbb{G}}$ be some group elements and, for $t \in \mathbb{G}_T$, let E_t be the equation (8). Let $t \in \mathbb{G}_T$ and $\vec{C}_1, \cdots, \vec{C}_n$ be some commitments on witnesses that verify the equation E_t, and $\hat{\pi}$ a valid proof w.r.t. E_t and those commitments. Let $t' \in \mathbb{G}_T$ and $w_1, \cdots, w_n \in \mathbb{G}$ be some witnesses that verify the equation $E_{t'}$. Then, for all $\theta' \in \mathbb{Z}_q$, the proof $\hat{\pi}$ is a valid proof w.r.t. $E_{tt'^{\theta'}}$ and the commitment $\vec{C}_1 \iota(w_1^{\theta'}), \cdots, \vec{C}_n \iota(w_n^{\theta'})$.

$$\prod_{i=1}^{n} e(X_i, \hat{B}_i) = t \tag{8}$$

Perfect rerandomization. Let $\vec{w}, \vec{\hat{w}}$ be a vector of witnesses for the equation E, $\mathcal{C}$ be some commitments and $\vec{\pi}, \vec{\hat{\pi}}$ a proof obtained with the Prove algorithm. Then the tuple $\mathcal{C}_1, \vec{\pi}_1, \vec{\hat{\pi}}_1$

obtained by running Com and Prove follows the same distribution as the tuple $\mathcal{C}_2, \vec{\pi}_2, \vec{\hat{\pi}}_2$ obtained by rerandomizing $\mathcal{C}$ and $\vec{\pi}, \vec{\hat{\pi}}$.

Witness indistinguishability. Let $\vec{w}_1, \vec{\hat{w}}_1$ and $\vec{w}_2, \vec{\hat{w}}_2$ be some witnesses for the equation E, and let $\vec{u}_1, \vec{u}_2$ and $\vec{\hat{v}}_1, \vec{\hat{v}}_2$ the corresponding CRS. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be some honestly generated commitments on those witnesses (*i.e.* using the commitment algorithm with uniformly random r) and $\vec{\pi}_1, \vec{\hat{\pi}}_1$ and $\vec{\pi}_2, \vec{\hat{\pi}}_2$ two honestly generated Groth-Sahai proofs (*i.e.* using the proving algorithm with a uniformly random ρ). Then, unless $\vec{u}_1, \vec{u}_2$ or $\vec{\hat{v}}_1, \vec{\hat{v}}_2$ is a DDH tuple, the tuple $\mathcal{C}_1, \vec{\pi}_1, \vec{\hat{\pi}}_1$ follows the same distribution as the tuple $\mathcal{C}_2, \vec{\pi}_2, \vec{\hat{\pi}}_2$. More precisely, they are uniformly distributed in the space of the tuples that satisfy the verification equation.

Extractability. Suppose that there exists τ and $\hat{\tau}$ such that $\vec{u}_1^{\tau} = \vec{u}_2$ and $\vec{\hat{v}}_1^{\hat{\tau}} = \vec{\hat{v}}_2$ (*i.e.* $\vec{u}_1, \vec{u}_2$ and $\vec{\hat{v}}_1, \vec{\hat{v}}_2$ are two DDH tuples). Then, for all product pairing equation E and from any tuple $\mathcal{C}, \vec{\pi}, \vec{\hat{\pi}}$ such that $\vec{\pi}, \vec{\hat{\pi}}$ is valid, it is possible to efficiently extract (using τ and $\hat{\tau}$) some witnesses $\vec{w}, \vec{\hat{w}}$ that verify the equation E.

9.4 Construction of a traceable encryption scheme

In [DPP22b], a construction of a traceable encryption scheme is proposed, and achieves verifiability, TCCA-security, traceability and strong rerandomization. However, the verifiability of the encryption scheme is limited as the plaintext space is large, which can be a problem in electronic voting. Indeed, to apply the usual homomorphic tally, it is often necessary that the plaintexts have a specific form, which is enforced with 0/1 proofs. By contrast, when any group element is a valid plaintext, it is often required that the tally relies on a mixnet, which may be too restrictive, leak more information than a homomorphic tally and enable, among others, forced abstention attacks based on specific write-ins. Although we do not consider forced abstention attacks in receipt freeness (nor any attack based on an information available in the result, such as Italian attacks), we consider that it is interesting to provide an alternative encryption scheme, which supports 0/1 proofs. This way, the protocol designer can decide whether they want to use a homomorphic tally or a mixnet, and use either our construction or that of [DPP22b].

Another advantage of our construction is that the key generation algorithm is public coin: in other words, the CRS required can be derived from the hash of some public elements (for instance, the group specification or the public key), which was not the case in the construction of [DPP22b]. This is extremely interesting in the context of electronic voting, since this means that we need fewer trust assumptions.

- **$\mathsf{Gen}(\lambda, \ell)$:** Given the security parameter λ and the dimension ℓ, the key generation algorithm proceeds as follows:

  1. Generate $g, h, (g_i)_{i=1}^{\ell}, S, T \in \mathbb{G}$ and $\hat{g}, \hat{h} \in \hat{\mathbb{G}}$ at random.
  2. For $i = 1$ to ℓ, pick some random $(\alpha_i, \beta_i) \xleftarrow{\$} \mathbb{Z}_q$ and set $f_i = g^{\alpha_i} h^{\beta_i}$.
  3. Generate two random Groth-Sahai CRS $\vec{\mathsf{crs}}$ and $\vec{\mathsf{crs}}'$ where the elements of $\hat{\mathbb{G}}$ in $\vec{\mathsf{crs}}'$ are dropped. That is, $\vec{\mathsf{crs}} = (\vec{u}_1, \vec{u}_2, \vec{\hat{v}}_1, \vec{\hat{v}}_2)$ and $\vec{\mathsf{crs}}' = (\vec{u}'_1, \vec{u}'_2)$ where $\vec{u}_1, \vec{u}_2, \vec{u}'_1, \vec{u}'_2 \in \mathbb{G}^2$ and $\vec{\hat{v}}_1, \vec{\hat{v}}_2 \in \mathbb{G}^2$ are uniformly random group elements.

  The private key consists of $\mathsf{sk} = (\alpha_i, \beta_i)_{i=1}^{\ell}$ and the public key $\mathsf{pk} \in \mathbb{G}^{12+2\ell} \times \hat{\mathbb{G}}^6$ is

$$\mathsf{pk} = \left(g, \hat{g}, h, \hat{h}, S, T, (g_i, f_i)_{i=1}^{\ell}, \vec{\mathsf{crs}}, \vec{\mathsf{crs}}'\right) .$$

- **$\mathbf{LGen(pk)}$:** First, generate a LHSP key pair $(\mathsf{osk}, \hat{\mathsf{opk}})$ from the public parameters $\hat{g}, \hat{h}$ and the dimension $n = 3$: $\mathsf{osk}, \hat{\mathsf{opk}} \longleftarrow \mathsf{Keygen}((\hat{g}, \hat{h}), 3)$. Then, derive $(F, G, H) = \mathrm{hash}(\hat{\mathsf{opk}})$ from the public key and set $\mathsf{lk} = (\mathsf{osk}, \hat{\mathsf{opk}}, F, G, H)$.

- **$\mathbf{LEnc_{pk}(lk, \vec{m}, r)}$:** To encrypt the plaintext $\vec{m} = (m_1, \cdots, m_\ell) \in \{0,1\}^\ell$ with the link key $\mathsf{lk} = (\mathsf{osk}, \hat{\mathsf{opk}}, F, G, H)$ and the randomness r, conduct the following steps:

  1. Pick $\theta \xleftarrow{\$} \mathbb{Z}_q$ and compute the CPA encryption $\vec{c} = (c_1, c_2, (d_i)_{i=1}^\ell)$, where $c_1 = g^\theta$, $c_2 = h^\theta$ and $d_i = g_i^{m_i} f_i^\theta$, and keep the random coin θ.
     Afterwards, the steps 2-4 are dedicated to the traceability.
  2. Authenticate the row space of the matrix $\boldsymbol{T} = (T_{i,j})_{1 \le i,j \le 3}$ given below:

$$\boldsymbol{T} = \begin{pmatrix} g & \prod_{i=1}^\ell d_i & c_1 \\ 1 & \prod_{i=1}^\ell f_i & g \\ 1 & F & G \end{pmatrix}. \tag{9}$$

     Namely, sign each row $\vec{T}_i = (T_{i,1}, T_{i,2}, T_{i,3})$ of $\boldsymbol{T}$ using the LHSP signature. This results in $\vec{\sigma} = (\vec{\sigma}_i)_{i=1}^3 \in \mathcal{G}^6$, where $\vec{\sigma}_i = (Z_i, R_i) \in \mathbb{G}^2$.
  3. Using the Groth-Sahai commitment scheme and the CRS crs', commit to $\vec{\sigma}_1 = (Z_1, R_1)$. This gives $\vec{C}_Z, \vec{C}_R \in \mathbb{G}^2$. To ensure that $\vec{\sigma}_1$ is a valid one-time LHSP signature on $(g, \prod_{i=1}^\ell d_i, c_1)$ with respect to the public key $opk = (\hat{l}_1, \hat{l}_2, \hat{l}_3)$, compute the Groth-Sahai proof $\vec{\hat{\pi}}_{sig}$ that $e(Z_1, \hat{g})e(R_1, \hat{h}) = e(g, \hat{l}_1)e(\prod_{i=1}^\ell d_i, \hat{l}_2)e(c_1, \hat{l}_3)$.
  4. Set $\hat{a} = \hat{b} = 1_{\hat{\mathbb{G}}}$ and $\hat{w} = \hat{g}$ and commit to these elements, using the Groth-Sahai commitment scheme and the CRS $\vec{\mathsf{crs}}$. This gives $\vec{\hat{C}}_A, \vec{\hat{C}}_B, \vec{\hat{C}}_W \in \hat{\mathbb{G}}^2$. Compute the Groth-Sahai proof $\vec{\pi}_{SS}$ that $e(S, \hat{a})e(T, \hat{b}) = e(H, \hat{g}/\hat{w})$.
  5. Set $M_i = g^{m_i}$, $\hat{M}_i = \hat{g}^{m_i}$ for $i = 1, \ldots, \ell$ and $\hat{\Theta} = \hat{g}^\theta$ and commit to these elements with the CRS crs. This yields $\vec{C}_{M_1}, \vec{\hat{C}}_{M_1}, \cdots, \vec{C}_{M_\ell}, \vec{\hat{C}}_{M_\ell} \vec{\hat{C}}_\Theta \in \mathbb{G}^2$. Compute the Groth-Sahai proofs $\vec{\pi}_1$ that $e(c_1, \hat{w}) = e(g, \hat{\Theta})$, $\vec{\pi}_2$ that $e(c_2, \hat{w}) = e(h, \hat{\Theta})$ and, for all i, $\vec{\pi}_{d_i}$ that $e(d_i, \hat{w}) = e(g_i, \hat{M}_i)e(f_i, \hat{\Theta})$.
  6. To prove that each m_i is equal to 0 or 1, compute the Groth-Sahai proof $(\vec{\pi}_{01,i}, \vec{\hat{\pi}}_{01,i})_{i=1}^\ell$ that $e(g/M_i, \hat{M}_i) = 1$ and $e(M_i, \hat{g}) = e(g, \hat{M}_i)$.
  7. To allow strong randomization, commit to $\hat{x} = \hat{g}$ with the CRS crs. This gives the commitment $\vec{\hat{C}}_x$. Then, compute the Groth-Sahai proofs $\vec{\pi}_{r,1}$ that $e(g, \hat{w}) = e(g, \hat{x})$, $\vec{\pi}_{r,2}$ that $e(h, \hat{w}) = e(h, \hat{x})$ and, for all i, $\vec{\pi}_{r,d_i}$ that $e(f_i, \hat{w}) = e(f_i, \hat{x})$.

  Output the ciphertext

$$\begin{aligned} \mathsf{CT} = \Big(& \vec{c}, \vec{C}_Z, \vec{C}_R, \vec{\sigma}_2, \vec{\sigma}_3, \vec{\hat{\pi}}_{sig}, \hat{\mathsf{opk}}, \vec{\hat{C}}_A, \vec{\hat{C}}_B, \\ & \vec{\hat{C}}_W, \vec{\pi}_{SS}, (\vec{C}_{M_i}, \vec{\hat{C}}_{M_i})_{i=1}^\ell, \vec{\hat{C}}_\Theta, \vec{\hat{C}}_X, \vec{\pi}_1, \vec{\pi}_2, \\ & (\vec{\pi}_{d_i}, \vec{\pi}_{01,i}, \vec{\hat{\pi}}_{01,i})_{i=1}^\ell, \vec{\pi}_{r,1}, \vec{\pi}_{r,2}, (\vec{\pi}_{r,d_i})_{i=1}^\ell \Big), \end{aligned}$$

  which consists of $20 + 11\ell$ elements of $\mathbb{G}$ and $15 + 6\ell$ elements of $\hat{\mathbb{G}}$.

- **$\mathbf{Enc_{pk}(\vec{m}, r)}$:** To encrypt the plaintext $\vec{m} = (m_1, \cdots, m_\ell) \in \{0,1\}^\ell$ with the randomness r, first generates lk using LGen and return $\mathrm{LEnc}_{\mathsf{pk}}(\mathsf{lk}, \vec{m}, r)$.

- **$\mathbf{Trace_{pk}(CT)}$:** Parse pk and CT as above, and output $\hat{\mathsf{opk}}$.

- **$\mathbf{Rand_{pk}(CT)}$:** Pick $\theta' \xleftarrow{\$} \mathbb{Z}_q$ and do the following operations:

  1. Parse the CPA encryption part as $c_1, c_2, (d_i)_{i=1}^{\ell}$ and compute $\vec{c'} = (c_1', c_2', (d_i')_{i=1}^{\ell})$ where $c_1' = c_1 g^{\theta'}$, $c_2' = c_2 h^{\theta'}$ and $d_i' = d_i f_i^{\theta'}$ for $i = 1, \ldots, \ell$.
  2. Update $(\vec{C}_Z, \vec{C}_R)$, the committed signature $\vec{\sigma}_1$. That is, parse $\vec{\sigma}_2$ as (Z_2, R_2) and compute $\vec{C}'_Z = \vec{C}_Z \iota(Z_2^{\theta'})$ and $\vec{C}'_R = \vec{C}_R \iota(R_2^{\theta'})$. By the linearity of the LHSP scheme and the Groth-Sahai proof, the proof $\hat{\pi}_{sig}$ is valid w.r.t. $(g, \prod d_i', c_1')$ and $\hat{\mathsf{opk}}$.
  3. Update the proofs $\vec{\pi}_1$, $\vec{\pi}_2$ and $\vec{\pi}_{d_i}$. That is, compute $\vec{\pi}'_1 = \vec{\pi}_1 \vec{\pi}_{r,1}^{\theta'}$, $\vec{\pi}'_2 = \vec{\pi}_2 \vec{\pi}_{r,2}^{\theta'}$ and $\vec{\pi}'_{d_i} = \vec{\pi}_{d_i} \vec{\pi}_{r,d_i}^{\theta'}$ for all i. Also, compute $\vec{\hat{C}}'_\Theta = \vec{\hat{C}}_\Theta \vec{\hat{C}}_X^{\theta'}$. By the linearity of the Groth-Sahai proofs, the adapted proofs are now valid with regards to $\vec{c}'$, $\vec{\hat{C}}_W, \vec{\hat{C}}_{M_i}$ and $\vec{\hat{C}}'_\Theta$.
  4. Rerandomize all the commitments with CRand and the proofs with PRand. We note $\vec{C}''_Z, \vec{C}''_R, \vec{\hat{C}}'_A, \vec{\hat{C}}'_B, \vec{\hat{C}}'_W, \vec{C}'_{M_i}, \vec{\hat{C}}'_{M_i}, \vec{\hat{C}}'_X, \vec{\hat{C}}''_\Theta$ the resulting commitments and $\vec{\hat{\pi}}'_{sig}, \vec{\pi}'_{SS}, \vec{\pi}'_1$, $\vec{\pi}'_2, \vec{\pi}'_{d_i}, \vec{\pi}'_{i,01}, \vec{\hat{\pi}}'_{i,01}, \vec{\pi}'_{r,1}, \vec{\pi}'_{r,2}, \vec{\pi}'_{r,d_i}$ the resulting proofs.

  Return the re-randomized ciphertext

$$\mathsf{CT}' = \Big(\boldsymbol{c}', \vec{C}''_Z, \vec{C}''_R, \vec{\sigma}_2, \vec{\sigma}_3, \vec{\hat{\pi}}'_{sig}, \hat{\mathsf{opk}}, \vec{\hat{C}}'_A, \vec{\hat{C}}'_B, \vec{\hat{C}}'_W, \vec{\pi}'_{SS}, \\ (\vec{C}'_{M_i}, \vec{\hat{C}}'_{M_i})_{i=1}^{\ell}, \vec{\hat{C}}''_\Theta, \vec{\hat{C}}'_X, \vec{\pi}'_1, \vec{\pi}'_2, (\vec{\pi}'_{d_i}, \vec{\pi}'_{i,01}, \vec{\hat{\pi}}'_{i,01})_{i=1}^{\ell}, \vec{\pi}'_{r,1}, \vec{\pi}'_{r,2}, (\vec{\pi}'_{r,d_i})_{i=1}^{\ell} \Big).$$

- **$\mathbf{Ver_{pk}(CT)}$:** First, output 0 if pk or CT does not parse properly. Second, verify the validity of the LHSP signatures $\vec{\sigma}_2$ and $\vec{\sigma}_3$ with respect to the public key $\hat{\mathsf{opk}}$, and output 1 if it is invalid. Third, verify that the proofs $\vec{\hat{\pi}}_{sig}$, $\vec{\pi}_{SS}$, $\vec{\pi}_1$, $\vec{\pi}_2$, $(\vec{\pi}_{d_i}, \vec{\pi}_{i,01}, \vec{\hat{\pi}}_{i,01})_{i=1}^{\ell}$, $\vec{\pi}_{r,1}, \vec{\pi}_{r,2}$, $(\vec{\pi}_{r,d_i})_{i=1}^{\ell}$ are valid with regard to their corresponding equations. If at least one of these proofs is invalid, output 0; otherwise, output 1.

- **$\mathbf{Dec_{sk}(CT)}$:** Given the secret decryption key $\mathsf{sk} = (\alpha_i, \beta_i)_{i=1}^{\ell}$ and $\boldsymbol{c} = (c_0, c_1, c_2)$ included in CT, set m_i as $\log_{g_i}(d_i c_1^{-\alpha_i} c_2^{-\beta_i})$ and return $(m_1, \cdots, m_\ell)$.

9.5 Security proofs for our traceable encryption scheme

It is easy to see that our encryption scheme is correct, and its strong rerandomization property comes readily from that of the Groth-Sahai proofs. In this section, we provide a security proof for the verifiability, the TCCA security and the traceable property.

9.5.1 Verifiability

First, we recall that a traceable encryption scheme is verifiable if the adversary cannot forge a valid ciphertext which does not decrypt into a valid plaintext (See Definition 26; the corresponding experiment $\mathrm{Exp}^{\mathsf{ver}}$ is given in Fig. 33). With this in mind, we prove that our traceable encryption scheme has verifiability.

Theorem 12. *Under the SXDH assumption and in the ROM, for all $\ell \geq 1$, our traceable encryption scheme defined in Section 9.4 is verifiable with the plaintext space $\mathscr{P} = \{0,1\}^\ell$.*

$\mathrm{Exp}^{\mathsf{ver}}(\lambda, \mathbb{A})$

Requires: $\mathscr{P}$, the plaintext space

$\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;

$C \longleftarrow \mathbb{A}(\mathsf{pk}, \mathsf{sk})$;

if $\mathrm{Ver}_{\mathsf{pk}}(C) = 1 \wedge \mathrm{Dec}_{\mathsf{sk}}(C) \notin \mathscr{P}$ **then return** 1;

else return 0;

Figure 33: The verifiability experiment for a traceable encryption scheme

Proof. Let $\mathbb{A}$ be an adversary against the verifiability. We give a succession of games $H_0, \cdots, H_7$ where H_0 corresponds to the verifiability game. For each of those games, we denote S_i the probability that H_i outputs 1. As we rely on the SXDH assumption, the main proof strategy is to show that for each i, $|S_i - S_{i+1}|$ is proportional to the advantage of an explicit PPT adversary in the DDH game. By abuse of notation, we denote this advantage $\varepsilon_{\mathrm{sxdh}}$.

Game 1: In this game, we modify the generation of the Groth-Sahai CRS by picking an exponent $\nu \xleftarrow{\$} \mathbb{Z}_q$, $\vec{u} \xleftarrow{\$} \mathbb{G}^2$ and computing $\vec{u}_2 = \vec{u}_1^{\nu}$.

Now, we show that $|S_1 - S_0| \leq 2\varepsilon_{\mathrm{sxdh}}$. For this purpose, we construct an adversary $\mathbb{B}$ for the DDH game. It is given a tuple $z_1, z_2, z_3, z_4 \in \mathbb{G}$ which is either a DDH tuple or a random tuple. $\mathbb{B}$ interacts with $\mathbb{A}$ and simulates H_0, excepts that it sets $u_{1,1} = z_1$, $u_{1,2} = z_2$, $u_{2,1} = z_3$ and $u_{2,2} = z_4$. Clearly, if $\mathbb{B}$ is given a DDH tuple, it plays a perfect simulation of H_1 while, if it is given a random tuple, $\mathbb{B}$ plays a perfect simulation of H_0. Hence $\mathbb{B}$'s probability to win the DDH game is $1/2(S_1 + 1 - S_0)$, and $|S_1 - S_0| \leq 2\varepsilon_{\mathrm{sxdh}}$.

Game 2: In this game, we abort (by outputting a random bit) if $u_{1,1} = 1_{\mathbb{G}}$. Note that when we abort, we still have a probability of $1/2$ to output the correct value, so that $|S_2 - S_1| \leq \frac{1}{2q}$.

Game 3: In this game, we still pick $u_{1,1}$ as in Game 2, but pick a trapdoor $\tau_u \xleftarrow{\$} \mathbb{Z}_q$ and compute $u_{1,2} = u_{1,1}^{\tau_u}$. Since $u_{1,1} \neq 1_{\mathbb{G}}$, this does not change the distribution of the CRS so that $S_3 = S_2$.

Game 4: In this game, we similarly trapdoor the Groth-Sahai CRS $\vec{u}'_1, \vec{u}'_2$ and $\vec{\hat{v}}_1, \vec{\hat{v}}_2$ with random $\nu, \hat{\nu} \xleftarrow{\$} \mathbb{Z}_q$ so that $\vec{u}'_2 = \vec{u}'^{\nu}_1$ and $\vec{\hat{v}}_2 = \vec{\hat{v}}_1^{\hat{\nu}}$.

With the same argument as in Game 1 (applied twice), we have $|S_4 - S_3| \leq 4\varepsilon_{\mathrm{sxdh}}$.

Game 5: In this game, we similarly abort when $u'_{1,1} = 1_{\mathbb{G}}$ or when $\hat{v}_{1,1} = 1_{\hat{\mathbb{G}}}$ and compute $u'_{1,2} = u'^{\tau_{u'}}_{1,1}$ and $\hat{v}_{1,2} = \hat{v}_{1,2}^{\tau_{\hat{v}}}$ with $\tau_{u'}, \tau_{\hat{v}} \xleftarrow{\$} \mathbb{Z}_q$.

As above, we have $|S_5 - S_4| \leq \frac{1}{q}$.

Game 6: In this game, we abort (by outputting a random bit) when $S = 1_{\mathbb{G}}$ or $T = 1_{\mathbb{G}}$.

Clearly $|S_6 - S_5| \leq \frac{1}{q}$.

Game 7: In this game, we denote

$$\begin{aligned}\mathsf{CT} = \Big(&\vec{c}, \vec{C}_Z, \vec{C}_R, \vec{\sigma}_2, \vec{\sigma}_3, \vec{\hat{\pi}}_{sig}, \hat{\mathsf{opk}}, \vec{\hat{C}}_A, \vec{\hat{C}}_B,\\ &\vec{\hat{C}}_W, \vec{\pi}_{SS}, (\vec{C}_{M_i}, \vec{\hat{C}}_{M_i})_{i=1}^{\ell}, \vec{\hat{C}}_\Theta, \vec{\hat{C}}_X, \vec{\pi}_1, \vec{\pi}_2,\\ &(\vec{\pi}_{d_i}, \vec{\pi}_{01,i}, \vec{\hat{\pi}}_{01,i})_{i=1}^{\ell}, \vec{\pi}_{r,1}, \vec{\pi}_{r,2}, (\vec{\pi}_{r,d_i})_{i=1}^{\ell}\Big)\end{aligned}$$

the ciphertext sent by the adversary and we use the extractability of the Groth-Sahai proofs to extract the witnesses from the commitments, using the trapdoors. More precisely, we extract the witnesses $Z_1, R_1, \hat{a}, \hat{b}, \hat{w}, \hat{\Theta}, (M_i, \hat{M}_i)_i, \hat{x}$ such that

$$e(Z_1, \hat{g})e(R_1, \hat{h}) = e(g, \hat{l}_1)e(\prod_{i=1}^{\ell} d_i, \hat{l}_2)e(c_1, \hat{l}_3) \tag{10}$$

$$e(S, \hat{a})e(T, \hat{b}) = e(H, \hat{g}/\hat{w}) \tag{11}$$

$$e(c_1, \hat{w}) = e(g, \hat{\Theta}) \tag{12}$$

$$e(c_2, \hat{w}) = e(h, \hat{\Theta}) \tag{13}$$

$$\forall i, e(d_i, \hat{w}) = e(g_i, \hat{M}_i)e(f_i, \hat{\Theta}) \tag{14}$$

$$\forall i, e(g/M_i, \hat{M}_i) = 1 \tag{15}$$

$$\forall i, e(M_i, \hat{g}) = e(g, \hat{M}_i) \tag{16}$$

$$e(g, \hat{w}) = e(g, \hat{x}) \tag{17}$$

$$e(h, \hat{w}) = e(h, \hat{x}) \tag{18}$$

$$e(f_i, \hat{w}) = e(f_i, \hat{x}), \tag{19}$$

where $(\hat{l}_1, \hat{l}_2, \hat{l}_3) = \hat{\mathsf{opk}}$, $(F, G, H) = \mathrm{hash}(\hat{\mathsf{opk}})$ and $(g, \hat{g}, h, \hat{h}, S, T, (g_i, f_i)_{i=1}^{\ell})$ are parts of the public key. Afterwards, we abort (by outputting 0) whenever the following equations are not verified.

$$\hat{a} = \hat{b} = 1_{\hat{\mathbb{G}}} \textbf{ and } \hat{w} = \hat{g}. \tag{20}$$

Clearly, the output of Game 7 differs from Game 6 if and only if the adversary outputs a valid CT (in particular, with valid Groth-Sahai proofs which allow the extraction) such that Eq. (20) is not verified. We denote E this event and ε its probability, so that $|S_7 - S_6| \leq \varepsilon$. We construct an adversary $\mathbb{B}$ for the DDH game as follows.

First, $\mathbb{B}$ gets a challenge tuple S, T, U, V from the DDH game. If $S = 1_{\mathbb{G}}$ or $T = 1_{\mathbb{G}}$, which happens with a negligible probability μ_{st}, $\mathbb{B}$ outputs a random guess. Otherwise, it simulates Game 6 to $\mathbb{A}$ in the ROM.

For this purpose, $\mathbb{B}$ generates the public key honestly, except that it uses the S, T defined above. In addition, whenever $\mathbb{A}$ makes a new query to the random oracle with an input of the form $\hat{l}_1, \hat{l}_2, \hat{l}_3 \in \hat{\mathbb{G}}$ (by new, we mean that the same query was not made previously; otherwise $\mathbb{B}$ simply outputs the same answer as before), $\mathbb{B}$ generates $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$ and computes $H = S^\alpha T^\beta$. It then samples F and G at random, and uses F, G, H to answer the random oracle query. Eventually, $\mathbb{A}$ outputs some ciphertext CT. If it is invalid, E cannot occur and $\mathbb{B}$ outputs a random guess, hence wins with probability 1/2. If the ciphertext is valid, $\mathbb{B}$ can extract the witnesses as in Game 7 and check whether Eq. (20) is verified or not. If the equation is verified, $\mathbb{B}$ similarly

outputs a random guess. If not, $\mathbb{B}$ checks whether $e(U, \hat{a}(\hat{g}/\hat{w})^{-\alpha})e(V, \hat{b}(\hat{g}/\hat{w})^{-\beta}) = 1_{\mathbb{G}_T}$ and states that the challenge tuple is a DDH challenge if and only if this identity is verified.

Now, remark that $\mathbb{B}$'s simulation is perfectly indistinguishable from Game 6. In addition, if E occurs when S, T, U, V is a DDH tuple, $e(S, \hat{a})e(T, \hat{b}) = e(H, \hat{g}/\hat{w})$ by Eq. (11), so that $e(U, \hat{a}(\hat{g}/\hat{w})^{-\alpha})e(V, \hat{b}(\hat{g}/\hat{w})^{-\beta}) = 1_{\mathbb{G}_T}$ and $\mathbb{B}$ wins the DDH game. By contrast, if E occurs when S, T, U, V is a random tuple, then either $\hat{a} \neq 1_{\hat{\mathbb{G}}}$, $\hat{b} \neq 1_{\hat{\mathbb{G}}}$ or $\hat{w} \neq \hat{g}$. Since the adversary had no information on α, β except that $H = S^{\alpha}T^{\beta}$, the probability μ_1 that $\hat{a}(\hat{g}/\hat{w})^{-\alpha}$ and $\hat{b}(\hat{g}/\hat{w})^{-\beta}$ are simultaneously trivial is a most $1/q$. Hence, $e(U, \hat{a}(\hat{g}/\hat{w})^{-\alpha})e(V, \hat{b}(\hat{g}/\hat{w})^{-\beta})$ is uniformly random (the adversary had no information about U and V) and is equal to $1_{\mathbb{G}_T}$ with probability $1/q$. Therefore $\mathbb{B}$ wins the DDH game with probability $1 - \mu'$ with $|\mu'| \leq 2/q$.

Finally, remark that since $\mathbb{A}$ is not given any information about U, V, the event E is independent from the fact that S, T, U, V is a DDH tuple or not. Consequently, $\mathbb{B}$'s probability to win the DDH game is

$$\begin{aligned}
&\frac{1}{2}\mu_{st} + (1 - \mu_{st})\left(\frac{1}{2}\varepsilon(1 + 1 - \mu') + \frac{1}{2}(1 - \varepsilon)\right) \\
&= \frac{1}{2}\mu_{st} + \frac{1}{2}(1 - \mu_{st})\left(1 + \varepsilon - \mu'\right) \\
&= \frac{1}{2} + \frac{1}{2}(\varepsilon - \mu' - \mu_{st}(\varepsilon - \mu')).
\end{aligned}$$

Hence, $\varepsilon \leq 2\varepsilon_{\text{sxdh}} + \frac{4}{q}$ and is indeed negligible.

Game 8: In this game, we abort (by outputting a random bit) if $g = 1_{\mathbb{G}}$, $h = \mathbb{G}$ or $\hat{g} = 1_{\hat{\mathbb{G}}}$, and if there exists i such that $f_i = 1_{\mathbb{G}}$ or $g_i = 1_{\mathbb{G}}$.

Clearly, $|S_8 - S_7| \leq \frac{3+2\ell}{2q}$.

Conclusion. Now, we have $\hat{a} = \hat{b} = 1_{\hat{\mathbb{G}}}$ and $\hat{w} = \hat{g}$. Since $\neq 1_{\hat{\mathbb{G}}}$, there exists a unique $\theta \in \mathbb{Z}_q$ such that $\hat{\Theta} = \hat{g}^{\theta}$. Similarly, for all i, there exists a unique $m_i, \hat{m}_i \in \mathbb{Z}_q$ such that $M_i = g^{m_i}$ and $\hat{M}_i = \hat{g}^{\hat{m}_i}$.

By Eq. (16), $m_i = \hat{m}_i$ for all i and, by Eq. (15), $m_i \in \{0, 1\}$ for all i. Also, by Eq. (12), $c_1 = g^{\theta}$; by Eq. (13), $c_2 = h^{\theta}$ and, for all i, Eq. (14) gives $d_i = g_i^{m_i} f_i^{\theta}$. □

9.5.2 Traceability

We now show that our traceable encryption scheme is traceable as of Definition 28, which means that one cannot forge a ciphertext $\mathsf{CT}^{\star}$ that has the same trace as a given ciphertext CT, but does not encrypt the same plaintext. To ease readability, we reproduce the traceability experiment in Fig. 34.

Theorem 13. *Under the SXDH assumption and in the ROM, for all $\ell \geq 1$, our traceable encryption scheme defined in Section 9.4 is traceable.*

Proof. Let $\mathbb{A}$ be an adversary against the traceability. Just as in the proof of verifiability, we give a succession of games $H_0, \cdots, H_3$ where H_0 corresponds to the traceability game. For each game, we denote S_i the probability that H_i outputs 1. As we rely on the SXDH assumption, the main strategy is to show that for each i, $|S_i - S_{i+1}|$ is proportional to the advantage of an explicit adversary in the DDH game. By abuse of notation, we denote this advantage $\varepsilon_{\text{sxdh}}$.

$\mathrm{Exp}^{\mathsf{trace}}(\lambda, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;
$m \longleftarrow \mathbb{A}(\mathsf{pk}, \mathsf{sk})$;
$r \xleftarrow{\$} \mathscr{R}$;
$C \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(m, r)$;
$C^\star \longleftarrow \mathbb{A}(C)$;
if $m \in \mathscr{P}$ **and** $\mathrm{Ver}_{\mathsf{pk}}(C^\star) = 1$ **and** $\mathrm{Trace}_{\mathsf{pk}}(C) = \mathrm{Trace}_{\mathsf{pk}}(C^\star)$ **and** $\mathrm{Dec}_{\mathsf{sk}}(C^\star) \neq m$ **then return** 1 **else return** 0;

Figure 34: Traceability experiment for a traceable encryption scheme

Game 1: We begin with the same transitions as in the proof of Theorem 12. Namely, we define Game 0 as the traceability experiment, and Game 1 is a modified game where the Groth-Sahai CRS are generated in the extractability mode and where we added some conditions to abort. With the same arguments, $|S_1 - S_0|$ is negligible under the SXDH assumption and in the ROM.

Game 2: In this game, we generate a LHSP keypair $\hat{\mathsf{opk}}, \mathsf{osk}$ at random, before giving pk, sk to the adversary. Also, we generate a random $\alpha \in \mathbb{Z}_q$, compute $F = \prod_{i=1}^{\ell} f_i^{\alpha}$ and $G = g^{\alpha}$ and we pick $H \xleftarrow{\$} \mathbb{G}$. Then, whenever the adversary queries the random oracle with the input $\hat{\mathsf{opk}}$, we answer with F, G, H. Also, when the adversary gives the plaintext m to encrypt, we use the link key osk instead of a random link key.

To argue that $|S_2 - S_1|$ is negligible, we construct an adversary $\mathbb{B}$ for the DDH game that interacts with $\mathbb{A}$ by simulating Game 1. $\mathbb{B}$ gets a challenge tuple g_1, g_2, g_3, g_4 from the DDH game and uses g_1 as g in the simulation. In addition, instead of generating the f_i's honestly, $\mathbb{B}$ generates $f_1, \cdots, f_{\ell-1}$ at random and sets f_ℓ such that $\prod_{i=1}^{\ell} f_i = g_2$. The remaining of the simulation is as in Game 2, except that $\mathbb{B}$ sets G as g_3 and F as g_4. Finally, to decide whether $\mathbb{A}$ wins or loses the simulation, $\mathbb{B}$ checks that $\mathsf{CT}^\star$ is valid and has the same trace as CT. If so, $\mathbb{B}$ extracts the plaintext $m_1^\star, \cdots, m_\ell^\star$ from the Groth-Sahai commitments of $\mathsf{CT}^\star$ (this is possible, thanks to the same argument as in the verifiability game), and tests whether $m^\star = m$. This way, $\mathbb{B}$ can decide whether $\mathbb{A}$ wins the simulated game. If $\mathbb{A}$ wins the simulation, $\mathbb{B}$ states that the challenge was a DDH tuple; otherwise, it states that it was a random tuple.

Remark that when the challenge is a random tuple, $\mathbb{B}$ plays a perfect simulation of Game 1 to $\mathbb{A}$ and hence wins with probability $1 - S_1$. Otherwise, $\mathbb{B}$ plays a perfect simulation of Game 2 and wins with probability S_2. Therefore, $\mathbb{B}$'s advantage in the DDH game is $\frac{1}{2}|S_2 - S_1|$, so that $|S_2 - S_1| \leq 2\varepsilon_{\mathrm{sxdh}}$.

Game 3: In this game, we use the same arguments as in the verifiability game to extract a plaintext $m'_1, \cdots, m'_\ell \in \{0,1\}^\ell$ from $\mathsf{CT}^\star$ (this is only possible if the latter is a valid ciphertext). If $m'_1, \cdots, m'_\ell \neq m_1, \cdots, m_\ell$ but $\prod_{i=1}^{\ell} g_i^{m'_i} = \prod_{i=1}^{\ell} g_i^{m_i}$, we abort by outputting 0.

The output of this game differs from that of Game 2 if and only if the adversary outputs a valid ciphertext $\mathsf{CT}^\star$ such that $\mathrm{Trace}_{\mathsf{pk}}(\mathsf{CT}^\star) = \mathrm{Trace}_{\mathsf{pk}}(\mathsf{CT})$, $\mathrm{Dec}_{\mathsf{sk}}(\mathsf{CT}) \neq \mathrm{Dec}_{\mathsf{sk}}(\mathsf{CT}^\star)$ and $\prod_{i=1}^{\ell} g_i^{m'_i} = \prod_{i=1}^{\ell} g_i^{m_i}$, where $m_1, \cdots, m_\ell = \mathrm{Dec}_{\mathsf{sk}}(\mathsf{CT})$ and $m'_1, \cdots, m'_\ell = \mathrm{Dec}_{\mathsf{sk}}(\mathsf{CT}^\star)$. We denote E this event and ε its probability, so that $|S_3 - S_2| \leq \varepsilon$.

We construct and adversary $\mathbb{B}$ against the DDH game as follows. First, $\mathbb{B}$ gets a challenge

tuple g_1, g_2, g_3, g_4. If $g_1 = 1$ or $g_2 = 1$, which happens with a negligible probability $\mu \leq 2/q$, $\mathbb{B}$ can trivially guess whether the challenge tuple is a DDH tuple or not, and hence wins the DDH game. Otherwise, $\mathbb{B}$ interacts with $\mathbb{A}$ by simulating Game 2; however, instead of generating the g_i's at random, for all i, it picks two random $\alpha_i, \beta_i \in \mathbb{Z}_q$ and computes $g_i = g_1^{\alpha_i} g_2^{\beta_i}$, so that $\mathbb{B}$'s simulation is perfectly indistinguishable from Game 2.

Remark that since $\mathbb{B}$ has the secret key sk, it can efficiently detect when E occurs and extract the corresponding $m_1, m'_1, \cdots, m_\ell, m'_\ell \in \{0,1\}$. Yet, when E occurs, we have

$$\prod_{i=1}^{\ell} g_i^{m_i} = \prod_{i=1}^{\ell} \ell g_i^{m'_i}$$
$$\prod_{i=1}^{\ell} g_1^{(m_i - m'_i)\alpha_i} = \prod_{i=1}^{\ell} g_2^{(m'_i - m_i)\beta_i},$$

so that $g_1 = g_2^s$, with $s = \frac{\sum_{i=1}^{\ell}(m'_i - m_i)\beta_i}{\sum_{i=1}^{\ell}(m_i - m'_i)\alpha_i}$. By computing s, and checking whether $g_3 = g_4^s$, $\mathbb{B}$ can decide whether g_1, g_2, g_3, g_4 is a DDH tuple or not, and thus win the DDH game. To sum up, $\mathbb{B}$ check whether E occurs, compute s and decide whether the challenge is a DDH tuple or not. If E does not occur or if E occurs but with $\sum_{i=1}^{\ell}(m_i - m'_i)\alpha_i = 0$ (in which case $\mathbb{B}$ cannot compute s), $\mathbb{B}$ outputs a random guess.

Now, note that since the adversary has no information about $(\alpha_i, \beta_i)_i$, when E occurs, the event $\sum_{i=1}^{\ell}(m_i - m'_i)\alpha_i = 0$ occurs with probability $1/q$. Hence, $\mathbb{B}$'s probability to win the DDH game is

$$\frac{1}{2}(1-\varepsilon) + \varepsilon\left(\frac{1}{2q} + (1 - 1/q)\right) = \frac{1}{2} + \frac{1}{2}\varepsilon - \frac{1}{2q}\varepsilon.$$

Therefore, $\varepsilon \leq 2\varepsilon_{\text{sxdh}} + \frac{1}{q}$ and is indeed negligible.

Conclusion. We conclude by giving a reduction to the lin-unf experiment defined in Definition 30. We refer to this experiment as the unforgeability game, and we construct an adversary $\mathbb{B}$ for the unforgeability game which interacts with $\mathbb{A}$ by simulating H_3. First, $\mathbb{B}$ gets $\hat{\mathsf{opk}}$ from the unforgeability game and uses this $\hat{\mathsf{opk}}$ to simulate H_3. However, $\mathbb{B}$ does not have the secret key osk; therefore, when $\mathbb{A}$ sends $m_1, \cdots, m_\ell \in \{0,1\}$, $\mathbb{B}$ queries the LHSP signatures of $(g, \prod_{i=1}^{\ell} d_i, c_1)$ and $(1, \prod_{i=1}^{\ell} f_i, g)$ using the signing oracle. Thanks to the trapdoor α introduced in Game 2, $\mathbb{B}$ can deduce a valid LHSP signature for $(1, F, G)$.

Recall that $\mathbb{A}$ wins H_3 with probability S_3, which means that it outputs a valid ciphertext $\mathsf{CT}^\star$ such that $\text{Trace}_{\mathsf{pk}}(\mathsf{CT}^\star) = \text{Trace}_{\mathsf{pk}}(\mathsf{CT}) = \hat{\mathsf{opk}}$. Since this ciphertext is valid, $\mathbb{B}$ can extract the signature Z_1, R_1 and the plaintext $m'_1, \cdots, m'_\ell \in \{0,1\}$ from the Groth-Sahai proofs. In particular, (Z_1, R_1) is a valid signature of $(g, \prod_{i=1}^{\ell} d_i^\star, c_1^\star)$. Yet, with the same arguments as in the verifiability, we know that there exists $\theta^\star \in \mathbb{Z}_q$ such that $c_1^\star = g^{\theta^\star}$ and $d_i^\star = g_i^{m'_i} f_i^{\theta^\star}$ for all i. Also, since CT was honestly generated, there exists $\theta \in \mathbb{Z}_q$ such that $c_1 = g^\theta$ and $d_i = g_i^{m_i} f_i^\theta$ for all i. Hence, the message $(g, \prod_{i=1}^{\ell} d_i^\star, c_1^\star)$ is linearly independent from the messages signed by the signing oracle if and only if the matrix (21) has rank 3. By Gaussian elimination, we can transform this matrix into matrix (22) then matrix (23); the latter clearly has rank 3.

Hence, whenever $\mathbb{A}$ wins the traceability game, $\mathbb{B}$ wins the unforgeability game. Since we showed in Lemma 16 that the LHSP scheme is unforgeable under the SXDH assumption and in the ROM, this shows that S_3 is indeed negligible. □

$$\begin{pmatrix} g & \prod_{i=1}^{\ell} g_i^{m_i} f_i^{\theta} & g^{\theta} \\ g & \prod_{i=1}^{\ell} g_i^{m'_i} f_i^{\theta^\star} & g^{\theta^\star} \\ 1 & \prod_{i=1}^{\ell} f_i & g \end{pmatrix} \quad (21) \qquad \begin{pmatrix} g & \prod_{i=1}^{\ell} g_i^{m_i} & 1 \\ g & \prod_{i=1}^{\ell} g_i^{m'_i} & 1 \\ 1 & \prod_{i=1}^{\ell} f_i & g \end{pmatrix} \quad (22) \qquad \begin{pmatrix} g & \prod_{i=1}^{\ell} g_i^{m_i} & 1 \\ 1 & \prod_{i=1}^{\ell} g_i^{m'_i - m_i} & 1 \\ 1 & \prod_{i=1}^{\ell} f_i & g \end{pmatrix} \quad (23)$$

$\mathrm{Exp}^{\mathsf{tcca}}(\lambda, \mathbb{A})$

$\mathsf{pk}, \mathsf{sk} \longleftarrow \mathrm{Gen}(\lambda)$;
$C_0, C_1 \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathrm{Dec}}}(\mathsf{pk})$;
$b \xleftarrow{\$} \{0,1\}$;
if $\mathrm{Trace}_{\mathsf{pk}}(C_0) \neq \mathrm{Trace}_{\mathsf{pk}}(C_1)$ **or** $\mathrm{Ver}_{\mathsf{pk}}(C_0) = 0$ **or** $\mathrm{Ver}_{\mathsf{pk}}(C_1) = 0$ **then return** b;
$r \xleftarrow{\$} \mathscr{R}$;
$C^\star \longleftarrow \mathrm{Rand}_{\mathsf{pk}}(C_b, r)$;
$b' \longleftarrow \mathbb{A}^{\mathcal{O}_{\mathrm{Dec}^\star}}(C^\star)$;
if $b = b'$ **then return** 1 **else return** 0;

Figure 35: TCCA-experiment for a traceable encryption scheme.

9.5.3 TCCA security

Finally, we prove that our traceable encryption scheme provides TCCA-security as of Definition 27, which means that the rerandomizations of two given ciphertexts that have the same trace are indistinguishable. To ease readability, we reproduce the TCCA experiment in Fig. 35.

Theorem 14. *Under the SXDH assumption and in the ROM, for all $\ell \geq 1$, our traceable encryption scheme defined in Section 9.4 is TCCA-secure.*

Proof. Let $\mathbb{A}$ be an adversary against the TCCA security. As usual, we give a succession of games $H_0, \cdots, H_3$ where H_0 corresponds to the TCCA game. For each of those games, we denote S_i the probability that H_i outputs 1. As we rely on the SXDH assumption, the main proof strategy is to show that for each i, $|S_i - S_{i+1}|$ is proportional to the advantage of an explicit PPT adversary in the DDH game. By abuse of notation, we denote this advantage $\varepsilon_{\mathrm{sxdh}}$.

Game 1: In this game, we abort (by outputting a random bit) when $S = 1_{\mathbb{G}}$ or $T = 1_{\mathbb{G}}$. Clearly, $|S_1 - S_0|$ is negligible.

Game 2: In this game, we trapdoor the random oracle answers: when the adversary makes a hash query with some input, if the same query was made previously we answer with the same output as before; otherwise we pick $F, G \xleftarrow{\$} \mathbb{G}$ as in game H_0, and pick $\tau_x, \tau'_x \xleftarrow{\$} \mathbb{Z}_q$, compute $H = S^{\tau_x} T^{\tau'_x}$, keep in memory the association between H and τ_x, τ'_x and output F, G, H.

Note that since $(S, T) \neq (1_{\mathbb{G}}, 1_{\mathbb{G}})$, the distribution of the output is not changed, so that $S_2 = S_1$.

Game 3: In this game, we change the way we generate $CT^\star$, and more precisely the Groth-Sahai proofs and their commitments. The commitments C_Z, C_R and the proof $\hat{\pi}_{sig}$ are processed as usual; however, for the other commitment and proofs, we do not follow the Rand procedure. Instead, we first generate some witnesses $\hat{a}, \hat{b}, \hat{w}, \hat{\Theta}, \hat{M}_i, M_i, \hat{x}$ that verify the equations (10) to (19). Since most of the equations are linear, we can set $\hat{w}, \hat{\Theta}, \hat{M}_i, M_i, \hat{x}$ as trivial elements. Finally, to

obtain $\hat{a}$ and $\hat{b}$ such that $e(S,\hat{a})e(T,\hat{b}) = e(H,\hat{g})$ (recall that we set $\hat{w}$ to the trivial element $1_{\hat{\mathbb{G}}}$), we use the trapdoors τ_x, τ'_x such that $H = S^{\tau_x}T^{\tau'_x}$, so that $e(H,\hat{g}) = e(S,\hat{g}^{\tau_x})e(T,\hat{g}^{\tau'_x})$. Namely, we set $\hat{a} = \hat{g}^{\tau_x}$ and $\hat{b} = \hat{g}^{\tau'_x}$.

Once those witnesses have been generated, we generate fresh random commitments and Groth-Sahai proofs using them, and use the corresponding commitments and proofs instead of the rerandomized ones.

By the perfect rerandomization and the witness indistinguishability of the Groth-Sahai proofs, unless there is a DDH tuple in the Groth-Sahai CRS, this game is perfectly indistinguishable from the previous one. Hence, $|S_3 - S_2| \leq \frac{3}{q}$.

Game 4: In this game, we abort (by outputting b) if the two ciphertexts CT_0 and CT_1 given by the adversary use different LHSP signatures, *i.e.* if $\sigma_2^0 \neq \sigma_2^1$ or $\sigma_3^0 \neq \sigma_3^1$.

When Game 4 differs from Game 3, $\mathrm{Trace}_{pk}(\mathsf{CT}_0) = \mathrm{Trace}_{pk}(\mathsf{CT}_1)$, hence the group elements F, G, H derived from this common public key are the same. Therefore σ_2^0 and σ_2^1 (resp. σ_3^0 and σ_3^1) are valid LHSP signatures of the same message $(1, \prod_{i=1}^{\ell} f_i, g)$ (resp. $(1, F, G)$). Thus, if $\sigma_2^0 \neq \sigma_2^1$ or $\sigma_3^0 \neq \sigma_3^1$, we have a straightforward reduction to SXDH. (We refer to the proof of Lemma 16 where we gave an explicit reduction from finding two different LHSP signatures on the same message and winning the DDH game.)

Game 5: In this game, we change the generation of the Groth-Sahai CRS crs: instead of picking them at random, we pick them as random DDH tuple, and we keep the corresponding trapdoor. Also, we abort by outputting a random bit if one of the element of the CRS is a trivial element. (Note that the second CRS crs', which is used for $\vec{\hat{\pi}}_{sig}$ only, is still generated honestly.)

Using a similar argument as in Games Game 3 to Game 5 from the proof of verifiability, it is easy to show that $|S_5 - S_4|$ is negligible under the SXDH assumption.

Game 6: In this game, whenever the adversary outputs a valid ciphertext (including during a decryption query), we extract $Z_1, R_1, \hat{a}, \hat{b}, \hat{w}, \hat{\Theta}, \hat{M}_i, M_i, \hat{x}$ from the commitments and abort (by outputing a random bit) if Eq. (20) is not verified.

Using a similar argument as in the proof of verifiability (see Game 7), we have that $|S_6 - S_5|$, is negligible.

Game 7: In this game, whenever the adversary $\mathbb{A}$ makes a new random oracle query, we picks a random $\tau \in \mathbb{Z}_q$, compute $F = \prod_{i=1}^{\ell} f_i^{\tau}$ and $G = g^{\tau}$.

To argue that $|S_7 - S_6|$ is negligible, we construct a succession of games $(H_i)_i$. The game H_i is Game 7, except that the first i queries to the random oracle are answered as in Game 6 (*i.e.* using uniformly random F, G and the trapdoored H). With the same idea as in the proof of traceability (see Game 2), it is easy to show that all the conditions of the hybrid lemma are verified, so that $|S_7 - S_6|$ is negligible.

Game 8: In this game, we denote $\mathsf{sk} = (\alpha_i, \beta_i)_{i=1}^{\ell}$ the secret key, and we abort (by outputting a random bit) if $h^{\sum_{i=1}^{\ell}\beta_i} g^{\sum_{i=1}^{\ell}\alpha_i} = 1$. Clearly $|S_8 - S_7|$ is negligible.

Game 9: In this game, we generate $g, h, \tilde{G}, \tilde{H} \in \mathbb{G}\backslash\{1\}$ at random and generate the secret key $(\alpha_i, \beta_i)_{i=1}^{\ell}$ such that $h^{\sum_{i=1}^{\ell}\beta_i} g^{\sum_{i=1}^{\ell}\alpha_i} \neq 1$. Also, we denote $\tilde{F} = \tilde{H}^{\sum_{i=1}^{\ell}\beta_i}\tilde{G}^{\sum_{i=1}^{\ell}\alpha_i}$.

Then, whenever the adversary makes a query to the random oracle with a new input $\hat{\mathsf{opk}}$, we do not compute $F = \prod_{i=1}^{\ell} f_i^{\tau}$ and $G = g^{\tau}$ for some random τ. Instead, we compute $F = \tilde{F}^{\tau}$

and $G = \tilde{G}^{\tau}$; as for H, it is generated as usual (*i.e.* by generating two random $\tau_H, \tau'_H \in \mathbb{Z}_q$ and computing $H = S^{\tau_H} T^{\tau'_H}$). We answer to the random oracle query with F, G, H.

When $\mathbb{A}$ outputs the ciphertexts CT_0 and CT_1, we pick a random bit $b \in \{0,1\}$. However, instead of rerandomizing CT_b as in Game 8, we parse it as

$$\begin{aligned}\mathsf{CT}_b = \Big(& c_1^b, c_2^b, (d_i^b)_{i=1}^{\ell}, \vec{C}_Z, \vec{C}_R, \vec{\sigma}_2, \vec{\sigma}_3, \vec{\tilde{\pi}}_{sig}, \hat{\mathsf{opk}}, \vec{\hat{C}}_A, \vec{\hat{C}}_B, \\ & \vec{\hat{C}}_W, \vec{\pi}_{SS}, (\vec{C}_{M_i}, \vec{\hat{C}}_{M_i})_{i=1}^{\ell}, \vec{\hat{C}}_{\Theta}, \vec{\hat{C}}_X, \vec{\pi}_1, \vec{\pi}_2, \\ & (\vec{\pi}_{d_i}, \vec{\pi}_{01,i}, \vec{\tilde{\pi}}_{01,i})_{i=1}^{\ell}, \vec{\pi}_{r,1}, \vec{\pi}_{r,2}, (\vec{\pi}_{r,d_i})_{i=1}^{\ell} \Big)\end{aligned}$$

and compute $\mathsf{CT}^{\star}$ as follows. First, we recover (F, G, H) and the corresponding $\tau \in \mathbb{Z}_q$ from $\mathrm{Trace}_{pk}(\mathsf{CT}_b)$, compute $\bar{H} = \tilde{H}^{\tau}$, pick $\mu_1, \mu_2 \xleftarrow{\$} \mathbb{Z}_q$ and compute

$$c_1^{\star} = c_1^b g^{\mu_1} G^{\mu_2},\ c_2^{\star} = c_2^b h^{\mu_1} \bar{H}^{\mu_2},\ \text{and, for all } i,\ d_i^{\star} = d_i^b f_i^{\mu_1} F_i^{\mu_2},$$

where $F_i = G^{\alpha_i} \bar{H}^{\beta_i}$.

To produce $\vec{C}_Z^{\star}$, $\vec{C}_R^{\star}$ and $\vec{\tilde{\pi}}_{sig}^{\star}$, we extract (Z_1, R_1) from the Groth-Sahai commitments $\vec{C}_Z^b$ and $\vec{C}_R^b$, parses $\vec{\sigma}_2$ as (Z_2, R_2), $\vec{\sigma}_3$ as (Z_3, R_3) and compute the witnesses $Z = Z_1 Z_2^{\mu_1} Z_3^{\mu_2}$ and $R = R_1 R_2^{\mu_1} R_3^{\mu_2}$. (Recall that $\vec{\sigma}_2$ and $\vec{\sigma}_3$ are the same for both ciphertexts CT_0 and CT_1.) Since

$$\prod_{i=1}^{\ell} F_i = G^{\sum_{i=1}^{\ell} \alpha_i} \bar{H}^{\sum_{i=1}^{\ell} \beta_i} = \tilde{F}^{\tau} = F,$$

the vector $(g, \prod_{i=1}^{\ell} d_i^{\star}, c_1^{\star})$ is a linear combination of $(g, \prod_{i=1}^{\ell} d_i^b, c_1^b)$, $(1, \prod_{i=1}^{\ell} f_i, g)$ and $(1, F, G)$, with the coefficients 1, μ_1 and μ_2. Therefore, by the linearity of the LHSP signature, (Z, R) is a valid signature of $(g, \prod_{i=1}^{\ell} d_i^{\star}, c_1^{\star})$. Using these witnesses, we generate $\vec{C}_Z^{\star}$ and $\vec{C}_R^{\star}$ using the Groth-Sahai commitment scheme, and $\vec{\tilde{\pi}}_{sig}^{\star}$ using the Groth-Sahai proving algorithm.

Finally, we generate the remaining of $\mathsf{CT}^{\star}$ as in Game 8, using the trapdoors τ_H and τ'_H such that $H = S^{\tau_H} T^{\tau'_H}$.

To argue that Game 9 is indistinguishable from Game 8, we show that both games are indistinguishable when $g, h, \tilde{G}, \tilde{H}$ happens to be a DDH tuple.

Indeed, consider that $g, h, \tilde{G}, \tilde{H}$ is a DDH tuple, and that none of its component is trivial. Then there exists $\gamma \in \mathbb{Z}_q^{\times}$ such that $\tilde{G} = g^{\gamma}$ and $\tilde{H} = h^{\gamma}$. This way, when we answer to a random oracle query with $F = \tilde{F}^{\tau}$ and $G = \tilde{G}^{\tau}$ for some random τ, we have

$$F = \tilde{F}^{\tau} = \left(\tilde{H}^{\sum_{i=1}^{\ell} \beta_i} \tilde{G}^{\sum_{i=1}^{\ell} \alpha_i} \right)^{\tau} = \left(h^{\sum_{i=1}^{\ell} \beta_i} g^{\sum_{i=1}^{\ell} \alpha_i} \right)^{\tau\gamma} = \prod_{i=1}^{\ell} f_i^{\tau\gamma}$$

and $G = g^{\tau\gamma}$. Hence (F, G, H) follows the same distribution as in Game 8.

In addition, we have

$$\begin{aligned} c_1^{\star} &= c_1^b g^{\mu_1} G^{\mu_2} = c_1^b g^{\mu_1} \tilde{G}^{\tau\mu_2} = c_1^b g^{\mu_1 + \gamma\tau\mu_2} \\ c_2^{\star} &= c_2^b h^{\mu_1} \bar{H}^{\mu_2} = c_2^b h^{\mu_1} \tilde{H}^{\tau\mu_2} = c_2^b h^{\mu_1 + \gamma\tau\mu_2} \\ d_i^{\star} &= d_i^b f_i^{\mu_1} F_i^{\mu_2} = d_i^b f_i^{\mu_1} \left(G^{\alpha_i} \bar{H}^{\beta_i} \right)^{\mu_2} = d_i^b f_i^{\mu_1 + \gamma\tau\mu_2}. \end{aligned}$$

Finally, it remains to argue that the vector (Z_2, R_2) and (Z_3, R_3) are collinear, except with a negligible probability. Indeed, when they are collinear, (Z, R) follows the same distribution as in Game 8, and since the Groth-Sahai proofs and commitments are computed from (Z, R) in the same way, they also follow the same distribution as in Game 8.

Now, let p be the probability that (Z_2, R_2) and (Z_3, R_3) are not collinear when $g, h, \tilde{G}, \tilde{H}$ is a DDH tuple. We construct an adversary $\mathbb{B}$ against DDH which interacts with $\mathbb{A}$ by simulating Game 9. However, it gets $\hat{g}, \hat{h}, \hat{g}', \hat{h}'$ as a tuple challenge from the DDH game, picks a random $\gamma \in \mathbb{Z}_q^\times$ and sets $\tilde{G} = g^\gamma$ and $\tilde{H} = h^\gamma$. At some point, $\mathbb{A}$ outputs the two ciphertexts from which $\mathbb{B}$ deduces (Z_2, R_2) and (Z_3, R_3). Since they are valid signatures, we have

$$e(Z_2, \hat{g})e(R_2, \hat{h}) = e(\prod_{i=1}^{\ell} f_i, \hat{l}_2)e(g, \hat{l}_3)$$

$$e(Z_3, \hat{g})e(R_3, \hat{h}) = e(F, \hat{l}_2)e(G, \hat{l}_3) = e(\prod_{i=1}^{\ell} f_i^{\tau\gamma}, \hat{l}_2)e(g^{\tau\gamma}, \hat{l}_3).$$

Hence, we have $e(Z_3 Z_2^{-\tau\gamma}, \hat{g})e(R_3 R_2^{-\tau\gamma}, \hat{h}) = 1$.

Therefore, $\mathbb{B}$ can proceed as follows. It computes $e(Z_3 Z_2^{-\tau\gamma}, \hat{g}')e(R_3 R_2^{-\tau\gamma}, \hat{h}')$ and, if this is equal to 1, states that $\hat{g}, \hat{h}, \hat{g}', \hat{h}'$ was a DDH tuple; otherwise, it states that it was a random tuple. Remark that when $\hat{g}, \hat{h}, \hat{g}', \hat{h}'$ is a DDH tuple, the corresponding equality always hold so that $\mathbb{B}$ wins with probability 1. If $\hat{g}, \hat{h}, \hat{g}', \hat{h}'$ is a random tuple, then, there are two possibilities: either $Z_3 Z_2^{-\tau\gamma} = R_3 R_2^{-\tau\gamma} = 1$ so that the equality also holds in any case; either this is not the case and the equality hold with probability $1/q$ (this is because the adversary had no information about $\hat{g}', \hat{h}'$, which is uniformly random). Let p' be the probability that $Z_3 Z_2^{-\tau\gamma} = R_3 R_2^{-\tau\gamma} = 1$ does not hold.

Then, the probability that $\mathbb{B}$ wins the DDH game is $\frac{1}{2}(1 + p'(1 - 1/q))$, so that $p \leq p' \leq 2\varepsilon_{\text{sxdh}} + \frac{1}{q}$. Indeed, if (Z_2, R_2) and (Z_3, R_3) are not collinear, the equalities $Z_3 Z_2^{-\tau\gamma} = R_3 R_2^{-\tau\gamma} = 1$ are not possible so that $p \leq p'$.

Conclusion. To conclude the proof, we show that the view of the adversary when $b = 0$ is statistically indistinguishable from that of the adversary when $b = 1$. For this purpose, recall that thanks to the transition to Game 6, the only decryption queries which are accepted contains a ciphertext CT for which the CPA part is of the form $(g^\rho, h^\rho, g_1^{m_1} f_1^\rho, \cdots, g_\ell^{m_\ell} f_\ell^\rho)$, for some $\rho \in \mathbb{Z}_q$. Consequently, the equations $g_i^{m_i} = d_i c_1^{-\alpha_i} c_2^{-\beta_i}$ that can be deduced by the answer of the oracle queries are collinear with the equations $f_i = g^{\alpha_i} h^{\beta_i}$ from the public key.

In addition, when we answer to a new random oracle query, we compute H independently and $F = \tilde{F}^\tau$ and $G = \tilde{G}^\tau$ with a fresh random τ. Therefore, a random query unrelated to opk does not give any more information than $\tilde{F}, \tilde{G}$.

Also, since the commitments $\vec{C}_Z^\star, \vec{C}_R^\star$ and the Groth-Sahai proof $\vec{\pi}_{sig}$ use another independent CRS crs' which is not trapdoored, unless this CRS contains a DDH tuple, they are perfectly witness indistinguishable, which means that they reveal no information about b.

Besides, since the other Groth-Sahai commitments and proofs are computed independently of CT_0, CT_1 and b, they similarly reveal no information about b.

Overall, a computationally unbounded adversary has access to the information contained in the following equations:

$$\tilde{H}^{\sum_{i=1}^{\ell}\beta_i}\tilde{G}^{\sum_{i=1}^{\ell}\alpha_i} = \tilde{F} \tag{24}$$

$$\forall i, g^{\alpha_i}h^{\beta_i} = f_i \tag{25}$$

$$c_1^b g^{\mu_1}\tilde{G}^{\tau\mu_2} = c_1^\star \tag{26}$$

$$c_2^b h^{\mu_1}\tilde{H}^{\tau\mu_2} = c_2^\star \tag{27}$$

$$\forall i, d_i^b f_i^{\mu_1}\left(g^{\tau\alpha_i}\tilde{H}^{\beta_i}\right)^{\mu_2} = d_i^\star. \tag{28}$$

Yet, unless $\tau = 0$ or $g, h, \tilde{G}, \tilde{H}$ is a DDH tuple, the vectors (g, h) and $(\tilde{G}^\tau, \tilde{H}^\tau)$ are linearly independent so that there exists a unique μ_1^1, μ_2^1 (resp. μ_1^0, μ_2^0) such that Eq. (26) and (27) are verified with $b = 1$ (resp. $b = 0$).

Also, remark that unless $g = 1_{\mathbb{G}}$ or $\tilde{H} = 1_{\mathbb{G}}$ or $\mu_2^1 = 0$ (resp. $\mu_2^0 = 0$), the 2ℓ equations defined in Eq. (25) and 28 are linear in the secret key, and linearly independent. Therefore, there exists a unique secret key $(\alpha_i^1, \beta_i^1)_{i=1}^{\ell}$ (resp. $(\alpha_i^1, \beta_i^1)_{i=1}^{\ell}$ that verifies them).

Finally, unless $\sum_{i=1}^{\ell}\beta_i = 0$, Eq. 24 is equivalent to $\tilde{H} = \left(\tilde{F}\tilde{G}^{-\sum_{i=1}^{\ell}\alpha_i}\right)^{\left(\sum_{i=1}^{\ell}\beta_i\right)^{-1}}$.

Therefore, except with probability at most $7/q$, the probability that a specific $\mathsf{CT}^\star$ is output to the adversary is the same when $b = 0$ as when $b = 1$. Hence, $\mathsf{CT}^\star$ is independent from b and the probability that the adversary wins is exactly $1/2$. Overall, this shows that the advantage of $\mathbb{A}$ in Game 9 is negligible as required.

□

9.6 Application to verifiable receipt-free electronic voting

Until now, we presented a new definition of receipt-freeness as well as the notion of traceable encryption, which is adapted to the rerandomization paradigm. In this section, we propose TREnc, a Helios-like voting scheme that allows to obtain receipt-freeness with very little assumptions on the protocol. This contribution is similar to that of [DPP22b], except that the setup is public coin and that we support 0/1 proofs.

9.6.1 A voting scheme based on a traceable encryption

To define our voting scheme, we give the algorithms ($\mathsf{Setup}, \mathsf{Vote}, \mathsf{Check}$, Valid, Tally, Verify). Recall that our construction is independent from the registration, so that we do not explicit the registration protocol nor how the eligibility is enforced. Just as for Helios, we propose a voting system that can be augmented with any mechanism to obtain eligibility. One of the specificities of our protocol is that the Vote protocol is interactive, and requires two messages from the voter. This is because we use an interactive Σ-protocol to prove the knowledge of the link key. This is needed in the strategy that we propose to deceive the vote buyer, since the voter can use this interactive protocol to extract the link key. (Note that the voter does not interact with the vote buyer, but rather with the *instructions* that are given, seen as a Turing machine.)

For the tally part, we use the fact that our traceable encryption has a somewhat homomorphic property, which means that a homomorphic strategy can be applied. Alternatively, a decryption mixnet can also be used. In our construction, the universal verifiability comes from the computational soundness of the threshold decryption protocol. Also, assuming that the voting device is honest, the individual verifiability comes from the traceability of the encryption scheme and the

Table 22: Trust assumptions in TREnc.

	Indiv. verif.	Univ. verif.	Privacy	Receipt-freeness
Server	not trusted	not trusted	not trusted	trusted
Talliers	not trusted	not trusted	t out of n_T	t out of n_T
Device	trusted	not trusted	trusted	trusted

zero knowledge property of the Σ-protocol used to prove the knowledge of the link key. The trust assumptions of our voting system are given in Table 22; in Section 9.7, we discuss about how to provide cast-as-intended verification against a malicious device. The proof of receipt-freeness is given in Section 9.6.3.

- **Setup(λ, n_T, t) :** Recall that the public key has the form $\mathsf{pk} = (g, \hat{g}, h, \hat{h}, S, T, (g_i, f_i)_{i=1}^{\ell}, \vec{\mathsf{crs}}, \vec{\mathsf{crs}}')$. To proceed with the setup, choose some pairing-friendly groups $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ of prime order $q > 2^{2\lambda}$ calibrated for the security parameter λ, and choose $g, \hat{g}$ at random, for instance by deriving them from a specification of $\mathbb{G}, \hat{\mathbb{G}}$ and the name of the protocol. Similarly, pick $h, \hat{h}, (g_i)_{i=1}^{\ell}, S, T$ as provably random group elements, as well as crs and crs'. Finally, use a DKG protocol to generate $(f_i)_{i=1}^{\ell}$, distribute the shares of the secret key among the n_T talliers with the threshold t and generate the corresponding commitments on the secret shares.

- **Vote($\mathsf{id}, \nu, \mathsf{RS}, \mathsf{PB}$) :** the vote protocol is done in interaction between the voter id, the rerandomizing server RS and the public board PB. It is illustrated in Fig. 36. First, the voter chooses a random link key lk using the LGen algorithm. Using this link key, the voter encrypts the desired voting option ν with the LEnc algorithm, yielding the valid ciphertext CT. In parallel, the voter computes a random commitment for interactively proving the knowledge of lk in zero-knowledge, using the commitment algorithm of the Σ-protocol. (Namely, in our scheme, the voter samples six random $r_1, r_2, r_3, r'_1, r'_2, r'_3 \in \mathbb{Z}_q$, computes $\hat{l}'_1 = \hat{g}^{r_1}\hat{h}^{r'_1}$, $\hat{l}'_2 = \hat{g}^{r_2}\hat{h}^{r'_2}$ and $\hat{l}'_3 = \hat{g}^{r_3}\hat{h}^{r'_3}$ and uses $(\hat{l}_1, \hat{l}_2, \hat{l}_3)$ as a commitment.) The voter sends $(\mathsf{id}, \mathsf{CT}, C)$ to the server, where C is the commitment.

  The server computes a random challenge $d \xleftarrow{\$} [0, 2^{\lambda} - 1]$ and sends it to the voter.

  The voter answer to this challenge using the Ans algorithm of the Σ-protocol.

  The server verifies the proof. If it is valid, it rerandomizes CT into CT' using Rand, checks that CT' is valid and that the answer of the voter to the challenge was valid. Finally, it checks that no other ciphertext in the public board use the same trace as CT'. If all the verifications succeed, it adds an entry $B' = (\mathsf{id}, \mathsf{CT}')$ to the public board.

- **Check($\nu, \mathsf{id}, B, \mathsf{PB}$) :** To check that the ballot $B = (\mathsf{id}, \mathsf{CT})$ was correctly processed, the voter checks that the last entry of the public board of the form $(\mathsf{id}, \mathsf{CT}')$ is such that $\mathrm{Trace}_{\mathsf{pk}}(\mathsf{CT}) = \mathrm{Trace}_{\mathsf{pk}}(\mathsf{CT}')$.

- **Valid(B, PB) :** to check that a ballot $B = (\mathsf{id}, \mathsf{CT}')$ is valid with respect to the public board, we check that $\mathrm{Ver}_{\mathsf{pk}}(\mathsf{CT}') = 1$ and that no entry of the public board contains a ballot of the form $(\mathsf{id}', \mathsf{CT})$ with $\mathrm{Trace}_{\mathsf{pk}}(\mathsf{CT}) = \mathrm{Trace}_{\mathsf{pk}}(\mathsf{CT}')$.

- **Tally($\mathsf{PB}, \{s_i\}$) :** Let $B_1, \cdots, B_n$ be the list of valid ballots obtained from PB by keeping the last valid ballot for each voter. For all i, we denote $c_1^i, c_2^i, (d_j^i)_{j=1}^{\ell}$ the CPA part of the ballot. The first step of the tally protocol is to aggregate the CPA parts to obtain

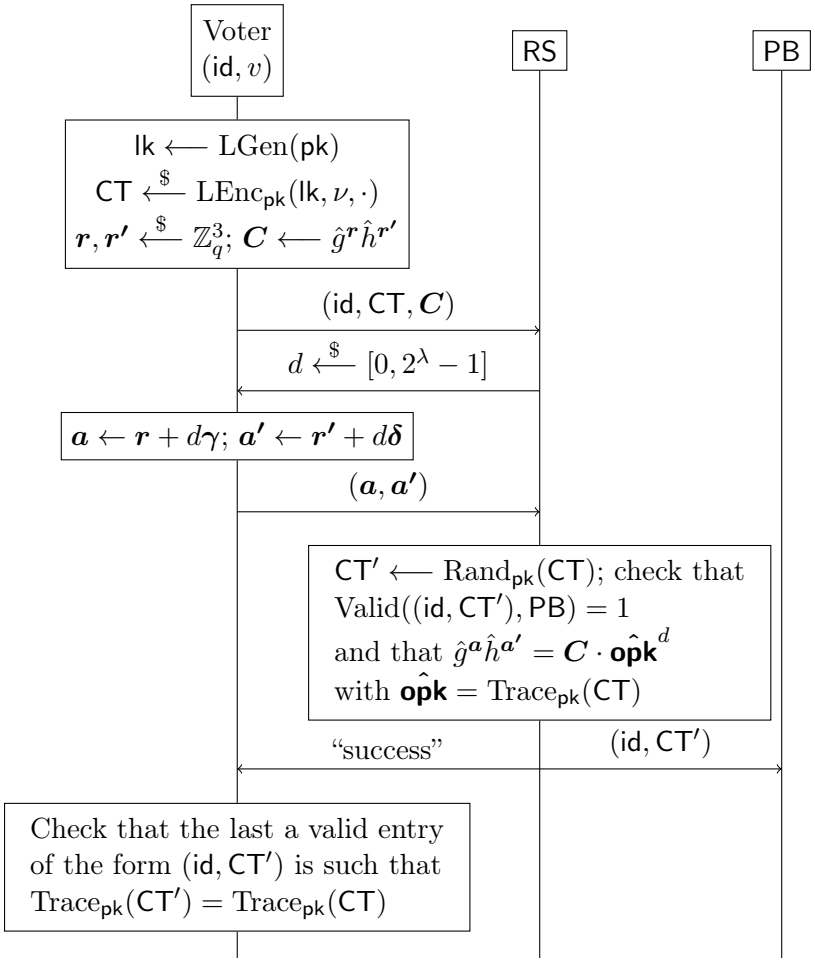


Figure 36: The Vote protocol

$c_1, c_2, (d_j^i)_{j=1}^{\ell}$ as the product of all the $c_1^i, c_2^i, (d_j^i)_{j=1}^{\ell}$'s. Then, the second part is to compute $R_j = d_j c_1^{-\alpha_j} c_2^{-\beta_j}$ for all j, using a threshold decryption protocol. This protocol also returns a transcript Π, which is used for the verifiability. Finally, the tally protocol outputs $(r_i)_{i=1}^{\ell}, \Pi$, where $r_i = \log_{g_i}(R_i)$ is the number of voters that set the ith bit to 1.

- **Verify(PB, Π, r) :** From PB and Π, recomputes the result of the tally r' and check that $r = r'$. Also, check that the transcript Π is valid with respect to PB.

9.6.2 Implementation

We provide a Python implementation in [tre], which evaluates the time to encrypt a ballot for various values of ℓ. Using this implementation, we compare the performance of our scheme to that of BeleniosRF. We conducted our experiments on a laptop with a 1.8Ghz Intel i7 processor and 16GB of RAM running Ubuntu 20.04 LTS and Python 3.7.9. We used the Charm framework [AGM+13] with pbc-0.5.14 and OpenSSL-1.0.1. We used a 159-bits MNT pairing curve. All our results are averaged over 100 runs.

We see in Table 23 that, for a 32-choices race, the encryption time of is a couple of seconds. Looking at the literature, one of the best alternative to our scheme is the one proposed in

Table 23: Time to encrypt a ballot with $\ell = 1, 2, 4, 32$ bits. The times reported are in seconds.

ℓ	1	2	4	32
BeleniosRF	0.07	0.10	0.16	0.94
TREnc	0.17	0.25	0.40	2.49

$\mathsf{Exp}^{\mathsf{s\text{-}cons}}(\lambda, \mathbb{A})$

$(\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_{i=1}^{n_T}, \Pi) \leftarrow \mathsf{Setup}(\lambda, n_T, t)$;
$\boldsymbol{B} \longleftarrow \mathbb{A}(\mathsf{pk}, \Pi)$;
$\mathsf{PB} \longleftarrow \varnothing$;
for $B \in \boldsymbol{B}$ **do**
  if $\mathrm{Valid}(B, \mathsf{PB}) = 1$ **then** $\mathsf{PB} \leftarrow \mathsf{PB} || B$;
$r, \_ \longleftarrow \mathsf{Tally}(\mathsf{PB}, \{s_i\})$;
if $r \neq \mathsf{tally}((\mathrm{Extract}_{\mathsf{sk}}(B))_{B \in \mathsf{PB}})$ **then return** 1 **else return** 0;

$\mathsf{Exp}^{\mathsf{s\text{-}corr}}(\lambda, \mathbb{A})$

$(\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_{i=1}^{n_T}, \Pi) \leftarrow \mathsf{Setup}(\lambda, n_T, t)$;
$\mathsf{id}, \nu, \boldsymbol{B} \longleftarrow \mathbb{A}(\mathsf{pk}, \Pi)$;
if $(\mathsf{id}, \nu) \notin (I \times \mathcal{V})$ **then**
  return 0
$\mathsf{PB} \longleftarrow \Pi || \boldsymbol{B}$;
$B \longleftarrow \mathsf{Vote}_{\mathsf{pk}}(\nu, \mathsf{id})$;
if $\mathrm{Valid}(B, \mathsf{PB}) = 0$ **then return** 1 **else return** 0;

Figure 37: The strong correctness and strong consistency experiments

BeleniosRF [CCFG16], which would be faster than ours.

However, the BeleniosRF scheme has several drawbacks; the most notable one is that the bitlength ℓ is limited to some small value, as the decryption algorithm requires an exponential number of group operations. By contrast, our scheme scales linearly with respect to ℓ.

9.6.3 Receipt-freeness

We now show that our TREnc voting scheme has receipt-freeness. Note that our definition of receipt-freeness is close to the BPRIV definition [BCG+15b] (see Definition 7) for privacy: the main difference is that, in privacy, the rerandomization server is not trusted. However, thanks to the strong rerandomization property, it is easy to adapt the proof of receipt-freeness to obtain privacy. For this reason, we do not give a separate proof of privacy.

Interestingly, the resulting proof is very modular so that it would be possible to adapt our result to another scheme that uses a verifiable, traceable, strongly rerandomizable and TCCA-secure traceable encryption scheme.

To begin with, we perform a necessary sanity check and argue that TREnc has strong correctness and strong consistency (the corresponding experiments are reproduced in Fig. 37).

Strong consistency. We define the Extract algorithm with the decryption algorithm: $\mathrm{Extract}_{\mathsf{sk}}(\mathsf{id}, \mathsf{CT}) = (\mathsf{id}, \mathrm{Dec}_{\mathsf{sk}}(\mathsf{CT}))$. Thanks to the correctness of the traceable encryption scheme, using Extract on a honestly generated ballot from voter id for the voting option $\nu \in \{0,1\}^\ell$ indeed returns (id, ν). Also, the verifiability of the traceable encryption scheme and the homomorphic property of the Tally protocol guarantees that no adversary can win the strong consistency experiment with a non-negligible probability.

Strong correctness. The strong-correctness comes from the correctness of the traceable encryption scheme, and the fact that LGen outputs a link key chosen randomly from a super-polynomial space.

Theorem 15. *Assume that the Σ-protocol used during the* Vote *protocol to prove the knowledge of* lk *has the special soundness property, is zero knowledge and has the correctness property.*

$\mathrm{Exp}^{\mathsf{rf\text{-}b}}(\lambda, \mathbb{A})$

$\mathsf{PB}_1 \longleftarrow \varnothing$; $\mathsf{PB}_2 \longleftarrow \varnothing$;
$\mathsf{pk}, \mathsf{sk}, \tau \longleftarrow \mathcal{O}_{\mathsf{init}}(\lambda, n_T, t)$;
$\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}, \mathcal{O}_{\mathsf{board}}, \mathcal{O}_{\mathsf{voteLR}}, \mathcal{O}_{\mathsf{receiptLR}}}(\mathsf{pk})$;
$r, \Pi \longleftarrow \mathcal{O}_{\mathsf{tally}}()$;
$b' \longleftarrow \mathbb{A}(r, \Pi)$;
if $b' = b$ **then return** 1 **else return** 0;

$\mathcal{O}_{\mathsf{init}}(\lambda, n_T, t)$

$\tau \longleftarrow \bot$;
if $b = 0$ **then**
  $\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_i, \_ \longleftarrow \mathsf{Setup}(\lambda, n_T, t)$;
else
  $\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_i, \tau \longleftarrow \mathrm{SimSetup}(\lambda, n_T, t)$;
return $\mathsf{pk}, \mathsf{sk}, \tau$;

$\mathcal{O}_{\mathsf{cast}}(B)$

if $\mathrm{Valid}(B, \mathsf{PB}_0) = 1$ and $\mathrm{Valid}(B, \mathsf{PB}_1) = 1$ **then**
  $\mathsf{Append}(\mathsf{PB}_0, B)$; $\mathsf{Append}(\mathsf{PB}_1, B)$;

$\mathcal{O}_{\mathsf{board}}()$

return PB_b;

$\mathcal{O}_{\mathsf{voteLR}}(\mathsf{id}, \nu_0, \nu_1)$

if $\nu_0 \notin \mathcal{V}$ or $\nu_1 \notin \mathcal{V}$ **then return** $\bot$;
$s_0, B_0 \longleftarrow \mathsf{Vote}(\mathsf{id}, \nu_0, \mathsf{RS}, \mathsf{PB})$;
$s_1, B_1 \longleftarrow \mathsf{Vote}(\mathsf{id}, \nu_1, \mathsf{RS}, \mathsf{PB})$;
$\mathsf{Append}(\mathsf{PB}_0, B_0)$; $\mathsf{Append}(\mathsf{PB}_1, B_1)$;

$\mathcal{O}_{\mathsf{receiptLR}}(\mathsf{id}, \mathsf{I}, \nu)$

if $\nu \notin \mathcal{V}$ **then return** $\bot$;
$s_0, B_0 \longleftarrow \mathsf{Vote}(\mathsf{id}, \mathsf{I}, \mathsf{RS}, \mathsf{PB})$;
$s_1, B_1 \longleftarrow \mathsf{Vote}(\mathsf{id}, \mathcal{D}^{\mathsf{I}}(\nu), \mathsf{RS}, \mathsf{PB})$;
$\mathsf{Append}(\mathsf{PB}_0, B_0)$; $\mathsf{Append}(\mathsf{PB}_1, B_1)$;
return s_b;

$\mathcal{O}_{\mathsf{tally}}()$

$r_0, \Pi_0 \longleftarrow \mathsf{Tally}(\mathsf{PB}_0, \mathsf{sk})$;
$\Pi_1 \longleftarrow \mathrm{SimProof}(\mathsf{PB}_1, r_0, \tau)$;
return r_0, Π_b;

Figure 38: The receipt-free experiment and its oracles, where $\mathcal{V}$ is the set of the voting options.

Assume that the tally protocol is zero knowledge in a sense that there exists a simulator SimProof *such that, for all* (r, PB)*, if* (r', Π) *is the output of* Tally*, then* $\mathrm{SimProof}(\mathsf{PB}, r, \tau)$ *is computationally indistinguishable from* Π*, where* τ *is a trapdoor generated by* SimSetup*.*

Then, under the SXDH assumption and in the random oracle model, TREnc has receipt-freeness as of Definition 24.

To improve readability, we reproduce the receipt-freeness experiment in Fig. 38.

Proof. First, we explicit the deceiving strategy $\mathcal{D}$. For this purpose, recall that we only consider non-restrictive adversaries. Therefore, at each call to $\mathcal{O}_{\mathsf{receiptLR}}$ with input (I, ν), I is a compatible instruction (*i.e.* a Turing machine that takes as input $(\mathsf{id}, \mathsf{PB})$ and returns a valid ciphertext CT_0 with respect to PB, as well as a commitment $\boldsymbol{C}$. It then takes as input a challenge d and returns a valid answer $\boldsymbol{a}, \boldsymbol{a'}$, as well as a receipt s.)

In this setting, $\mathcal{D}$ creates three copies of I and runs them in parallel. For the first copy, it inputs a random challenge d_1; for the second copy, it inputs a random challenge $d_2 \neq d_1$. This allows to extract the secret link key osk which corresponds to the produced ballot CT, using the special soundness property. Then $\mathcal{D}$ uses this link key to produce another ciphertext CT_1 with the desired voting option ν, and computes the commitment $\boldsymbol{c'}$ honestly. The RS answers to $(\mathsf{id}, \mathsf{CT}_1, \boldsymbol{c'})$ with some random challenge d that $\mathcal{D}$ plays in the third copy of I to get the receipt s.

To prove receipt-freeness, we give a succession of games $H_0, \cdots, H_5$ such that H_0 corresponds to the experiment $\mathrm{Exp}^{\mathsf{rf\text{-}0}}$ while H_5 corresponds to the experiment $\mathrm{Exp}^{\mathsf{rf\text{-}1}}$. For each of these games, we denote S_i the probability that H_i outputs 1.

Game 1: In this game, we create a third fake board PB and answer each call to $\mathcal{O}_{\mathsf{board}}$ with PB. For this purpose, we update PB the same way we update PB_0, and we replace every validity check that implies PB_0 with its counterpart with PB. This way, the adversary's view does not change. Clearly $S_1 = S_0$.

Game 2: In this game, we compute Π using $\mathrm{SimProof}(r, \mathsf{PB})$ instead. By assumption, $|S_2 - S_1|$ is negligible.

Game 3: In this game, each time a new ballot B is added to a board (and thus is valid with respect to this board), we use the secret key to decrypt the encryption part and abort if this is not a valid plaintext. Clearly, $|S_3 - S_2|$ is negligible by the verifiability of the traceable encryption scheme.

Game 4: In this game, each time the adversary calls $\mathcal{O}_{\mathsf{voteLR}}$ with valid inputs (ν_0, ν_1), we add a random ballot for ν_1 (instead of ν_0) to PB, and still add a random ballot for ν_0 to PB_0, using the same link key.

To argue that $|S_4 - S_3|$ is negligible, we use a hybrid argument and show that it is possible to modify the calls to $\mathcal{O}_{\mathsf{voteLR}}$ one by one. More precisely, we construct a succession of hybrids $(H_i)_{\mathbb{N}}$ such that for all i, H_i is Game 4 except that the first i calls to $\mathcal{O}_{\mathsf{voteLR}}$ are handled as usual (the subsequent one are handled as explained above).

We construct an adversary $\mathbb{B}$ that, given i, interacts with $\mathbb{A}$ by simulating H_{i+1}. For this purpose, each time $\mathbb{A}$ makes a $\mathcal{O}_{\mathsf{cast}}$ query with a valid ballot, $\mathbb{B}$ uses the decryption query to decrypt it. This way, $\mathbb{B}$ knows the plaintext that correspond to every ballot in every public board, so that it can compute the tally and run a perfect simulation of H_{i+1}.

However, for the $(i+1)$th call to $\mathcal{O}_{\mathsf{voteLR}}$, $\mathbb{B}$ generates two random ballots B_0 and B_1 for the voting options ν_0 and ν_1 (using the same link key) and plays them in the TCCA game which answers with the challenge ciphertext $\mathsf{CT}^\star$. Then, $\mathbb{B}$ uses this $\mathsf{CT}^\star$ to form the ballot to add to PB instead of a rerandomization of B_0. The remaining of the simulation is ra honestly, except that to compute the result of the tally, $\mathbb{B}$ since it cannot decrypt the ballot cast by $\mathbb{A}$. Finally, if $\mathbb{A}$ wins the simulation, $\mathbb{B}$ outputs 1 in the TCCA game; otherwise, it outputs 0.

Clearly, when the TCCA game rerandomizes CT_0, $\mathbb{B}$ plays a perfect simulation of game H_{i+1} and, when the TCCA game rerandomizes CT_1, $\mathbb{B}$ plays a perfect simulation of game H_i. By the hybrid argument, $|S_4 - S_3|$ is negligible.

Game 5: In this game, whenever the adversary calls $\mathcal{O}_{\mathsf{receiptLR}}$ with instruction I and voting option ν, we compute (s_1, B_1) from $\mathcal{D}^{\mathsf{I}}(v)$, add B_1 (instead of B_0) to PB and return s_1 (instead of s_0). We still compute (s_0, B_0) from I and add B_0 to PB_0.

Remark that s_1 is obtained from I with the same inputs as for s_0 so that $s_1 = s_0$. Remark that as in the previous transition, it is easy to give a hybrid argument and show that Game 5 is indistinguishable from Game 4, thanks to the TCCA-security of the encryption scheme.

Conclusion. We deduce that $|\Pr\left(\mathrm{Exp}^{\mathsf{rf\text{-}0}} = 1\right) - \Pr\left(\mathrm{Exp}^{\mathsf{rf\text{-}1}} = 1\right)|$ is negligible, so that TREnc has receipt-freeness. □

9.7 Adapting the scheme to provide cast-as-intended verification

The Benaloh challenge is a counter-measure to protect the voters against a cheating voting device, which would encrypt another voting option than the chosen one. It works as follows: first, the

voter decides whether to cast or to audit the ballot. In the cast scenario, the voter types on the device the desired voting option. Otherwise, they type a random and independent option. In any case, the voting device is not aware of the choice and must produce a ballot. The voter then reveals if they want to cast or audit it. When the voter chooses to audit, the voting device must reveal an opening of the encrypted ballot (for instance, the randomness used for encryption). The voter then inputs the ballot and this opening to an auditing device which checks whether the voting device encrypted the correct voting option.

Interestingly, the voter may gain confidence as to whether the cast ballot contains the correct voting option, but this cannot be used to break receipt-freeness because the audited ballots are never cast (they instead are spoiled using some mechanism to prevent casting). However, when the ballots are re-randomized, this strategy cannot be used as it is. Indeed, the cheating voting device can always display a honestly generated ballot but cast a dishonestly generated one which uses the same trace, hence defeating the Benaloh challenge. In addition, it may be desirable to publish the spoiled ballots for various reasons; for instance to delegate a part of the verification or to avoid having to transfer data from the voting device to the auditing device. Unfortunately, the published spoiled ballots may then be used as a receipt.

9.7.1 Adapting our scheme for the Benaloh challenge

We propose to slightly adapt the voting protocol so that the voting device can no longer defeat the Benaloh challenge, even if the re-randomization server is dishonest. Our modification does not compromising receipt freeness, even if (a part of) the spoiled ballots are published. (Recall, however, that the re-randomization server is supposed honest when considering receipt-freeness.) The voter still decides in advance whether to audit or to cast, and picks a random voting option when auditing. The voting device encrypts the choice and commits to some random group elements as before. Then, the re-randomization server replies with the re-randomized ballot and a random challenge. At this point, the voter reveals if they want to audit or cast:

- In the cast scenario, the voting device answers to the challenge, the server checks the ZKP and the validity of the ballot and adds it to the public board. The voter then checks that the added ballot corresponds to the one sent by the server.
- In the audit scenario, the voting device and the server reveal the randomness used, so that the voter can check that the ballot opens to the correct voting option.

At the end of the interaction, the voter chooses which spoiled ballots they want to publish.

The audit and cast protocols are pictured in Fig. 39. In this figure, $(s_i)_{i=1}^n$ is a bitstring that indicates which of the previously spoiled ballots the voter wants to publish. For such ballots, the voter can delegate the opening and just check that there is an entry with the correct CT' and ν. Note that allowing the voter to have some spoiled ballots published may expose them to coercion if the adversary is active during the voting phase. It is also more demanding for the public board. For this reason, if there is a direct channel from the voting device or the re-randomization server to the auditing device (for instance, by flashing a QR code), it may be preferable not to let the voters publish any spoiled ballot.

Theorem 16. *Assume that the Σ-protocol used during the* Vote *protocol to prove the knowledge of* Ik *has the special soundness property, is zero knowledge and has the correctness property.*

Assume that the tally protocol is zero knowledge in a sense that there exists a simulator SimProof *such that, for all (r, PB), if (r', Π) is the output of* Tally*, then* SimProof(PB, r, τ) *is computationally indistinguishable from Π, where τ is a trapdoor generated by* SimSetup*.*

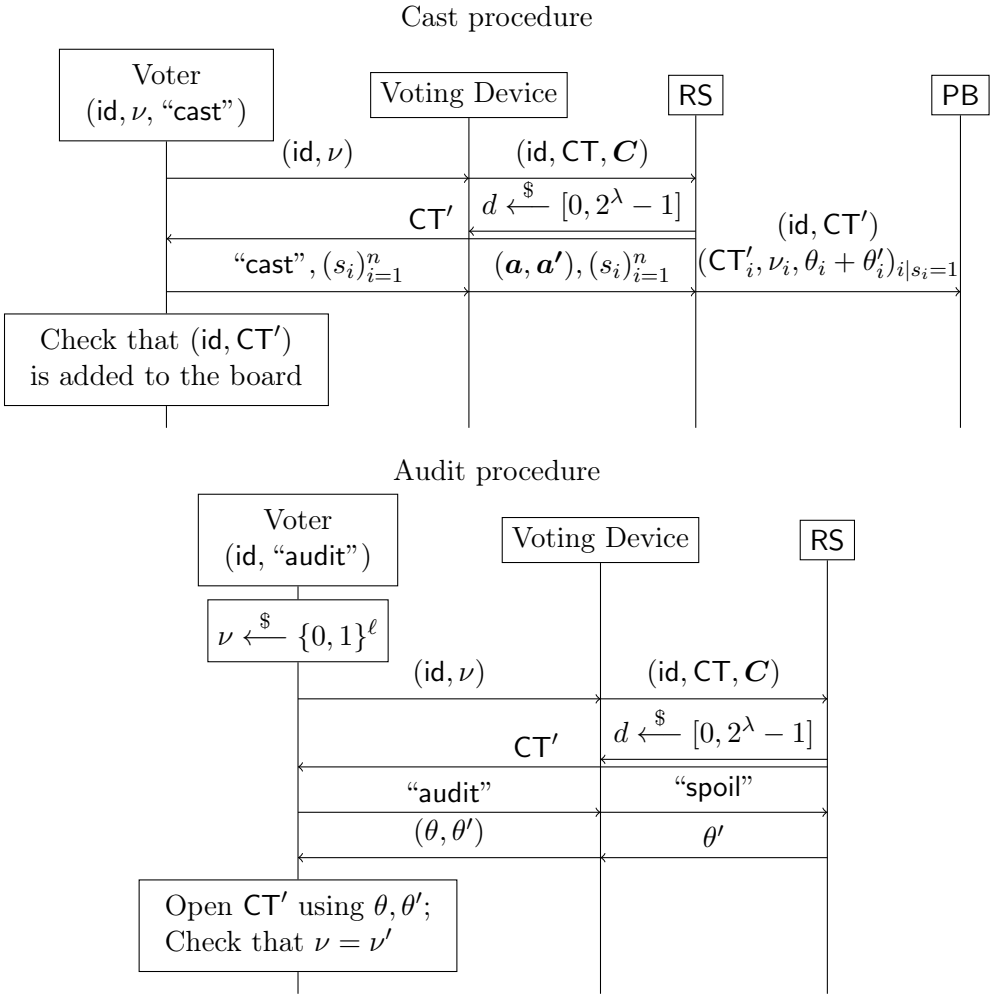


Figure 39: The Vote protocol, adapted to the Benaloh's challenge, where $(\mathsf{CT}'_i, \theta_i + \theta'_i, v_i)_{i=1}^n$ is the list of previously spoiled ballots. θ and θ' are respectively the randomnesses used to encrypt and rerandomize a ballot.

Then, under the SXDH assumption and in the random oracle model, the modified voting system presented in this section has receipt-freeness as of Definition 24.

Proof sketch. We give the new deceiving strategy $\mathcal{D}$. First, $\mathcal{D}$ makes several copies of I, denoted A, B and C. It runs A honestly in interaction with RS, until the first time A sends a "cast" query. Let CT_0 be the ballot created by A and CT^0 the honest re-randomization given by RS. At this point, $\mathcal{D}$ runs B with the exact same inputs, but replaces the last challenge with another randomly chosen challenge. Since A answered "cast" when given an identically distributed challenge, B will also cast with some non-negligible probability (otherwise $\mathcal{D}$ rewinds and picks another challenge again). This allows $\mathcal{D}$ to extract osk with the special soundness property and to produce another ballot CT_1 which has the same trace as CT_0, but encrypts the desired voting option ν. $\mathcal{D}$ asks RS to spoil CT^0 and sends $(\mathsf{id}, \mathsf{CT}_1, \boldsymbol{C})$ to RS, which answers with d and CT^1, where $\boldsymbol{C}$ is a the commitment in the Σ-protocol computed by $\mathcal{D}$. Using a third copy of I, $\mathcal{D}$ rewinds A to replace CT^0 by CT^1 and the last challenge by d. From the TCCA security, C cannot distinguish CT^0 from CT^1. Consequently, C also outputs "cast" with some non-negligible probability (otherwise $\mathcal{D}$ sends "spoil" to the server and starts over). When C eventually outputs "cast", $(s_i)_{i=1}^n$, $\mathcal{D}$ sends $(\boldsymbol{a}, \boldsymbol{a'}), (s'_i)_{i=1}^{n'}$ to the server, so that the bits set at 1 corresponds to the spoiled ballots that C wants published. Finally, $\mathcal{D}$ outputs C's output as the receipt.

Note that $\mathcal{D}$ is a PPT, which means that there is a non-zero (but negligible) probability that $\mathcal{D}$ does not terminate in polynomial time. For instance, if I casts with probability $1/2$, there is a probability $1/2^{2^\kappa}$ that I does not cast after 2^κ interactions. A less extreme example is where I tries to fake the proof of knowledge of osk by picking a specific challenge d and casting only when d is given as a challenge. Then there is a probability of $2^{-\lambda}$ that RS picks this d as a challenge, in which case $\mathcal{D}$ will never be able to extract osk since I will never cast with a challenge $d' \neq d$.

Now, we show that the couple (s_1, B_1) produced by $\mathcal{D}$ in interaction with RS is indistinguishable from the couple (s_0, B_0) produced by I, even if the adversary can see the public board. Indeed, let $\mathbb{A}$ be an adversary which can distinguish both distributions with an advantage ε. We construct an adversary $\mathbb{B}$ for TCCA which interacts with I and $\mathbb{A}$ by simulating $\mathcal{D}$ and RS. $\mathbb{B}$ runs $\mathcal{D}$'s algorithm, but instead of replacing CT^0 by CT^1, it replaces it by the CT^* obtained in the TCCA game, using the ciphertexts CT_0 and CT_1. The remaining of the experiment is simulated honestly, so that $\mathbb{B}$ wins the TCCA game with the same advantage ε.

Finally, remark that if publishing the spoiled ballots is not an option, it is even easier to achieve receipt-freeness since $\mathcal{D}$ no longer needs to compute $(s'_i)_{i=1}^{n'}$. □

9.7.2 On the fly cast-as-intended verification

The solution from Section 9.7.1 allows to use the Benaloh challenge without modifying the protocol too much. In particular, casting requires the same number of interactions for the voter. The main drawback is that checking may be delayed at some point *after* casting, which could be embarrassing if there is no revoting policy. This can be the case, for instance, if there is no direct channel from the voting device and the rerandomization server to the auditing device.

In this section, we sketch an alternative solution which allows on-the-fly cast-as-intended verification, so as to offer the same properties as the original Benaloh challenge, assuming a channel from the public board to the auditing device. The idea is that instead of choosing cast or audit, the voter chooses "cast", "audit-private" or "audit-public". When "audit-private" is chosen, the re-randomization server sends the randomness used to re-randomize the ciphertext and the voting device must display the randomness used to encrypt the ballot. When "audit-public" is chosen, the voting device sends the randomness used for encryption to the server which

publishes the ballot, the necessary data for opening it and the corresponding voting option. This way, the voter can check the opening of the ballot with the auditing device.

When this solution is used, the adversary can now instruct the voter to "audit-public" at some specific points, which would prevent our deceiving strategy from working (completely rewinding is no longer an option since we cannot rewind the public board). Consequently, we need a more refined deceiving strategy, described as follows, where I is the instruction given by the adversary.

1. Run honestly until I decides to cast.
2. Let $(\mathsf{CT}_I, d_I, \mathsf{CT}'_I)$ be the last conversation.
   (a) Using a second copy of I, extract the link key.
   (b) Use "audit-private" to have RS drop CT'_I and submit CT_V, a ciphertext which contains the desired voting option. The server answers with d_V and CT'_V.
   (c) Rewind I to replace d_I and CT'_I with d_V and CT'_V.
3. If I casts again, we are done. Otherwise, audit-private to have RS drop CT'_V
4. Rewind I back to before step 2 but replace d_I and CT'_I by RS's when given again the ballot CT_I. Audit-private and rewind until I decides to cast again.

With the same arguments as in Theorem 16, we can prove that this achieves receipt-freeness.

9.8 Conclusion

In this collaboration, we proposed a new definition of receipt-freeness, which better models the risk of vote buying. This definition considers that the vote buyer may give any instruction to the voter, and ask the latter to provide a receipt to prove that the instruction was followed. With this approach, receipt-freeness becomes closer to coercion-resistance: the main limitations are that we do not consider forced-abstention attacks and that the adversary cannot ask the voter to give away their voting material. Interestingly, our definition does not make any assumption about the eligibility mechanism nor the registration phase, and considers that they are independent from receipt-freeness. This means that we can use any long term secret key to authenticate the voters, such as the electronic identity card used in Estonia or the health insurance system. This way, it is easier to argue that the voter is not going to give away their credential for a small amount of money.

An unusual specificity of our voting scheme is that the Vote protocol requires the voter to send several messages, in interaction with a rerandomization server. This requirement comes from the cryptographic techniques that we deployed, that feature an interactive ZKP to prevent the adversary from providing the ballot to vote with. In electronic voting, it is often considered that the situation where the voter can "vote and go" is preferable, and that the Vote protocol should be non-interactive. We argue that this interaction is actually required for the voting device: on the voter side, the user experience would be similar to that of the Helios voting protocol. In practice, the voting device needs to exchange many additional messages with the server, for instance to establish a TLS channel. Therefore, the interactive nature of the Vote protocol might not be detrimental. In any case, an interesting future work would be to investigate on the possibility to design a verifiable receipt-free voting scheme that does not use any assumption on the registration phase, but where the voter only needs to send a single message.

Conclusion

In hindsight, this thesis was articulated around three thematics: security definitions, security proofs and protocol design. To conclude this manuscript, we summarize our main contributions and mention some possible future works.

Security definitions

In provable security, providing a security definition is a fundamental key step. It is important to make sure that the formalization that we provide actually corresponds to the security property we want to capture. Unfortunately, the existing definitions are not stabilized, and it is often required to adapt an existing definition to the specificities of a protocol. This is all the more problematic when we want to assess additional, less standard security properties, such as receipt-freeness and coercion-resistance.

Our contributions. In this thesis, we studied the academic definition of coercion-resistance provided in [JCJ05] and remarked that it did not model the possibility of revoting. When revoting is allowed, we detected a shortcoming in the JCJ protocol where various side-information – which are publicly available during the tally protocol – allow the adversary to gain a non-negligible advantage in guessing whether the coerced voter obeyed or not, using bayesian inference. Hence, we designed a new definition of coercion-resistance, that properly models revoting and the potential presence of *dummy* ballots, *i.e.* ballots cast with an invalid credential, but not by the coercer. This definition covers a wider variety of scenarios compared to the JCJ definition, and compares the probability that the adversary correctly guesses the behavior of the coerced voter in the real protocol (*i.e.* whether they obey or not) to that probability in an ideal protocol, where the only available information are the result (which includes the number of voters that contributed to this result), the number of ballots cast and the number of eligible voters. This way, our definition does not state that the adversary cannot gain a non-negligible advantage (*i.e.* it can still use – for instance – bayesian inference); rather, it states that this advantage is the same as in the ideal protocol. Intuitively, when the information contained in the result are exactly those required by the regulation, a protocol that satisfies our definition would be as coercion-resistant as possible.

A notion related to coercion-resistance is that of receipt-freeness. By contrast with coercion-resistance, there is no canonical definition of receipt-freeness in the literature: some interesting definitions are given, for instance, in [MN06, KZZ15, CCFG16, DPP22b]. The problem with these definitions is that they do not model the vote-buying scenario, where an adversary can give some specific instructions to the voters and reward those who followed the instructions: for each of these definitions, it is easy to design a voting protocol that verifies the definition but for which vote buying would not be prevented. For this reason, we propose a new definition of receipt-freeness, that was calibrated to address vote buying. In this definition, we consider that some malicious voters may be willing to follow some arbitrary instructions given by the adversary, providing that they are compatible with the voting protocol. This makes the notion of receipt-

freeness closer to that of coercion-resistance, and more suitable for real-world threats. The main differences with coercion-resistance is that it does not consider forced-abstention attacks, and that the adversary cannot ask the voters to give away their voting credential.

Future works. Readily, there are a few short-term future works that might be worth investigating. First, we remarked that the BPRIV definition suffered from a small glitch which caused it to reject too many voting protocols. Indeed, recall that in this definition, there are two public board PB_0 and PB_1 and the adversary is able to cast a ballot in both boards. yet, we only verify that the ballot is valid with respect to PB_b, the board that the adversary is able to see, which might lead to some invalid ballots being added to PB_0. As explained in Section 1.3.2, the naive way to fix this is to check that the ballot is valid in both boards, but further investigations are required to verify that the remaining properties of the BPRIV definition are preserved; namely, we need to verify that the fixed definition still implies that the voting protocol securely realizes some ideal voting protocol in an universally composable framework. Similarly, we adapted the BPRIV definition in Section 9.1 to obtain a new definition of receipt-freeness, and it would be interesting to prove that this definition also implies that the voting protocol similarly realizes an ideal voting protocol.

More generally, the lack of composability is an open problem in electronic voting: the existing definitions are limited, interdependent and make various assumptions on the nature of the protocol or the trust assumptions. Consequently, it is often impossible to use an existing definition for a protocol it was not designed for: it would use different phases, participants and trust assumptions. For instance, the BPRIV definition does not model the fact that the adversary might actively impersonate some talliers during the tally phase. For this reason, it would be interesting to provide more modular, composable security definitions. In Section 6.6, we used the universally composable framework of [CCL15] to exhibit a reduction from a situation where the adversary can impersonate some talliers during the tally protocol to a situation where the adversary is inactive during the tally phase. Hence, a possible approach would be to generalize this result by giving some definitions and conditions under which a similar reduction would be possible. This way, we would be able to create a security framework which gives more modularity on the trust assumptions.

Security proofs

The security proofs are probably what kept us busy for the longest during this thesis. Providing a satisfying level of provable security is definitely a non-trivial task, and we managed to prove various security properties – such as privacy, verifiability, coercion-resistance and receipt-freeness – for our different voting systems. In particular, we used the SUC-framework, which resulted in complex and hard to verify hand-written proofs. Due to the complexity of those proofs, an interesting future work would be to investigate the possibility of machine-checking them. In [CDDW18], for instance, EasyCrypt [BDG+13] has been used to check various security proofs on Belenios. Compared to game-based definitions, however, the simulation-based definitions of the universally composable frameworks are different in nature. Therefore, the first step would be to determine whether the existing tools such as EasyCrypt allow to machine-check the proofs in the SUC framework.

Protocol design

Apart from writing proofs, a large part of this thesis was spent designing protocols. However, we made several concessions in the process, which might lead to interesting future works.

Our contributions. In this thesis, we provide a toolbox for generic MPC in the ElGamal setting, and show that it can be used to achieve full tally-hiding in electronic voting, which addresses the threat of Italian attacks. In particular, we design some explicit tally protocols for Condorcet-Schulze, STV, Majority Judgment and the D'Hondt method, and show that they are efficient and practical, even compared to other approaches based on MPC in the Paillier setting. We also propose CHide, which uses this toolbox to achieve our definition of coercion-resistance. Interestingly, the complex functionalities provided by the toolbox, such as sorting, allow CHide to be more scalable than the original JCJ protocol.

Finally, we also propose TREnc, a voting system that aims at achieving receipt-freeness. Compared to the two other proposals, TREnc is probably more ready-to-go as it is extremely similar to Helios. In particular, it does not require the talliers to run any MPC protocol outside of the DKG protocol and the threshold decryption.

Future works. First, despite the proof of security in the SUC framework, our toolbox does not provide accountability. This can be detrimental during the tally phase, as people are waiting for the result to be published. If the adversary is able to make the protocol abort without being punished, it may use this as a strategy to diminish the public confidence in the protocol. For this reason, providing accountability and dispute resolution (*i.e.* a way to punish at least a malicious participant when the protocol aborts, without punishing any honest one) is extremely important. In the SUC framework, the messages are authenticated, which means that the participants can blame each other using the messages they received. However, an authentication is not an identification: when Alice receives a message m from Bob, Alice knows that m comes from Bob but cannot use this knowledge to convince a third party, since she might have forged m herself. For this reason, accountability requires a dispute resolution protocol which clearly states which participant is to blame in any given situation. A possible solution is to use non-repudiable signatures, but the latter are often more expensive to compute compared to, for instance, a message authentication code. Therefore, signing every single message could lead to an efficiency issue. Analyzing which message must be signed and how to solve the disputes with the signed messages would be an interesting future work.

Concerning coercion-resistance, a challenging topic is that of registration. More precisely, it is often required that the adversary is inactive during the registration phase, or that the latter is perfect, in a sense that the adversary cannot record any of the messages exchanged between the voter and the registrars. This can be achieved, for instance, by an in person registration process. However, if the registration is made online, then the assumptions on the registration may be too strong. Indeed, even if we assume that the registration is done through an untappable channel, the adversary might perform the registration instead of the voter and, when required to authenticate itself, forwards the authentication messages to the voter. This way, the adversary would be able to receive the voter's credential directly from the registrars. Therefore, designing a registration protocol that supports the presence of an active adversary can be an interesting future work.

Finally, all of our constructions were based on the DDH assumption, or the SXDH assumption which is a variant of DDH in the context of bilinear groups. Yet, the emergence of quantum computing is looming large, so that the DDH problem might become easy in the foreseeable future. Consequently, a post-quantum solution will be required soon. In this context, we mentioned in Section 4.1.3 that FHE might be an interesting lead to design a post-quantum MPC toolbox. On this subject, an interesting starting point would be the contribution of [KLO+19].

Appendix A

ZK-TCPA security of the ElGamal threshold encryption scheme

In this Section, we prove Theorem 1 that states that the ElGamal threshold encryption scheme is ZK-TCPA under the DDH assumption. To ease the readability, we reproduce the ZK-TCPA game here, as well as the simulator.

$\mathrm{Sim}_{n_T,t}((x,y),m,A,(s_i)_{i\in A})$

Requires: $A \subset [1,n_T]$ has size $|A| \leq t$

$S \longleftarrow A \bigcup \mathrm{Complete}(A, n_T)$;
for $(i,j) \in ([1,n_T]\backslash A) \times S$ **do**
  $\Lambda_{i,j}^S \longleftarrow \prod_{k\in S\backslash\{j\}} \frac{i-k}{j-k}$;
$w_0 \longleftarrow y/m$;
for $i \in A$ **do** $w_i \longleftarrow x^{s_i}$;
for $i \in S\backslash(A\bigcup\{0\})$ **do** $w_i \xleftarrow{\$} G$;
for $i \in [1,n_T]\backslash S$ **do** $w_i \longleftarrow \prod_{j\in S} w_j^{\Lambda_{i,j}^S}$;
return $(w_i)_{i\in[1,n_T]\backslash A}$;

$\mathrm{Exp}^{\mathsf{ZK\text{-}TCPA}}(\lambda,\mathbb{A})$

$\mathsf{pk},\mathsf{sk},(h_i,s_i)_{i=1}^{n_T},\Pi \leftarrow \mathsf{Setup}(\lambda,n_T,t)$;
$A \leftarrow \mathbb{A}(\mathsf{pk},(h_i)_{i=1}^{n_T})$;
$b \xleftarrow{\$} \{0,1\}$;
if $|A| > t$ **or** $A \not\subset [1,n_T]$ **then return** b;
$m \longleftarrow \mathbb{A}((s_i)_{i\in A})$;
$r \xleftarrow{\$} \mathscr{R}$;
$C \longleftarrow \mathrm{Enc}_{\mathsf{pk}}(m,r)$;
$S_0 \longleftarrow \mathrm{Sim}_{n_T,t}(C,m,A,(s_i)_{i\in A})$;
$S_1 \longleftarrow (\mathrm{PartDec}(C,s_i))_{i\notin A}$;
$b' \longleftarrow \mathbb{A}(C,S_b)$;
if $b' = b$ **then return** 1 **else return** 0;

Theorem 1. *The threshold ElGamal encryption scheme is ZK-TCPA in the ROM and under the DDH assumption.*

To establish this result, we consider Shamir's secret sharing scheme, with a threshold t and n_T participants. This way, the Setup algorithm consists of the following:

- Pick a random generator g;
- Pick a random polynomial $f \in \mathbb{Z}_q[X]$ of degree t;
- For all $i \in [1,n_T]$, set $s_i = f(i)$ and $h_i = g^{s_i}$;
- Set $\mathsf{sk} = f(0)$, $h = g^{\mathsf{sk}}$ and $\mathsf{pk} = (g,h)$;
- Return $\mathsf{pk},\mathsf{sk},(h_i,s_i)_{i=1}^{n_T}$.

Finally, we consider that the algorithm $\mathrm{PartDec}((x,y),s)$ returns $w = x^s$ (the ZKP part is discussed in Section 3.2.1), and we recall that the decryption of a ciphertext (x,y) is derived from the partial decryptions using Lagrange interpolation.

Proof of Theorem 1. We give a succession of games $H_t, \cdots, H_0$. Each game is a copy of the ZK-TCPA game, except that the adversary has to corrupt exactly i participants in game H_i. Remark that, when the adversary corrupts t participants, the simulator $\mathrm{Sim}_{n_T,t}$ outputs some partial decryptions which are equal to (and thus perfectly indistinguishable from) the real partial decryptions. This is due to Lagrange interpolation. Therefore, for all adversary $\mathbb{A}_t$, $\mathbb{A}_t$ wins H_t with probability $1/2$. Now, for all $i < t$, we exhibit a polynomial reduction (*i.e.* a game hop) from H_i to H_{i+1}, which means that the adversary cannot gain some non-negligible advantage by corrupting less participants.

Game hop. Let $i < t$, $\mathbb{A}_i$ be an adversary for H_i and p_i its probability to win. We construct an adversary $\mathbb{A}_{i+1}$ for H_{i+1} as follows. First, $\mathbb{A}_{i+1}$ is given $(\mathsf{pk}, (h_i)_{i=1}^{n_T})$ in H_{i+1} and forwards this to $\mathbb{A}_i$ which answers with some set $A \subset [1, n_T]$ of size i. Then $\mathbb{A}_{i+1}$ adds to A the smallest element j of $[1, n_T]\backslash A$ to form A' which it plays in H_{i+1}. In return, $\mathbb{A}_{i+1}$ is given pk and $(s_i)_{i\in A'}$. Since $A \subset A'$, $\mathbb{A}_{i+1}$ can send pk and $(s_i)_{i\in A}$ to $\mathbb{A}_i$. The latter answers with some $m \in G$, that $\mathbb{A}_{i+1}$ plays in H_{i+1}. $\mathbb{A}_{i+1}$ is then given a ciphertext $C = (x,y)$ and a set of $n_T - i - 1$ partial tallies. To complete this to a set of $n_T - i$ partial tallies as required for $\mathbb{A}_i$, $\mathbb{A}_{i+1}$ computes $w_j = x^{s_j}$. Finally, it returns $\mathbb{A}_i$'s output.

Reduction to DDH. Now, let p_{i+1} be the probability that $\mathbb{A}_{i+1}$ wins H_{i+1}. To argue that $p_i \approx p_{i+1}$ with up to a negligible difference, we construct an adversary $\mathbb{B}$ for DDH whose advantage in the DDH game is proportional to $|p_{i+1} - p_i|$. Under the DDH assumption, $\mathbb{B}$'s advantage should be negligible, therefore $|p_{i+1} - p_i|$ is also negligible. The adversary $\mathbb{B}$ interacts with $\mathbb{A}_i$ by simulating game H_i as follows.

Line 1. First, $\mathbb{B}$ gets a challenge (g_1, g_2, g_3, g_4) from the DDH game. (For simplicity, we assume that $g_1 \neq 1$; see the reduction from IND-CPA to DDH in Section 2.2.2 to see how to handle this possibility properly.) It sets $g = g_1$, chooses a random $\mathsf{sk} \in \mathbb{Z}_q$ and computes $h_0 = g^{\mathsf{sk}}$. At this point, $\mathsf{pk} = (g, h_0)$ is an ElGamal public key. Now, $\mathbb{B}$ chooses a random subset $A' \subset [1, n_T]$ of size i and sets j as the smallest element of $[1, n_T]\backslash A'$. For $k \in A'$, it chooses a random secret share $s_k \in \mathbb{Z}_q$ and computes $h_k = g^{s_k}$. Also, it sets $h_j = g_2$. Afterwards, $\mathbb{B}$ computes $S = A' \bigcup \mathrm{Complete}(A', n_T)$ (this set of $t+1$ elements contains 0 and j) and, for $k \in S\backslash(A' \bigcup \{j, 0\})$, sets h_k as a random group element. Finally, for $k \in [1, n_T]\backslash S$, $\mathbb{B}$ computes $h_k = \prod_{\ell\in S} h_\ell^{\Lambda_{k,\ell}}$, where $\Lambda_{k,\ell} = \prod_{m\in S\backslash\{\ell\}} \frac{k-m}{\ell-m}$ modulo q.

The above operations allow $\mathbb{B}$ to simulate the setup: it can now call $\mathbb{A}_i$ with the entry $(\mathsf{pk}, (h_i)_{i=1}^{n_T})$. At this point, $\mathbb{A}_i$ answers with a set A of size i which is equal to A' with probability $1/\binom{n_T}{i}$ (if this is not the case, $\mathbb{B}$ starts over again with some fresh randomness). Since n_T does not depend on the security parameter λ, $\mathbb{B}$ has to start over a constant number of times until $A = A'$.

Line 5. Since $A = A'$, $\mathbb{B}$ can send $(s_j)_{j\in A}$ to $\mathbb{A}_i$ which answers with $m \in G$. To encrypt m, $\mathbb{B}$ sets $x = g_3$ and computes $y = mx^{\mathsf{sk}}$. The ciphertext (x,y) is therefore a well-formed encryption of m with the public key pk. For $k \in A$, $\mathbb{B}$ computes $w_k = x^{s_k}$. For $k = j$, $\mathbb{B}$ sets $w_j = g_4$. For $k = 0$, $\mathbb{B}$ sets $w_0 = x^{\mathsf{sk}}$. For $k \in S\backslash(A \bigcup \{j, 0\})$, $\mathbb{B}$ sets w_k as a random element. For $k \in [1, n_T]\backslash S$, $\mathbb{B}$ computes $w_k = \prod_{\ell\in S} w_\ell^{\Lambda_{k,\ell}}$. Finally, $\mathbb{B}$ sends $(x,y), (s_i)_{i\in[1,n_T]\backslash A}$ to $\mathbb{A}_i$ which answers b' that $\mathbb{B}$ outputs as its guess in the DDH game.

Probability success of $\mathbb{B}$. Now, if the challenge (g, h_j, x, w_j) is a DDH tuple, then $\mathbb{B}$ played $\mathbb{A}_{i+1}$'s simulation of H_i and therefore wins with probability p_{i+1}. However, if the challenge is a random tuple, then $\mathbb{B}$ played a perfect simulation of H_i, but must output 0 to win. Hence it

wins with probability $1 - p_i$. Overall $\mathbb{B}$'s advantage in the DDH game is $\frac{1}{2}|p_{i+1} - p_i|$, so that $|p_{i+1} - p_i|$ is negligible.

Conclusion. By the triangular inequality, for all $i \in [0, t]$ and for all $\mathbb{A}_i$, $\mathbb{A}_i$'s advantage is bounded by $(t - i)\varepsilon_{DDH}$, where ε_{DDH} is some negligible function. Now, if we consider an adversary $\mathbb{A}$ in the ZK-TCPA game, $\mathbb{A}$ must corrupt i participants for some $i \in [0, t]$, therefore $\mathbb{A}$'s probability to win (say, p) is some barycenter of $\mathbb{A}$'s probability to win in each H_i. Consequently, $\mathbb{A}$'s advantage, which can be interpreted geometrically as the distance between the barycenter p and $1/2$, is bounded by the largest of those distances, and therefore is negligible (since they are all negligible). Overall, $\mathbb{A}$'s advantage in the ZK-TCPA game is bounded by $t\varepsilon_{DDH}$, where $\varepsilon_{\mathsf{DDH}}$ is the advantage of some PPT adversary in the DDH game. □

Appendix B

The hybrid argument

The hybrid argument is a fundamental proof strategy in cryptography, that allows to prove the indistinguishaiblity of two distributions. In this thesis, however, we do not want to prove the indistinguishability of two distributions, but rather the equivalence of two games. For this reason, we adapted the statement of [MF21, Theorem 3.17] into Theorem 2, which gives a game-based version of the hybid lemma. A game is an ITM which, in interaction with an adversary, may give it a view which can be considered as a random variable sampled from a specific distribution. Since the view depends on the interactions with the adversary, the game-based version of the hybrid argument is intuitively more expressive than the distribution-based version, and it is not clear that the former is a consequence of the latter. For this reason, we prove Theorem 2 (restated below) in this appendix.

The goal of Theorem 2 is to give a list of easy-to-check conditions that matches the natural proof strategy in game-based definitions. Nevertheless, it is remarkable that, in the statement of the theorem, the succession of hybrids $(H_i)_{\mathbb{N}}$ is constructed in the *reverse* order: H_0 corresponds to G_2 and, when n grows larger, H_n gets closer to G_1. It may be possible that someone may want to construct the succession $(H_i)_{\mathbb{N}}$ in the *natural* order: from G_1 to G_2. For this reason, we give another version of the hybrid lemma, which is Theorem 17, and we prove that this version is also valid.

Due to space and time limitations, it is common that the hybrid lemma is not properly stated or used in the litterature. Thanks to Theorem 2 and 17, it is now not longer to properly use the hybrid lemma.

Theorem 2 (The hybrid lemma)**.** *Let G_1 and G_2 two games. We consider a sequence of games $(H_i)_{i\in\mathbb{N}}$ which are hybrids between G_1 and G_2. With these notations, assume that the following conditions are met:*

1. *For all PPT $\mathbb{A}$, for all security parameter λ, $\Pr(G_2(\lambda,\mathbb{A})=1)=\Pr(H_0(\lambda,\mathbb{A})=1)$.*
2. *For all PPT $\mathbb{A}$ for game G_1, there exists a polynomial $n_{\mathbb{A}}$ such that, for all $\lambda\in\mathbb{N}$, $\Pr(H_{n_{\mathbb{A}}(\lambda)}(\lambda,\mathbb{A})=1)=\Pr(G_1(\lambda,\mathbb{A})=1)$.*
3. *There exists a polynomial P and two transformation T and T' such that, given any PPT adversary $\mathbb{A}_{i+1}$ (resp. $\mathbb{A}_i$) for game H_{i+1} (resp. H_i), $\mathbb{A}_i=T(\mathbb{A}_{i+1})$ (resp. $\mathbb{A}_{i+1}=T'(\mathbb{A}_i)$) is an adversary for game H_i (resp. H_{i+1}) which makes at most $P(\lambda)$ additional transitions.*
4. *There exists a game G which depends on a parameter $b\in\{0,1\}$ such that, for all PPT adversary $\mathbb{B}$, $\varepsilon_{\mathbb{B}}=2|\Pr(G(\lambda,\mathbb{B})=1)-1/2|$ is negligible in λ.*

5. *There exists a PPT* $\mathbb{B}$ *such that, for all* $i \in \mathbb{N}$ *and all PPT* $\mathbb{A}_{i+1}$ *for game* H_{i+1} *(which in turns defines a PPT* $\mathbb{A}_i$ *for* H_i*), we have* $\Pr(G(\lambda, \mathbb{B}^{\mathbb{A}_{i+1}}(i)) = 1 \mid b = 0) = \Pr(H_i(\lambda, A_i) = 1)$ *and* $\Pr(G(\lambda, \mathbb{B}^{\mathbb{A}_{i+1}}(i)) = 1 \mid b = 1) = \Pr(H_{i+1}(\lambda, A_{i+1}) = 1)$.

Then, for all PPT $\mathbb{A}_1$*, there exists a PPT* $\mathbb{A}_2$ *and a PPT* $\mathbb{B}$ *such that*

$$|\Pr(G_1(\lambda, \mathbb{A}_1) = 1) - \Pr(G_2(\lambda, \mathbb{A}_2) = 1)| \leq n_{\mathbb{A}_1}\varepsilon_{\mathbb{B}}.$$

Proof. Let $\mathbb{A}_1$ be a PPT adversary for G_1. From condition 3, and the polynomial $n_{\mathbb{A}_1}$ from condition 2, we construct a succession of adversaries $(\tilde{\mathbb{A}}_i)_{i\in\mathbb{N}}$ s.t. for all i, $\tilde{A}_i$ is an adversary for the hybrid H_i. If $n_{\mathbb{A}_1}(\lambda) = i$, $\tilde{\mathbb{A}}_i$ runs $\mathbb{A}_1$'s algorithm. If $n_{\mathbb{A}_1}(\lambda) < i$, $\tilde{\mathbb{A}}_i$ runs $T'(\tilde{\mathbb{A}}_{i-1})$'s algorithm. Finally, if $n_{\mathbb{A}_1}(\lambda) > i$, $\tilde{\mathbb{A}}_i$ runs $T(\tilde{\mathbb{A}}_{i+1})$'s algorithm. By condition 3, $\tilde{\mathbb{A}}_i$ makes $|n_{\mathbb{A}_1}(\lambda) - i|P(\lambda)$ more transitions than $\mathbb{A}_1$, and is indeed a PPT adversary.

We define $\mathbb{A}_2$ as $\tilde{\mathbb{A}}_0$, which makes at most $n_{\mathbb{A}_1}P$ additional transitions from $\mathbb{A}_1$. Remark that, for all $\lambda \in \mathbb{N}$, conditions 1 and 2 give

$$\begin{aligned}
|\Pr(G_1(\lambda, \mathbb{A}_1) = 1) - \Pr(G_2(\lambda, \mathbb{A}_2) = 1)| &= |\Pr(H_{n_{\mathbb{A}_1}(\lambda)}(\lambda, \mathbb{A}_1) = 1) - \Pr(H_0(\lambda, \mathbb{A}_2) = 1)| \\
&= |\Pr(H_{n_{\mathbb{A}_1}(\lambda)}(\lambda, \tilde{\mathbb{A}}_{n_{\mathbb{A}_1}(\lambda)}) = 1) - \Pr(H_0(\lambda, \tilde{\mathbb{A}}_0) = 1)| \\
&= \left| \sum_{i=0}^{n_{\mathbb{A}_1}(\lambda)-1} \Pr(H_{i+1}(\lambda, \tilde{\mathbb{A}}_{i+1}) = 1) - \Pr(H_i(\lambda, \tilde{\mathbb{A}}_i) = 1) \right|.
\end{aligned}$$

Now, we denote $\tilde{\mathbb{B}}$ the PPT from condition 5. We construct a PPT $\mathbb{B}$ from the game G (from condition 4) as follows: $\mathbb{B}$ chooses a random $i \in [0, n_{\mathbb{A}_1}(\lambda) - 1]$ and interacts with $\tilde{\mathbb{A}}_{i+1}$, using $\tilde{\mathbb{B}}(i)$'s algorithm. This way, we have

$$\begin{aligned}
\varepsilon_{\mathbb{B}} = 2|\Pr(G(\lambda, \mathbb{B}) = 1) - 1/2| &= \frac{2}{n_{\mathbb{A}_1}(\lambda)} \left| \sum_{i=0}^{n_{\mathbb{A}_1}(\lambda)-1} (\Pr(G(\lambda, \tilde{\mathbb{B}}^{\tilde{\mathbb{A}}_{i+1}}(i) = 1) - 1/2) \right| \\
&= \frac{1}{n_{\mathbb{A}_1}(\lambda)} \left| \sum_{i=0}^{n_{\mathbb{A}_1}(\lambda)-1} \Big(\Pr(G(\lambda, \tilde{\mathbb{B}}^{\tilde{\mathbb{A}}_{i+1}}(i)) = 1 \mid b = 1) - \Pr(G(\lambda, \tilde{\mathbb{B}}^{\tilde{\mathbb{A}}_{i+1}}(i)) = 1 \mid b = 0) \Big) \right|.
\end{aligned}$$

By condition 4, $\varepsilon_{\mathbb{B}}$ is negligible. In addition, by condition 5, we have

$$\varepsilon_{\mathbb{B}} = \frac{1}{n_{\mathbb{A}_1}(\lambda)} \left| \sum_{i=0}^{n_{\mathbb{A}_1}(\lambda)-1} (\Pr(H_{i+1}(\lambda, \tilde{\mathbb{A}}_{i+1}) = 1) - \Pr(H_i(\lambda, \tilde{\mathbb{A}}_i) = 1) \right|.$$

Hence,

$$|\Pr(G_1(\lambda, \mathbb{A}_1) = 1) - \Pr(G_2(\lambda, \mathbb{A}_2) = 1)| = n_{\mathbb{A}_1}(\lambda)\varepsilon_{\mathbb{B}}.$$

□

Theorem 17. *Let* G_1 *and* G_2 *two games. We consider a sequence of games* $(H_i)_{i\in\mathbb{N}}$ *which are hybrids between* G_1 *and* G_2*. With these notations, assume that the following conditions are met:*

1. *For all PPT* $\mathbb{A}$*, for all security parameter* λ*,* $\Pr(G_1(\lambda, \mathbb{A}) = 1) = \Pr(H_0(\lambda, \mathbb{A}) = 1)$.
2. *There exists a polynomial* P *and a transformation* T *such that, given any PPT adversary* $\mathbb{A}_i$ *for game* H_i*,* $\mathbb{A}_{i+1} = T(\mathbb{A}_i)$ *is an adversary for game* H_{i+1} *which makes at most* $P(\lambda)$ *additional transitions.*

3. *For all PPT* $\mathbb{A}$ *for game* G_1*, there exists a polynomial* $n_{\mathbb{A}}$ *such that, for all* $\lambda \in \mathbb{N}$*,* $\Pr(H_{n_{\mathbb{A}}(\lambda)}(\lambda, \mathbb{A}_{n_{\mathbb{A}}(\lambda)}) = 1) = \Pr(G_2(\lambda, \mathbb{A}_{n_{\mathbb{A}}(\lambda)}) = 1)$*, where* $\mathbb{A}_{n_{\mathbb{A}}(\lambda)} = T^{n_{\mathbb{A}}(\lambda)}(\mathbb{A})$*.*
4. *There exists a game* G *which depends on a parameter* $b \in \{0,1\}$ *such that, for all PPT adversary* $\mathbb{B}$*,* $\varepsilon_{\mathbb{B}} = 2|\Pr(G(\lambda, \mathbb{B}) = 1) - 1/2|$ *is negligible in* λ*.*
5. *There exists a PPT* $\mathbb{B}$ *such that, for all* $i \in \mathbb{N}$ *and all PPT* $\mathbb{A}_i$ *for game* H_i *(which in turns defines a PPT* $\mathbb{A}_{i+1}$ *for* H_{i+1}*), we have* $\Pr(G(\lambda, \mathbb{B}^{\mathbb{A}_i}(i)) = 1 \mid b = 1) = \Pr(H_i(\lambda, A_i) = 1)$ *and* $\Pr(G(\lambda, \mathbb{B}^{\mathbb{A}_i}(i)) = 1 \mid b = 0) = \Pr(H_{i+1}(\lambda, A_{i+1}) = 1)$*.*

Then, for all PPT $\mathbb{A}_1$*, there exists a PPT* $\mathbb{A}_2$ *and a PPT* $\mathbb{B}$ *such that*

$$|\Pr(G_1(\lambda, \mathbb{A}_1) = 1) - \Pr(G_2(\lambda, \mathbb{A}_2) = 1)| \leq n_{\mathbb{A}_1}\varepsilon_{\mathbb{B}}.$$

Proof. Let $\mathbb{A}_1$ be a PPT for G_1. Bys condition 1, we can interpret $\mathbb{A}_1$ as an adversary $\tilde{\mathbb{A}}_0$ for game H_0. From condition 2, we construct a succession of adversaries $(\tilde{\mathbb{A}}_i)_{\mathbb{N}}$ such that, for all i, $\tilde{\mathbb{A}}_i = T^i(\mathbb{A})$ is a PPT adversary for H_i which makes at most iP more transitions than $\mathbb{A}$. From condition 3, we define the polynomial $n_{\mathbb{A}_1}$, and the PPT $\mathbb{A}_2$ that, given λ, computes $n_{\mathbb{A}_1}(\lambda)$ and runs $\tilde{\mathbb{A}}_{n_{\mathbb{A}_1}(\lambda)}$'s algorithm. This way, for all $\lambda \in \mathbb{N}$,

$$\begin{aligned}|\Pr(G_1(\lambda, \mathbb{A}_1) = 1) - \Pr(G_2(\lambda, \mathbb{A}_2) = 1)| &= |\Pr(H_0(\lambda, \tilde{\mathbb{A}}_0) = 1) - \Pr(H_{n_{\mathbb{A}}(\lambda)}(\lambda, \tilde{\mathbb{A}}_{n_{\mathbb{A}}(\lambda)}) = 1)| \\ &= \left| \sum_{i=0}^{n_{\mathbb{A}_1}(\lambda)-1} \Pr(H_i(\lambda, \tilde{\mathbb{A}}_i) = 1) - \Pr(H_{i+1}(\lambda, \tilde{\mathbb{A}}_{i+1}) = 1) \right|.\end{aligned}$$

Now, let $\tilde{\mathbb{B}}$ be the PPT from condition 5 and G the game from condition 4. We define $\mathbb{B}$, a PPT for game G, that picks a random $i \in [0, n_{\mathbb{A}_1}(\lambda) - 1]$ and runs $\tilde{\mathbb{B}}(i)$'s simulation to $\tilde{\mathbb{A}}_i$. This way,

$$\varepsilon_{\mathbb{B}} = 2|\Pr(G(\lambda, \mathbb{B}) = 1) - 1/2| = \frac{2}{n_{\mathbb{A}_1(\lambda}} \left| \sum_{i=0}^{n_{\mathbb{A}_1}(\lambda)} (\Pr(G(\lambda, \tilde{\mathbb{B}}^{\tilde{\mathbb{A}}_i}(i)) = 1) - 1/2) \right|.$$

By condition 4, $\varepsilon_{\mathbb{B}}$ is negligible. In addition, by condition 5, we have

$$\begin{aligned}\varepsilon_{\mathbb{B}} &= \frac{1}{n_{\mathbb{A}_1(\lambda}} \left| \sum_{i=0}^{n_{\mathbb{A}_1}(\lambda)} (\Pr(G(\lambda, \tilde{\mathbb{B}}^{\tilde{\mathbb{A}}_i}(i)) = 1 \mid b = 1) - \Pr(G(\lambda, \tilde{\mathbb{B}}^{\tilde{\mathbb{A}}_i}(i)) = 1 \mid b = 0)) \right| \\ &= \frac{1}{n_{\mathbb{A}_1(\lambda}} \left| \sum_{i=0}^{n_{\mathbb{A}_1}(\lambda)} (\Pr(H_i(\lambda, \tilde{\mathbb{A}}_i) = 1) - \Pr(H_{i+1}(\lambda, \tilde{\mathbb{A}}_{i+1}) = 1)) \right|\end{aligned}$$

Hence,

$$|\Pr(G_1(\lambda, \mathbb{A}_1) = 1) - \Pr(G_2(\lambda, \mathbb{A}_2) = 1)| \leq n_{\mathbb{A}_1}\varepsilon_{\mathbb{B}}.$$

□

Appendix C

Proof of correctness for the Majority Judgment algorithm

This appendix is dedicated to the proof of Theorem 4, that we restate below. For convenience, we also reproduced Algorithm 88 in Fig. 40.

Theorem 4. *Algorithm 88 returns the set of maxima according to* $\leq_{\text{maj}}$ *in* $O(n_C n_G)$ *comparisons between grades, where* n_C *is the number of candidates and* n_G *the number of grades.*

To prove this theorem, we define the *median sequence* in Definition 31 and remark that $\leq_{\text{maj}}$ is the lexicographic order for the median sequences. Hence, it is important to describe the behavior of the median sequence, which is done in Lemma 17.

Definition 31 (The median sequence)**.** *The median sequence of a sorted n-tuple* u*, denoted* $m(u)$ *is the sequence formed by* $\text{med}(u)$ *followed by* $m(\hat{u})$*.*

Lemma 17. *Let* u *be a sorted n-tuple. For* $k \in [1, n]$*, the* k^{th} *element of the median sequence of* u *is the element of index* $m + (-1)^{k+n} \lfloor k/2 \rfloor$*, where* $m = \lceil \frac{n}{2} \rceil$*.*

Proof. We distinguish the cases where n is even or odd and give a recurrence in k.

Case 1: n **is even.** The first element of the median sequence is u_m by definition. Let $k \geq 1$. Suppose that for $i \in [1, k]$, the i^{th} element of the median sequence is $u_{m+(-1)^i \lfloor i/2 \rfloor}$. By definition, the $(k+1)^{th}$ element of the median sequence is the element of index $\lceil \frac{n-k}{2} \rceil$ of some $(n-k)$-tuple, obtained by removing the first k elements of the median sequence of u.

If k is even, by recurrence hypothesis, the removed elements have indexes $m, m+1, m-1, \cdots, m-(k/2-1), m+k/2$ thus the remaining elements are

$$(u_1, \cdots, u_{m-k/2}, u_{m+k/2+1}, \cdots, u_n).$$

As n and k are even, $\lceil \frac{n-k}{2} \rceil = m - k/2$. Therefore, the $(k+1)^{th}$ element of the median sequence is $u_{m-k/2}$, and since k is even, $m - k/2 = m + (-1)^{k+1} \lfloor \frac{k+1}{2} \rfloor$.

If k is odd, by recurrence hypothesis, the removed elements have indexes $m, m+1, m-1, \cdots, m+(k-1)/2, m-(k-1)/2$ so the remaining elements are

$$(u_1, \cdots, u_{m-(k+1)/2}, u_{m+(k+1)/2}, \cdots, u_n).$$

Since n is even while k odd, $\lceil \frac{n-k}{2} \rceil = m - (k-1)/2$, so the $(k+1)^{th}$ element of the median sequence is the one following $u_{m-(k+1)/2}$ in the above list, namely $u_{m+(k+1)/2}$, with $m+(k+1)/2 = m + (-1)^{k+1} \lfloor \frac{k+1}{2} \rfloor$.

Majority Judgment

Requires: n_C, the number of candidates
n_G, the number of grades
n_V, the number of voters
Inputs: a, the aggregated grade matrix s.t.
$a[i, j]$ is the number of voters who gave the rank j to the candidate i

$m \longleftarrow \max\{m_i \mid m_i \text{ is the median of candidate } i\}$;
$C \longleftarrow \{i \mid m_i = m\}$;
$I^- \longleftarrow 1$; $I^+ \longleftarrow 1$;
$s \longleftarrow 1$;
for $i \in C$ **do**
  $p_i \longleftarrow \sum_{j=1}^{m-1} a_{i,j}$;
  $q_i \longleftarrow \sum_{j=l+1}^{n_G} a_{i,j}$;
  $m_i^- \longleftarrow \lfloor n_V/2 \rfloor - p_i$;
  $m_i^+ \longleftarrow \lfloor n_V/2 \rfloor - q_i$;
while $|C| > 1$ ***and*** $s \neq 0$ **do**
  for $i \in C$ **do**
    if $m_i^- \leq m_i^+$ **then** $s_i \longleftarrow p_i$;
    else $s_i \longleftarrow -q_i$;
  $s \longleftarrow \max\{s_i \mid i \in C\}$;
  $C \longleftarrow \{i \in C \mid s_i = s\}$;
  if $s \geq 0$ **then**
    for $i \in C$ **do**
      $m_i^+ \longleftarrow m_i^+ - m_i^-$;
      $m_i^- \longleftarrow a_{i,m-I^-}$;
      $p_i \longleftarrow p_i - a_i i, m - I^-$;
    $I^- \longleftarrow I^- + 1$;
  else
    for $i \in C$ **do**
      $m_i^- \longleftarrow m_i^- - m_i^+$;
      $m_i^- \longleftarrow a_{i,m+I^+}$;
      $q_i \longleftarrow q_i - a_{i,m+I^+}$;
    $I^+ \longleftarrow I^+ + 1$;
return C;

Figure 40: Reproduction of our algorithm to compute the Majority Judgment

Case 2: n is odd. The first element of the median sequence is u_m by definition. Let $k \geq 1$. Suppose that for $i \in [1,k]$, the i^{th} element of the median sequence is $u_{m-(-1)^i\lfloor i/2 \rfloor}$. By definition, the $(k+1)^{th}$ element of the median sequence is the element of index $\lceil \frac{n-k}{2} \rceil$ of some $(n-k)$-tuple, obtained by removing the first k elements of the median sequence of u.

If k is even, by recurrence hypothesis, the removed elements have indexes $m, m-1, m+1, \cdots, m+(k/2-1), m-k/2$ so the remaining elements are

$$(u_1, \cdots, u_{m-k/2-1}, u_{m+k/2}, \cdots, u_n).$$

As n is odd and k even, $\lceil \frac{n-k}{2} \rceil = m-k/2$. Therefore the $(k+1)^{th}$ element of the median sequence is the one following $u_{m-k/2-1}$ in the above list, namely $u_{m+k/2}$ with $m+k/2 = m-(-1)^{k+1}\lfloor \frac{k+1}{2} \rfloor$.

If k is odd, by recurrence hypothesis, the removed elements have indexes $m, m-1, m+1, \cdots, m-(k-1)/2, m+(k-1)/2$ so the remaining elements are

$$(u_1, \cdots, u_{m-(k+1)/2}, u_{m+(k+1)/2}, \cdots, u_n).$$

As n and k are odds, $\lceil \frac{n-k}{2} \rceil = m-(k+1)/2$. Hence the $(k+1)^{th}$ element of the median sequence is $u_{m-(k+1)/2}$, with $m-(k+1)/2 = m-(-1)^{k+1}\lfloor \frac{k+1}{2} \rfloor$. □

We now exhibit a collection of loop invariants, where a sum indexed with the empty set is 0 and $g_{i,1}, \cdots, g_{i,n_V}$ denote the list of grades received by candidate i, sorted in decreasing order. Note that m is used to denote the best median, and not $\lceil \frac{n_V}{2} \rceil$ as in the previous lemma.

Lemma 18. *In Algorithm 88, the following loop invariants hold at the beginning (line 10) and at the end (line 27) of the while loop.*

1. *For all $i \in C$, $p_i + m_i^- = m_i^+ + q_i$, and this value is the same for all i.*
2. *For all $i \in C$, $m_i^+ \geq 0$ and $m_i^- \geq 0$.*
3. *For all $i \in C$, $p_i = \sum_{j=1}^{m-I^-} a_{i,j}$. Hence $p_i \geq 0$.*
4. *For all $i \in C$, $q_i = \sum_{j=m+I^+}^{n_G} a_{i,j}$. Hence, $q_i \geq 0$.*
5. *Let $L = p_i + m_i^- + m_i^+ + q_i$. The $n_V - L$ first elements of the median sequence are identical for all $i \in C$.*
6. *For all $i \in C$, for all $j \in [1, m_i^-]$, $g_{i,p_i+j} = m-I^-+1$ and, for all $j \in [1, m_i^+]$, $g_{i,n_V-q_i-j+1} = m + I^+ - 1$.*
7. *C contains all the MJ winners.*

Proof. **Initialization.** First of all, we verify that the loop invariants are true after line 7.

Invariants 1 to 4:

We have $p_i + m_i^- = \lfloor n_V/2 \rfloor = m_i^+ + q_i$.

Moreover p_i is the number of grades strictly greater than the median, so by definition of the median, $p_i \leq \lfloor n_V/2 \rfloor$ hence $m_i^- = \lfloor n_V/2 \rfloor - p_i \geq 0$. Similarly, q_i is the number of grades strictly worse than the median, so by definition of the median, $q_i \leq \lfloor n_V/2 \rfloor$ hence $m_i^+ = \lfloor n_V/2 \rfloor - q_i \geq 0$.

Finally, Equalities 3 and 4 are true with $I^- = I^+ = 1$.

Invariant 5:

Initially, $L = p_i + m_i^- + m_i^+ + q_i = 2\lfloor n_V/2 \rfloor$ so if n_V is even, $n_V - L = 0$. Else, $n_V - L = 1$. As the first element of the median sequence is the median, the $n_V - L$ first elements are the same for all candidates in C after line 9.

Invariant 6:

After line 7, p_i is the number of grades strictly greater than the median for candidate i so, for all $j \geq 1$, $g_{i,p_i+j} \geq m$. Moreover m_i^- is lower than the number of grades equal to the median received by i. So for all $j \leq m_i^-$, $g_{i,p_i+j} \leq m$. Hence, for all $j \in [1, m_i^-]$, $g_{i,p_i+j} = m$. Similarly, for all $j \in [1, m_i^+]$, $g_{i,n-q_i-j+1} = m$.

Invariant 7:

After line 9, C contains the candidates who have the best median, thus contains the winners.

Heredity. Assume that the loop invariants are verified at the beginning of the loop, we show that they are preserved at the end of the loop.

We first show the following result, which is a consequence of loop invariants 1 to 4.

Sub-lemma. For all candidates i, $s_i \geq 0$ if and only if $m_i^- \leq m_i^+$.

Let i be a candidate. Suppose $s_i \geq 0$ and $m_i^- > m_i^+$. Then $0 \leq s_i = -q_i \leq 0$ so $q_i = 0$ and as $p_i + m_i^- = m_i^+ + q_i$, we have $p_i + m_i^- = m_i^+$, which contradicts $p_i \geq 0$. Conversely, if $m_i^- \leq m_i^+$, $s_i = p_i \geq 0$.

To show that the loop invariants are preserved, we denote C_1 the set C at the beginning of the loop and C_2 the set C at the end of the loop. Let $i \in C_2$. Let $i \in C_2$, then $i \in C_1$ so the loop invariants hold at the beginning of the loop, for all $i \in C_2$. We denote p_1 the value of p_i at the beginning of the loop and p_2 at the end, and the same for all other variable m_i^-, m_i^+, q_i, I^-, I^+ and L.

Invariants 1 to 4: Let $s = \max\{s_i \mid i \in C\}$. $C_2 = \{i \mid s_i = s\}$.

If $s \geq 0$, then $s_i = s \geq 0$ so $m_1^- \leq m_1^+$ by the sub-lemma. Hence $m_2^+ = m_1^+ - m_1^- \geq 0$, $m_2^- = a_{i,m-I_1^-} \geq 0$.

In addition, $p_2 = p_1 - a_{i,m-I_1^-}$ and $q_2 = q_1$. Therefore $p_2 + m_2^- = p_1 = s_i = s$, which is the same for all i. Moreover $m_2^+ + q_2 = m_1^+ - m_1^- + q_1 = p_1 + m_1^- - m_1^- = p_1 = S$.

Finally, line 20 together with line 21 and loop invariant 3 give $p_2 = \sum_{j=1}^{m-I_2^-} a_{i,j}$, which shows that invariant 3 is preserved. (Invariant 4 is also preserved because $q_2 = q_1$ and $I_2^+ = I_1^+$.)

If $s < 0$, then $s_i = s < 0$ so $m_1^- > m_1^+$ by the sub-lemma. Hence $m_2^- = m_1^- - m_1^+ \geq 0$, $m_2^+ = a_{i,m+I_1^+} \geq 0$, $q_2 = q_1 - a_{i,m+I_1^+}$ and $p_2 = p_1$. So $m_2^+ + q_2 = q_1 = -S_i = -S$, which is the same for all i. In addition $p_2 + m_2^- = p_1 + m_1^- - m_1^+ = m_1^+ + q_1 - m_1^+ = q_1 = -S$. Finally line 26 together with line 27 and loop invariant 4 give $q_2 = \sum_{j=m+I_2^+}^{c} a_{i,j}$, so that invariant 4 is preserved. (Invariant 3 is also preserved because $p_2 = p_1$ and $I_2^- = I_1^-$.)

Invariant 5:

If $s \geq 0$, $m_1^- \leq m_1^+$. Consequently, $p_1 = s_i = s$ and since $p_1 + m_1^-$ is the same for all i, we deduce that m_1^- is the same for all i. In addition we have $p_2 + m_2^- = p_1$ (lines 19 and 20), $m_2^+ = m_1^+ - m_1^-$ (line 18) and $q_2 = q_1$, so

$$\begin{aligned} L_2 &= p_2 + m_2^- + m_2^+ + q_2 \\ &= p_1 + m_1^+ - m_1^- + q_1 \\ &= p_1 + m_1^- + m_1^+ + q_1 - 2m_1^- = L_1 - 2m_1^-, \end{aligned}$$

and since the $n - L_1$ first elements of the median sequence are the same for all candidates in C_1, we only have to show that the $2m_1^-$ next elements are the same for all candidates in C_2. For this purpose, we remark that loop invariant 1 implies that L_1 is even and we suppose $m_1^- > 0$. (If $m_1^- = 0$, our job is already done.)

By Lemma 17, the elements of indexes $n - L_1 + 1, \cdots, n - L_1 + 2m_1^-$ of the median sequence are the elements

$$g_{i,\lceil n/2\rceil+(-1)^{2n-L_1+1}\lfloor (n-L_1+1)/2\rfloor},\ g_{i,\lceil n/2\rceil+(-1)^{2n-L_1+2}\lfloor (n-L_1+2)/2\rfloor},\cdots$$
$$g_{i,\lceil n/2\rceil+(-1)^{2n-L_1+2m_1^-}\lfloor (n-L_1+2m_1^-)/2\rfloor}\ ;$$

which are also the elements

$$g_{i,\lceil n/2\rceil-\lfloor (n+1)/2\rfloor+L_1/2},\ g_{i,\lceil n/2\rceil+\lfloor n/2\rfloor-L_1/2+1},\cdots$$
$$g_{i,\lceil n/2\rceil-\lfloor (n-1)/2\rfloor+L_1/2-m_1^-},\ g_{i,\lceil n/2\rceil+\lfloor n/2\rfloor-L_1/2+m_1^-}.$$

But $L_1 = p_1 + m_1^- + m_1^+ + q_1$ so, by invariant 1, $L_1/2 = p_1 + m_1^- = m_1^+ + q_1$. Since $\lceil n/2\rceil = \lfloor (n+1)/2\rfloor$ and $\lceil n/2\rceil + \lfloor n/2\rfloor = n$ for all n, we can rewrite them as

$$g_{i,p_1+m_1^-}, g_{i,n-q_1-m_1^++1}, \cdots, g_{i,p_1+1}, g_{i,n-q_1-m_1^++m_1^-}.$$

In what follows, we prove that for all $j \in [1, m_i^-]$, $g_{i,n-q_1-m_1^++j} = m + I_1^+ - 1$. Indeed, $n - q_1 - m_1^+ + j = n - q_1 - (m_1^+ - j + 1) + 1$ and since $m_1^+ \geq m_1^- > 0$, $m_1^+ - j + 1 \in [1, m_1^+]$ for all $j \in [1, m_1^-]$, which allows to prove our claim by invariant 6.

In addition, $g_{i,p_1+j} = m - I_1^- + 1$ for all $j \in [1, m_1^-]$ by invariant 6, so the elements listed above are equal to $m - I_1^- + 1, m + I_1^+ - 1, \cdots, m - I_1^- + 1, m + I_1^+ - 1$ and therefore are the same for all $i \in C_2$, which shows that invariant 5 is preserved.

If $s < 0$, $m_1^- > m_1^+$. Consequently, $q_1 = -s_i = -s$ and since $m_1^+ + q_1$ is the same for all i, so is m_1^+. Moreover $m_2^+ + q_2 = q_1$ (lines 25 and 26), $m_2^- = m_1^- - m_1^+$ (line 24) and $p_2 = p_1$ so

$$\begin{aligned} L_2 &= p_2 + m_2^- + m_2^+ + q_2 \\ &= p_1 + m_1^- - m_1^+ + q_1 \\ &= p_1 + m_1^- + m_1^+ + q_1 - 2m_1^+ = L_1 - 2m_1^+, \end{aligned}$$

and since the $n - L_1$ first elements of the median sequence are the same for all candidates in C_1, we only have to show that the $2m_1^+$ next elements are the same for all candidates in C_2. For this purpose, we remark that invariant 1 implies that L_1 is even and we suppose that $m_1^+ > 0$. (If $m_1^+ = 0$, our job is done.)

By Lemma 17, the elements of indexes $n - L_1 + 1, \cdots, n - L_1 + 2m_1^+$ of the median sequence are

$$g_{i,\lceil n/2\rceil+(-1)^{2n-L_1+1}\lfloor (n-L_1+1)/2\rfloor},\ g_{i,\lceil n/2\rceil+(-1)^{2n-L_1+2}\lfloor (n-L_1+2)/2\rfloor},\cdots$$
$$g_{i,\lceil n/2\rceil+(-1)^{2n-L_1+2m_1^+}\lfloor (n-L_1+2m_1^+)/2\rfloor}\ ;$$

which are also the elements

$$g_{i,\lceil n/2\rceil-\lfloor (n+1)/2\rfloor+L_1/2},\ g_{i,\lceil n/2\rceil+\lfloor n/2\rfloor-L_1/2+1},\cdots$$
$$g_{i,\lceil n/2\rceil-\lfloor (n-1)/2\rfloor+L_1/2-m_1^+},\ g_{i,\lceil n/2\rceil+\lfloor n/2\rfloor-L_1/2+m_1^+}.$$

But $L_1 = p_1 + m_1^- + m_1^+ + q_1$ so, by invariant 1, $L_1/2 = p_1 + m_1^- = m_1^+ + q_1$. Since $\lceil n/2 \rceil = \lfloor (n+1)/2 \rfloor$ et $\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$ for all n, we can rewrite them as

$$g_{i,p_1+m_1^-}, g_{n-q_1-m_1^++1}, \cdots, g_{i,p_1+m_1^--m_1^++1}, g_{i,n-q_1}.$$

We now show that for all $j \in [1, m_i^+]$, $g_{i,p_1+m_1^--j+1} = m - I_1^- + 1$. Indeed, $p_1 + m_1^- - j + 1 = p_1 + (m_1^- - j + 1)$ and since $m_1^- > m_1^+ > 0$, $(m_1^- - j + 1) \in [1, m_1^-]$ for all $j \in [1, m_1^+]$, which allows to prove our claim by invariant 6.

In addition, $g_{i,n-q_1-j+1} = m + I_1^+ - 1$ for all $j \in [1, m_1^+]$ by invariant 6, so the elements listed above are equal to $m - I_1^- + 1, m + I_1^+ - 1, \cdots, m - I_1^- + 1, m + I_1^+ - 1$ and therefore are the same for all $i \in C_2$, which shows that invariant 5 is preserved.

Invariant 6:

If $s \geq 0$, $m_1^- \leq m_1^+$ so $p_2 = p_1 - a_{i,m-I_1^-}$ and $m_2^- = a_{i,m-I_1^-}$. But $p_1 = \sum_{j=1}^{m-I_1^-} a_{i,j}$, which is exactly the number of grades strictly greater than $m - I_1^- + 1$ received by i so by definition of $a_{i,m-I_1^-}$, p_2 is the number of grades strictly greater than $m - I_1^-$. Therefore g_{i,p_2+1} is lower than $m - I_1^-$ and as there are $a_{i,m-I_1^-} = m_2^-$ grades equal to $m - I_1^-$, we deduce that $g_{i,p_2+j} = m - I_1^- = m - (I^- + 1) + 1 = m - I_2^- + 1$ for all $j \in [1, m_2^-]$. In addition, for all $j \in [1, m_1^+]$, $g_{i,n-q_1-j+1} = m + I_1^+ - 1$ so, *a fortiori*, for all $j \in [1, m_1^+ - m_1^-]$, $g_{i,n-q_1-j+1} = m + I_2^+ - 1$.

If $s < 0$, $m_1^- > m_1^+$ so $q_2 = q_1 - a_{i,m+I_1^+}$ and $m_2^+ = a_{i,m+I_1^+}$. But $q_1 = \sum_{j=m+I_1^+}^{c} a_{i,j}$, which is exactly the number of grades strictly worse than $m + I_1^+ - 1$ so by definition of $a_{i,m+I_1^+}$, q_2 is the number of grades strictly worse than $m + I_1^+$. Therefore $g_{i,n-q_2}$ is greater than $m + I_1^+$ and as there are $a_{i,m+I_1^+} = m_2^+$ grades equal to $m + I_1^+$, we deduce that $g_{i,n-q_2-j+1} = m + I_1^+ = m + (I^+ + 1) - 1 = m + I_2^+ - 1$ for all $j \in [1, m_2^+]$. In addition, for all $j \in [1, m_1^-]$, $g_{i,p_1+j} = m - I_1^- + 1$ so, *a fortiori*, for all $j \in [1, m_1^+ - m_1^-]$, $g_{i,p_1+j} = m - I_2^- + 1$.

Invariant 7:

Let $b \in C_2$, (namely $b \in C_1$ such that $s_b = s$). We show that for all $a \in C_1 \backslash C_2$, (namely for all $a \in C_1$ such that $s_a < s$), $a <_{maj} b$.

Positive case. Suppose that $s \geq 0$. Let $a \in C_1$ such that $s_a < s$.

Positive-negative case. We first assume that $s_a < 0$. Therefore $s_a < 0 \leq s = s_b$. By the sub-lemma, we have $m_a^- > m_a^+$ and $m_b^- \leq m_b^+$.

Suppose that $m_a^+ < m_b^-$. With the same reasoning as in the proof of invariant 6, we show that the elements of indexes 1 to $n - L + 2m_a^+$ of the median sequence of a and b are the same. Since $m_a^- > m_a^+$, by Lemma 17 and loop invariant 1 and 6, the $n - L + 2m_a^+ + 1$ th elements of the median sequence of a and b are respectively

$$\begin{aligned} &g_{a,p_a+m_a^--m_a^+} = m - I^- + 1 \text{ and} \\ &g_{b,p_a+m_a^--m_a^+} = g_{b,p_b+m_b^--m_a^+} = m - I^- + 1. \end{aligned}$$

However, the $n - L + 2m_a^+ + 2$ th element of the median sequence of a is

$$g_{a,n-q_a+1} < g_{a,n-q_a} = m + I^+ - 1,$$

while b's is

$$g_{b,n-q_a-m_a^++m_a^++1} = g_{b,n-q_b-(m_b^+-m_a^+)+1} = m + I^+ - 1.$$

Therefore $a <_{\text{maj}} b$.

Now suppose that $m_a^+ \geq m_b^-$. As above, the $n - L + 2m_b^-$ first elements of the median sequence of a and b are the same. The elements of index $n - L + 2m_b^- + 1$ are respectively

$$g_{a,p_a+m_a^- - m_b^-} = g_{a,p_a+(m_a^- - m_a^+)+(m_a^+ - m_b^-)} = m - I^- + 1^- \text{ and}$$
$$g_{b,p_a+m_a^- - m_b^-} = g_{b,p_b} > m - I^- + 1.$$

Therefore $a <_{\text{maj}} b$.

Positive-positive case. Now suppose that $0 \leq s_a$. By the sub-lemma, $m_b^- \leq m_b^+$, $m_a^- \leq m_a^+$. Consequently $s_a = p_a$ and $s_b = p_b$ and since $s_a < s_b$, by invariant 1, we have $m_a^- > m_b^-$. Then again, we deduce that the $n - L + 2m_b^-$ first elements of the median sequence are the same and that b wins over a thanks to the next element.

Negative case. Finally, suppose that $s < 0$. Then $s_a < s_b = s < 0$ so, by the sub-lemma, $m_a^- > m_a^+$ and $m_b^- > m_b^+$. Consequently $s_a = -q_a$ and $s_b = -q_b$ and since $s_a < s_b$, by invariant 1, we have $m_b^+ > m_a^+$. Then again, we deduce that the $n - L + 2m_a^+$ first elements of the median sequence are the same. In addition $m_a^- > m_a^+$, so by Lemma 17 and invariants 1 and 6, the $n - L + 2m_a^+ + 1$ th elements of the median sequence of a and b are

$$g_{a,p_a+m_a^- - m_a^+} = m - I^- + 1 \text{ and}$$
$$g_{b,p_a+m_a^- - m_a^+} = g_{b,p_b+m_b^- - m_a^+} = m - I^- + 1.$$

However, the $n - L + 2m_a^+ + 2$ th element for a is

$$g_{a,n-q_a+1} > g_{a,n-q_a} = m + I^+ - 1,$$

while b's is

$$g_{b,n-q_a-m_a^++m_a^++1} = g_{b,n-q_b-(m_b^+ - m_a^+)+1} = m + I^+ - 1.$$

Therefore $a <_{\text{maj}} b$. □

Once the loop invariants are established, we can use them to derive the correctness of our algorithm.

Proof of Theorem 4. **Complexity.** By Lemma 18, $p_i = \sum_{j=1}^{m-I^-} a_{i,j}$ and $q_i = \sum_{j=m+I^+}^{c} a_{i,j}$. But at each iteration, we subtract $a_{i,m-I^-}$ to p_i or $a_{i,m+I^+}$ to q_i so there cannot be more than n_G iterations before both are equal to 0. When $p_i = q_i = 0$ for all i, $s = 0$, which terminates the loop. Hence the Algorithm terminates after $O(n_C n_G)$ comparisons.

Correctness. If the algorithm terminates because $\mid C \mid = 1$, C contains only one element and since C contains the winners, C is the set of winners. Otherwise, $s = 0$. Recall that s is the maximum of s_i and let i such that $s_i = s$. If $m_i^- > m_i^+$, we have $s_i = -q_i$ thus $q_i = 0$, which contradicts $p_i + m_i^- = m_i^+ + q_i$ and $p_i \geq 0$ so $m_i^- \leq m_i^+$ and $p_i = s_i = s = 0$. But $m_i^- \leq m_i^+$ and $p_i + m_i^- = m_i^+ + q_i$. Since $q_i \geq 0$, $q_i = 0$ thus $m_i^- = m_i^+$. Hence, by invariants 6 and 7, each candidate in C are equal with respect to $\leq_{maj}$. Since C contains the winners, C is the set of winners. □

Appendix D

Computing the coercion levels

In the context of coercion-resistance, the framework of [KTV10a] provides an effective methodology to evaluate the *coercion level* of a given protocol, and to compare it with that of an ideal protocol. As explained in Section 7.3, the coercion level can be evaluated thanks to Eq. (3), that we reproduce below. This assumes that the cryptography is perfect, that a large and unpredictable number of ballots is removed during the tally phase and that the adversary is able to compare $\Pr(\mathsf{R}^g|\alpha)$ and $\Pr(\mathsf{R}^g|\beta)$, given R^g with $g \in \{\text{Real}, \text{Ideal}\}$. In this appendix, we give more details about how this comparison can be done as well as some efficient ways to evaluate the formula from Eq. (3). In particular, this allows to understand how the figures from Section 7.3 were obtained.

$$\delta^g = \max_{(\alpha,\beta)} \sum_{\mathsf{R}^g \in M_{\alpha,\beta}} \Pr(\mathsf{R}^g|\beta) - \Pr(\mathsf{R}^g|\alpha). \tag{3}$$

D.1 The coercion level in the ideal game

In the ideal game, $\mathsf{R}^{\text{Ideal}}$ is the vector $\overrightarrow{res}$, and computing $\Pr(\overrightarrow{res}|\alpha)$ for any α can be done thanks to a formula given in the following result from [KTV10a].

Lemma 19 ([KTV10a])**.** *Let (α, β) be two options, n_H the number of honest voters and a pure result $\overrightarrow{res}$ such that $\sum_{i=0}^{C} \mathsf{res}_i = n_H + 1$. Let $\overrightarrow{P}$ be the probability distribution for the honest voters. Assuming $P_\alpha P_\beta \neq 0$, we have $\Pr(\overrightarrow{res}|\beta) \geq \Pr(\overrightarrow{res}|\alpha)$ if and only if $\mathsf{res}_\beta P_\alpha \geq \mathsf{res}_\alpha P_\beta$.*

In addition, we have

$$\Pr(\overrightarrow{res}|\alpha) = \frac{n_H!}{\prod_{k=0}^{C} \mathsf{res}_k!} \left(\prod_{k=0}^{C} P_k^{\mathsf{res}_k} \right) \frac{\mathsf{res}_\alpha}{P_\alpha}.$$

Finally, with $M_{\alpha,\beta} = \{\overrightarrow{res} \mid \Pr(\overrightarrow{res}|\beta) \geq \Pr(\overrightarrow{res}|\alpha)\}$, the optimal value of $\delta^{\text{Ideal}}_{\alpha,\beta}$ is

$$\delta^{\mathsf{Ideal}^{\mathsf{CR}}}_{\alpha,\beta} = \sum_{\overrightarrow{res} \in M_{\alpha,\beta}} (\Pr(\overrightarrow{res}|\beta) - \Pr(\overrightarrow{res}|\alpha)) = \sum_{\overrightarrow{res} \in M_{\alpha,\beta}} \frac{n_H!}{\prod_{k=0}^{C} \mathsf{res}_k!} \left(\prod_{k=0}^{C} P_k^{\mathsf{res}_k} \right) \left(\frac{\mathsf{res}_\beta}{P_\beta} - \frac{\mathsf{res}_\alpha}{P_\alpha} \right).$$

Note that computing the sum over $M_{\alpha,\beta}$ might be expensive due to the size of this set. Thanks to this result, the adversary can compare $\Pr(\mathsf{R}^{\text{Ideal}}|\alpha)$ and $\Pr(\mathsf{R}^{\text{Ideal}}|\beta)$ in $O(1)$ floating operations, as long as it has access to P_α and P_β (note that it does not need the whole distribution), where

the O notation considers that n_H grows to infinity while the other parameters are fixed. To deduce the resulting coercion level, we further analyze this expression to give a representation which is easier to compute. For this purpose, we give Lemma 20, which allows to compute δ^{Ideal} in $O(n_H)$ floating operations.

Lemma 20. *Let (α, β) be two options, n_H the number of honest voters and a pure result $\overrightarrow{res}$ such that $\sum_{i=0}^{C} \mathsf{res}_i = n_H + 1$. Let $\overrightarrow{P}$ be the probability distribution for the honest voters. Assuming $P_\alpha P_\beta \neq 0$, we have*

$$\delta_{\alpha,\beta}^{\mathsf{Ideal}^{\mathsf{CR}}} = n_H! \sum_{x=0}^{n_H} \frac{T_x P_\beta^{T_x - 1} P_\alpha^{N_x - T_x}}{x! T_x! (N_x - T_x)!} (1 - P_\beta - P_\alpha)^x, \text{ where } N_x = n_H - x + 1 \text{ and } T_x = \left\lceil \frac{P_\beta}{P_\beta + P_\alpha} N_x \right\rceil.$$

Proof. First, we partition $M_{\alpha,\beta}$ into $\bigcup_{x=0}^{n_H} M_{\alpha,\beta}^x$, where $M_{\alpha,\beta}^x$ is the subset of all results in $M_{\alpha,\beta}$ where the options other than α and β received exactly x votes. Then, we further partition $M_{\alpha,\beta}^x$ into subsets where $\mathsf{res}_\beta = y$ (and, thus, $\mathsf{res}_\alpha = N_x - y$). Note that due to the condition from Lemma 19, $\mathsf{res}_\beta P_\alpha \geq \mathsf{res}_\alpha P_\beta$ hence y ranges from T_x to N_x. To express the formula from Lemma 19 using this partition, we assume that α and β are the last two voting options (otherwise we reorder them) and denote $\tilde{M}^x = \{\overrightarrow{\tilde{res}} \in \mathbb{N}^{C-2} \mid \sum_{k=0}^{C-2} \mathsf{res}_k = x\}$. With these notations, we have

$$\begin{aligned}
\delta_{\alpha,\beta}^{\mathsf{Ideal}^{\mathsf{CR}}} &= \sum_{x=0}^{n_H} \sum_{\overrightarrow{\tilde{res}} \in \tilde{M}^x} \sum_{y=T_x}^{N_x} \frac{n_H!}{\prod_{k=0}^{C-2} \mathsf{res}_k!} \left(\prod_{k=0}^{C-2} P_k^{\mathsf{res}_k} \right) \frac{P_\beta^y P_\alpha^{N_x - y}}{y!(N_x - y)!} \left(\frac{y}{P_\beta} - \frac{N_x - y}{P_\alpha} \right) \\
&= \sum_{x=0}^{n_H} \sum_{\overrightarrow{\tilde{res}} \in \tilde{M}^x} \frac{n_H!}{\prod_{k=0}^{C-2} \mathsf{res}_k!} \left(\prod_{k=0}^{C-2} P_k^{\mathsf{res}_k} \right) \sum_{y=T_x}^{N_x} \frac{P_\beta^y P_\alpha^{N_x - y}}{y!(N_x - y)!} \left(\frac{y}{P_\beta} - \frac{N_x - y}{P_\alpha} \right).
\end{aligned}$$

Now, we show that, for all $0 \leq T \leq N$, $\displaystyle\sum_{y=T}^{N} \frac{P_\beta^y P_\alpha^{N-y}}{y!(N-y)!} \left(\frac{y}{P_\beta} - \frac{N-y}{P_\alpha} \right) = \frac{T P_\beta^{T-1} P_\alpha^{N-T}}{T!(N-T)!}$.

For this purpose, we first fix some $N \geq 0$ and we proceed by backward iteration over T. First, it is true when $T = N$. Now, suppose that it is true for some $0 < T \leq N$; we show that it is also true for $T - 1$:

$$\begin{aligned}
&\sum_{y=T-1}^{N} \frac{P_\beta^y P_\alpha^{N-y}}{y!(N-y)!} \left(\frac{y}{P_\beta} - \frac{N-y}{P_\alpha} \right) = \\
&= \frac{P_\beta^{T-1} P_\alpha^{N-T+1}}{(T-1)!(N-T+1)!} \left(\frac{T-1}{P_\beta} - \frac{N-T+1}{P_\alpha} \right) + \sum_{y=T}^{N} \frac{P_\beta^y P_\alpha^{N-y}}{y!(N-y)!} \left(\frac{y}{P_\beta} - \frac{N-y}{P_\alpha} \right) \\
&= \frac{P_\beta^{T-1} P_\alpha^{N-T+1}}{(T-1)!(N-T+1)!} \left(\frac{T-1}{P_\beta} - \frac{N-T+1}{P_\alpha} \right) + \frac{T P_\beta^{T-1} P_\alpha^{N-T}}{T!(N-T)!} \\
&= \frac{(T-1) P_\beta^{T-2} P_\alpha^{N-T+1}}{(T-1)!(N-T+1)!}.
\end{aligned}$$

Hence, with $T = T_x$ and $N = N_x$, we deduce that

$$\begin{aligned}
\delta_{\alpha,\beta}^{\mathsf{Ideal}^{\mathsf{CR}}} &= \sum_{x=0}^{n_H} \sum_{\overrightarrow{res} \in \tilde{M}^x} \frac{n_H!}{\prod_{k=0}^{C-2} \mathsf{res}_k!} \left(\prod_{k=0}^{C-2} P_k^{\mathsf{res}_k} \right) \frac{T_x P_\beta^{T_x - 1} P_\alpha^{N_x - T_x}}{T_x!(N_x - T_x)!} \\
&= n_H! \sum_{x=0}^{n_H} \frac{T_x P_\beta^{T_x - 1} P_\alpha^{N_x - T_x}}{x! T_x! (N_x - T_x)!} (1 - P_\beta - P_\alpha)^x \sum_{\overrightarrow{res} \in \tilde{M}^x} \frac{x!}{\prod_{k=0}^{C-2} \mathsf{res}_k!} \prod_{k=0}^{C-2} \left(\frac{P_k}{1 - P_\beta - P_\alpha} \right)^{\mathsf{res}_k} \\
&= n_H! \sum_{x=0}^{n_H} \frac{T_x P_\beta^{T_x - 1} P_\alpha^{N_x - T_x}}{x! T_x! (N_x - T_x)!} (1 - P_\beta - P_\alpha)^x.
\end{aligned}$$

Indeed, the last summation is the sum over all possibilities of a multinomial distribution, which is 1. □

D.2 Modeling the real game

For the real game where revoting is possible, we propose a model in which each honest voter does the following

- Abstain with probability P_0,
- Otherwise, vote for option $\ell > 0$ with probability $p_{\ell,\varnothing}$,
- Revote with probability r_ℓ,
- In this case, choose option $k > 0$ with probability $p_{\ell,k}$.

In this model, a voter may revote at most once. Also, a voter who abstains does not "revote". If a voter who initially wanted to abstain changes their mind, it will count as voting once. This approximation is made because the adversary has access to the number of revotes, and not the number of voters who changed their mind. Also, the probability to revote depends on the initial choice. This is to capture the fact that an announcement on the press can make some voter revote if their candidate has been compromised. Similarly, the probability distribution when revoting depends on the first choice. See Fig. 41 for an illustration. In this figure, the label on the left of an edge denotes the nature of the transition and the label on the right denotes the probability of the transition. On leafs, we used labels of the form (n, P) when n denotes the total number of honest voters who choose this path and P the probability to choose this path. For instance, $P_{1,2}$ is the probability to first vote for option 1 but to finally revote for option 2 while $P_{2,\varnothing}$ is the probability to vote for option 2 and not to revote. Since abstention plays a specific role, we denote it as choice 0, while the other choices actually imply to send a ballot.

Now, we must decide on the information the adversary has access to. The most conservative approach is that it may have access to anything which is not a secret; here, all the probability transitions. With this assumption, if $\overrightarrow{res} = (\mathsf{res}_0, \cdots, \mathsf{res}_C)$ is the number of votes for each option and n_R is the number of revotes, then $\Pr((\overrightarrow{res}, n_R)|\alpha)$ is the probability that the honest voters votes $\mathsf{res}_k - \mathbb{1}_{k=\alpha}$ times for each candidate k and revote n_R times. Given the above tree, for any possible outcome $(\overrightarrow{res}, n_R)$ with n_H honest voters, we have

$$\Pr(\overrightarrow{res}, n_R) = \sum_{H^{n_r}_{\overrightarrow{res}}} \frac{n_H!}{\mathsf{res}_0! \prod_{k=1}^{C} n_{k,\varnothing}! \prod_{\ell=1}^{C} n_{\ell,k}!} P_0^{\mathsf{res}_0} \prod_{k=1}^{C} P_{k,\varnothing}^{n_{k,\varnothing}} \prod_{\ell=1}^{C} P_{\ell,k}^{n_{\ell,k}}, \tag{29}$$

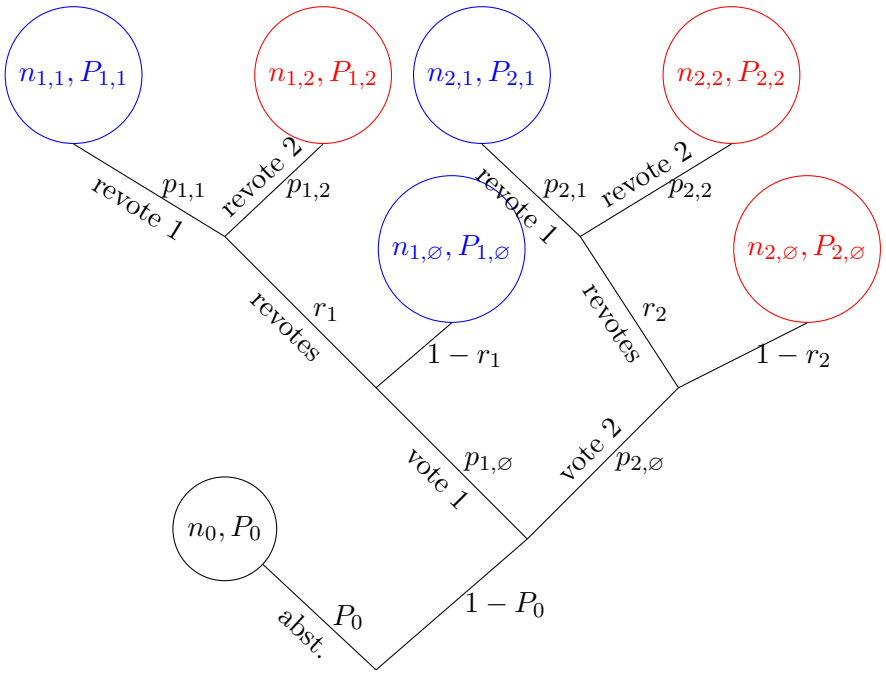


Figure 41: Tree structure for two candidates.

where $H_{\overrightarrow{res}}^{n_r} = \{(n_{k,\varnothing}), (n_{\ell,k}) \mid \sum_{\ell,k} n_{\ell,k} = n_r, \forall k > 0, n_{k,\varnothing} + \sum_{k=1}^{C} n_{\ell,k} = \mathsf{res}_k\}$ is the set of all possible atomic outcomes compatible with $\overrightarrow{res}$ and n_R.

D.3 Quantifying the coercion level in some specific cases

Because of the complexity of Eq. (29), we only consider the specific cases where there are two candidates and the possibility to abstain (*i.e.* $C = 2$). In these simpler cases, we give Lemma 21 which allows to compute the coercion level in $O({n_H}^4)$ floating operations. Interestingly, δ^{Real} does not depend on the $P_{\ell,k}$ directly but only on the sums P_{r1} and P_{r2}, which correspond to the probability to revote for 1 or 2 respectively. This is because the adversary can only observe the number of revotes and has no information about the initial intention of the voter. Hence, they cannot exploit the dependency between the final choice and the first choice. In practice, we can imagine a scenario where the probability to revote and the probability distributions when revoting and when not revoting are known to the adversary, for instance thanks to exit poll or social media, where the voters tell whether they revote and what was their final choice. In this more realistic scenario when the adversary does not know all the probabilities on the tree, the latter can still compare $\Pr(\mathsf{R}^{\text{Real}}|\alpha)$ and $\Pr(\mathsf{R}^{\text{Real}}|\beta)$, which takes $O(n_H)$ floating operations.

Lemma 21. *Let n_H be the number of honest voters, $(\mathsf{res}_0, \mathsf{res}_1, \mathsf{res}_2)$ such that $\mathsf{res}_0 + \mathsf{res}_1 + \mathsf{res}_2 = n_H + 1$ and $n_R \leq n_H - \mathsf{res}_0$. Let $\beta \in \{0, 1, 2\}$ be a voting option. Then the probability $\Pr((\mathsf{res}_0, \mathsf{res}_1, \mathsf{res}_2), n_R|\beta)$ that there are res_0 abstentions, res_1 votes for option 1, res_2 votes for option 2 and n_R revotes when the coerced voter chooses the voting option β is given by the following algorithm.*

- *First, set $\overrightarrow{res} = (\mathsf{res}_0, \mathsf{res}_1, \mathsf{res}_2)$.*

- *If* $\mathsf{res}_\beta = 0$, *return* 0, *otherwise, do* $\mathsf{res}_\beta \leftarrow \mathsf{res}_\beta - 1$.
- *Set* $N = n_H - \mathsf{res}_0 - n_R$, $P_{r1} = P_{1,1} + P_{2,1}$ *and* $P_{r2} = P_{1,2} + P_{2,2}$. *Return*

$$\Pr(\overrightarrow{res}, n_R) = \frac{n_H! P_0^{\mathsf{res}_0}}{\mathsf{res}_0!} \sum_{n_{1,\varnothing}} \frac{P_{1,\varnothing}^{n_{1,\varnothing}} P_{2,\varnothing}^{N-n_{1,\varnothing}}}{n_{1,\varnothing}!(N-n_{1,\varnothing})!} \frac{P_{r1}^{\mathsf{res}_1 - n_{1,\varnothing}}}{(\mathsf{res}_1 - n_{1,\varnothing})!} \frac{P_{r2}^{n_R + n_{1,\varnothing} - \mathsf{res}_1}}{(n_R + n_{1,\varnothing} - \mathsf{res}_1)!},$$

where $\max(0, \mathsf{res}_1 - n_R) \le n_{1,\varnothing} \le \min(\mathsf{res}_1, n_H - \mathsf{res}_0 - n_R)$.

Proof. As shown in Section D.2, $\Pr((\mathsf{res}_0, \mathsf{res}_1, \mathsf{res}_2), n_R|\beta)$ is given by Eq. (29). We explicit the sum over $H_{\overrightarrow{res}}^{n_R}$. First, $0 \le n_{2,\varnothing} \le \mathsf{res}_2$ and $n_{1,\varnothing} + n_{2,\varnothing} = n_H - \mathsf{res}_0 - n_r$ and $\mathsf{res}_0 + \mathsf{res}_1 + \mathsf{res}_2 = n_H$, therefore

$$\begin{aligned} 0 \le & n_{1,\varnothing} \le \mathsf{res}_1 \\ \mathsf{res}_1 - n_R \le & n_{1,\varnothing} \le n_H - \mathsf{res}_0 - n_R. \end{aligned}$$

In addition, $n_{1,\varnothing} + n_{1,1} + n_{2,1} = \mathsf{res}_1$ and $0 \le n_{2,1} \le n_R$ so that

$$\begin{aligned} 0 \le & n_{1,1} \le n_R \\ \mathsf{res}_1 - n_{1,\varnothing} - n_R \le & n_{1,1} \le \mathsf{res}_1 - n_{1,\varnothing}. \end{aligned}$$

Finally, $n_{1,2} + n_{2,2} = n_R - n_{1,1} - n_{2,1} = n_R + n_{1,\varnothing} - \mathsf{res}_1$, thus

$$0 \le n_{1,2} \le \mathsf{res}_2 - n_{2,\varnothing} = n_R + n_{1,\varnothing} - \mathsf{res}_1.$$

Let $N = n_H - \mathsf{res}_0 - n_R$, $N_{n_{1,\varnothing}} = \mathsf{res}_1 - n_{1,\varnothing}$ and $N'_{n_{1,\varnothing}} = n_R + n_{1,\varnothing} - \mathsf{res}_1$. Considering all these inequalities, we have

$$\begin{aligned} \Pr(\overrightarrow{res}, n_R) &= \frac{n_H! P_0^{\mathsf{res}_0}}{\mathsf{res}_0!} \sum_{n_{1,\varnothing}} \frac{P_{1,\varnothing}^{n_{1,\varnothing}} P_{2,\varnothing}^{N-n_{1,\varnothing}}}{n_{1,\varnothing}!(N-n_{1,\varnothing})!} \sum_{n_{1,1}} \frac{P_{1,1}^{n_{1,1}} P_{2,1}^{N_{n_{1,\varnothing}} - n_{1,1}}}{n_{1,1}!(N_{n_{1,\varnothing}} - n_{1,1})!} \sum_{n_{1,2}} \frac{P_{1,2}^{n_{1,2}} P_{2,2}^{N'_{n_{1,\varnothing}} - n_{1,2}}}{n_{1,2}!(N'_{n_{1,\varnothing}} - n_{1,2})!} \\ &= \frac{n_H! P_0^{\mathsf{res}_0}}{\mathsf{res}_0!} \sum_{n_{1,\varnothing}} \frac{P_{1,\varnothing}^{n_{1,\varnothing}} P_{2,\varnothing}^{N-n_{1,\varnothing}}}{n_{1,\varnothing}!(N-n_{1,\varnothing})!} \sum_{n_{1,1}} \frac{P_{1,1}^{n_{1,1}} P_{2,1}^{N_{n_{1,\varnothing}} - n_{1,1}}}{n_{1,1}!(N_{n_{1,\varnothing}} - n_{1,1})!} \frac{(P_{1,2} + P_{2,2})^{N'_{n_{1,\varnothing}}}}{N'_{n_{1,\varnothing}}!}. \end{aligned}$$

Now, $\mathsf{res}_1 - n_{1,\varnothing}$ is equal to the number of revotes for 1, so that $\mathsf{res}_1 - n_{1,\varnothing} \le n_R$. Therefore, the innermost summation is also on the full domain for $n_{1,2}$. Hence,

$$\begin{aligned} \Pr(\overrightarrow{res}, n_r) &= \frac{n_H! P_0^{\mathsf{res}_0}}{\mathsf{res}_0!} \sum_{n_{1,\varnothing}} \frac{P_{1,\varnothing}^{n_{1,\varnothing}} P_{2,\varnothing}^{N-n_{1,\varnothing}}}{n_{1,\varnothing}!(N-n_{1,\varnothing})!} \frac{(P_{1,1} + P_{2,1})^{N_{n_{1,\varnothing}}}}{N_{n_{1,\varnothing}}!} \frac{(P_{1,2} + P_{2,2})^{N'_{n_{1,\varnothing}}}}{N'_{n_{1,\varnothing}}!} \\ &= \frac{n_H! P_0^{\mathsf{res}_0}}{\mathsf{res}_0!} \sum_{n_{1,\varnothing}} \frac{P_{1,\varnothing}^{n_{1,\varnothing}} P_{2,\varnothing}^{N-n_{1,\varnothing}}}{n_{1,\varnothing}!(N-n_{1,\varnothing})!} \frac{P_{r1}^{\mathsf{res}_1 - n_{1,\varnothing}}}{(\mathsf{res}_1 - n_{1,\varnothing})!} \frac{P_{r2}^{n_r + n_{1,\varnothing} - \mathsf{res}_1}}{(n_r + n_{1,\varnothing} - \mathsf{res}_1)!}, \end{aligned}$$

where $P_{r1} = P_{1,1} + P_{2,1}$ and $P_{r2} = P_{1,2} + P_{2,2}$.

Note that this can be computed in $O(n_H)$ floating operations. Since we must range over all possibilities for $\mathsf{res}_0, \mathsf{res}_1, \mathsf{res}_2, n_r$, the overall δ^{Real} can be computed in $O({n_H}^4)$ floating operations. □

In Section 7.3, we used the formulas from Lemma 21 and 20 to evaluate the coercion levels. The only exceptions are Figures 46 and 24, where the real coercion levels for 513 and 1025 voters were obtained with a Monte-Carlo estimates with 1000000 iterations, which gives a sufficiently small confidence interval for our purpose. The reason why is that evaluating the formula from Lemma 21 would be too long for such values of n_V.

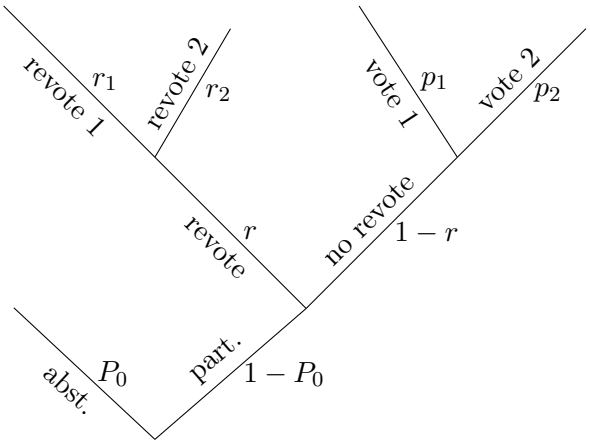


Figure 42: Simplified tree structure for two candidates.

D.4 The impact of the parameters

We now give some relevant parameters and evaluate their impact on the real and ideal coercion levels. This analysis shows that the scenarios presented in Sections 7.3.2 and 7.3.3 are not the only plausible ones that might occur. In Lemma 21, we saw that δ^{Real} only depends on P_0, the probability of abstention, P_{r1} (resp. P_{r2}), the probability to revote for option 1 (resp. 2) and $P_{1,\varnothing}$ (resp. $P_{2,\varnothing}$), the probability to vote for option 1 (resp. 2) without revoting. To give a better representation of these parameters, we define

$$r = (P_{r1} + P_{r2})/(P_{r1} + P_{r2} + P_{1,\varnothing} + P_{2,\varnothing}),$$

which is the probability to revote (after voting),

$$p_1 = P_{1,\varnothing}/(P_{1,\varnothing} + P_{2,\varnothing}) \text{ and } p_2 = P_{2,\varnothing}/(P_{1,\varnothing} + P_{2,\varnothing}),$$

which are the probabilities to vote for option 1 and option 2 when voting exactly once and

$$r_1 = P_{r1}/(P_{r1} + P_{r2}) \text{ and } r_2 = P_{r2}/(P_{r1} + P_{r2}),$$

the probabilities to vote for option 1 and 2 when revoting (see Fig. 42 for an illustration).

A first important parameter is r, the probability of revoting. In Fig. 43, we let it vary from 0 to 1, and we plot the coercion level in JCJ and in the ideal protocol for various values of (p_1, p_2) and (r_1, r_2). As expected, since the leakage we detected comes from the revotes, both are the same when there is no revote. Note that when everyone revotes, there is no coercion resistance anymore. This is due to the considered evasion strategy, which instructs the voter to vote once (which is detected if they are the only one to do so).

Now, an important point to notice is that, if (p_1, p_2) and (r_1, r_2) are close, the difference between the ideal and the real coercion level is small, unless there are a lot of revotes. On the other hand, if both distributions are opposite, the difference is noticeable, even with a reasonable proportion of revotes. Therefore, another interesting parameter to consider is the distance between (p_1, p_2) and (r_1, r_2), defined as $2|p_1 - r_1|$, which we call the *bias*. In Fig. 44, we let the bias vary from 0 to 2 and we plot the coercion level in the real and ideal protocols, for various probability of revoting. As expected, when the bias is maximal, there is no coercion

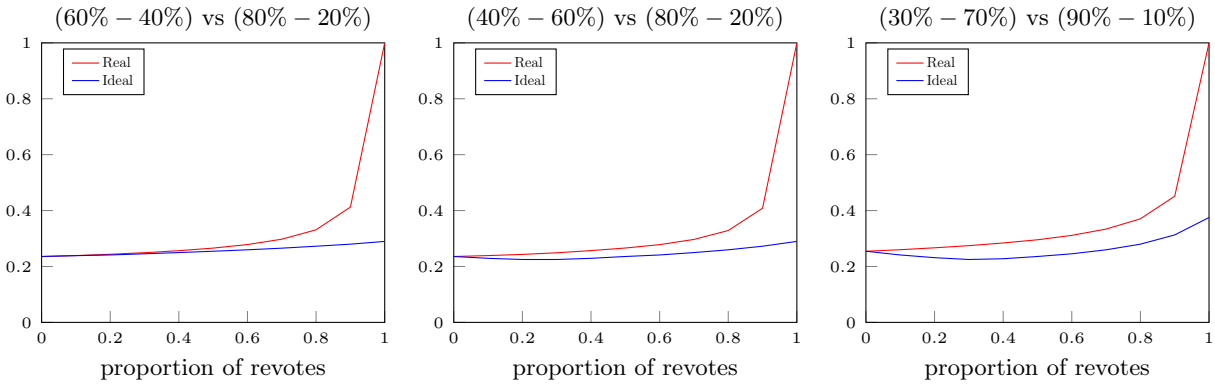


Figure 43: Coercion levels as a function of the probability of revoting; with 20 voters, 2 candidates and 30% abstention, with three different distributions between the candidates.

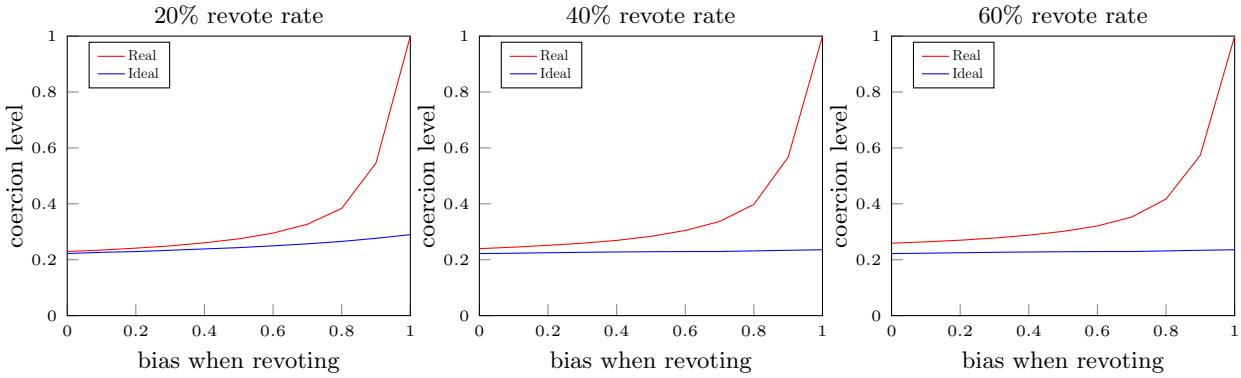


Figure 44: Coercion levels as a function of the bias when revoting; with 20 voters, 2 candidates and 30% abstention.

resistance at all (this corresponds to the scenario of Section 7.2.1). Note, however, that the leakage is non-zero when there is no bias. This, once again, is due to the fact that the adversary can count the number of revotes, and therefore have a non-negligible advantage to decide whether the coerced voter voted or abstained.

Another natural parameter to consider is the abstention rate. Until now, it was fixed at 0.3 in our experimentations. However, it can variate a lot in real-life elections. In Fig. 45, we plot the real and ideal coercion levels as a function of the abstention rate, with various probabilities of revoting and bias. As expected, when there is no abstention, the attacker can trivially break coercion-resistance with a forced-abstention attack. Similarly, when everyone but the coerced voter abstains, there is no coercion-resistance (both situations are captured by the ideal model).

Finally, the last parameter of interest is the number of honest voters. In Fig. 46, we plot the real and ideal coercion levels for $16, 32, 64, 128$, 256, 512 and 1024 honest uncoerced voters, with various probabilities of revoting and bias. As expected, the coercion level decreases as n_H grows for both the ideal and real game. Interestingly, the difference remains noticeable as long as there is enough revotes.

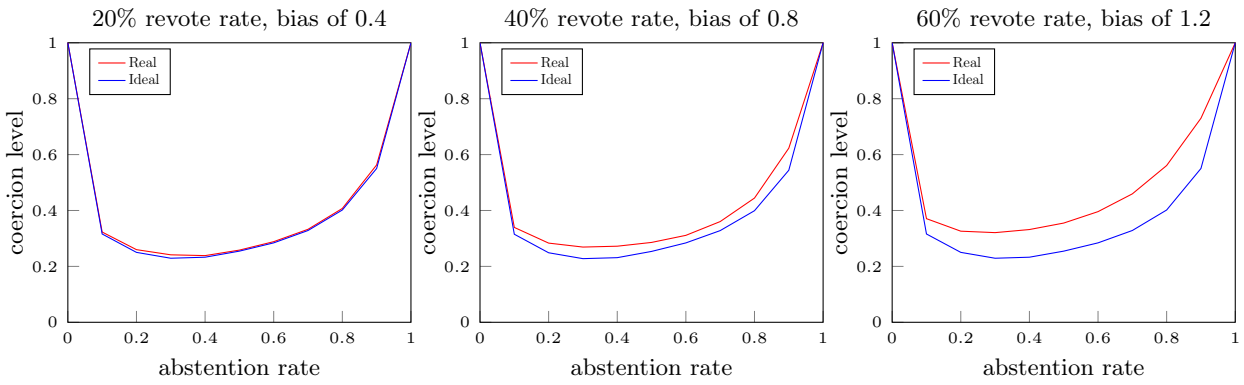


Figure 45: Coercion level as a function of the abstention rate; with 20 voters and 2 candidates.

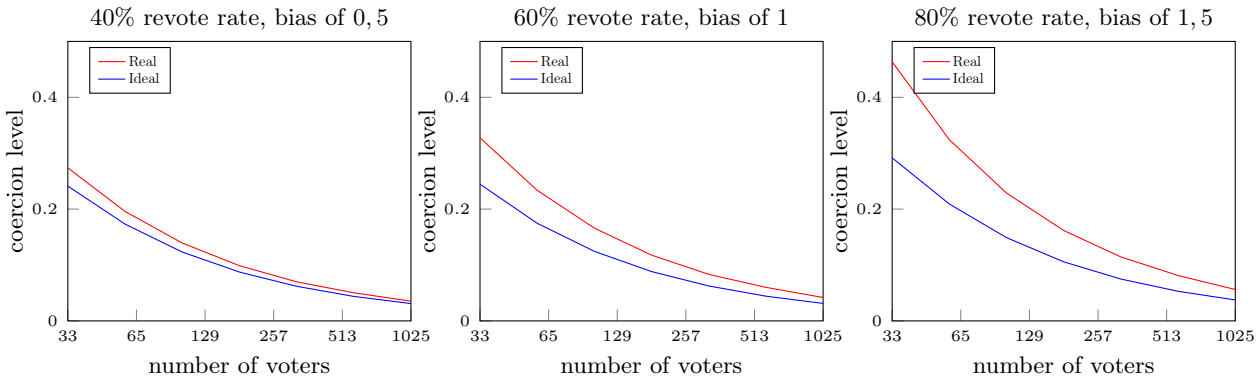


Figure 46: Coercion level as a function of the number of voters; with 2 candidates and 30% abstention

Appendix E

Proof of privacy for CHide

In this appendix, we provide a proof that CHide is private, as of Definition 9 (the corresponding real and ideal games are reproduced below). The proof is extremely similar to that of coercion-resistance and some parts are reproduced *verbatim*, with some little tweaks to take into account the differences between $\text{Real}^{\mathsf{CR}}$ and $\text{Real}^{\mathsf{Priv}}$. Just as for coercion-resistance, we make the same assumptions as in JCJ; however, we no longer assume an anonymous channels for voting.

Theorem 18. *Under the DDH assumption and in the ROM, assuming a SUC-secure decryption mixnet, CHide is private.*

Proof. We give a succession of games such that Game 0 is the real game and Game 9 is the ideal game. We consider a PPT $\mathbb{A}_0$ for Game 0. For Game i, we construct a PPT adversary $\mathbb{A}_i$ for this game and we denote S_i the probability that $\mathbb{A}_i$ wins this game. (To ease the notations, we drop the dependency in λ when the context is clear.) For all i, we show that $|S_{i+1} - S_i|$ is negligible, which proves that $|S_0 - S_9|$ is also negligible.

Game 1: In this game, the adversary no longer takes part into the whole tally process at line 17, but only in the decryption mixnet process. Instead, it is given the result of all the conditional gates, computed by a trusted party. With a similar argument as in Theorem 6, we can show that the cleansing phase up to the decryption mixnet is SUC-secure, so that there exists an adversary $\mathbb{A}_1$ such that $|S_1 - S_0|$ is negligible.

Game 2: In this game, the adversary no longer takes part in the decryption mixnet and is instead given the result at line 18, computed by a trusted party. Since the decryption mixnet is supposed UC-secure, we can similarly construct an adversary $\mathbb{A}_2$ such that $|S_2 - S_1|$ is negligible.

Game 3: In this game, the adversary is no longer given the output of the conditional gates. Just as in the transition to Game 2 in the proof of Theorem 8, under the DDH assumption and in the ROM, there exists $\mathbb{A}_3$ such that $|S_3 - S_2|$ is negligible.

Game 4: In this game, we modify the sequence B so that the honest voters no longer revote. Instead, for all honest voter x, we replace all but the last occurrence of the form (x, ν) in B by an occurrence of the form $(x \to \tilde{x}, \nu)$ where $\tilde{x}$ is a fresh and unique negative number. Then, at line 12, when i is of the form $x \to \tilde{x}$, we give x to the adversary but we add a ballot of the form $\mathsf{Vote}_{\mathsf{pk}}(c, \nu)$ with a fresh random (fake) credential c.

Although this transition is slightly different from the transition to Game 4 in the proof of coercion resistance, the reduction argument is the same.

Algorithm 108: $\text{Real}^{\mathsf{Priv}}$

Requires: $\lambda, n_T, C_t, n, n_A, n_C, \mathscr{B}, \mathbb{A}$

$\mathsf{pk}, \mathsf{sk}, (h_i, s_i)_{i=1}^{n_T}, \Pi^{\mathsf{S}} \leftarrow \mathsf{Setup}(\lambda, n_T, t)$;
$(c_i, \pi_i), \Pi^{\mathsf{R}} \longleftarrow \mathsf{Register}(\mathsf{pk}, n)$;
$\mathsf{PB} \longleftarrow \Pi^{\mathsf{S}} || \Pi^{\mathsf{R}}$;
$A \longleftarrow \mathbb{A}(\mathsf{pk}, \mathsf{PB}, \{s_i \mid i \in C_t\})$;
$j, \nu_0, \nu_1 \longleftarrow \mathbb{A}(\{c_i \mid i \in A\})$;
(* chooses the voter to observe *);
if $|A| \neq n_A \vee j \notin [1, n] \backslash A$ **then**
  return 0;
$B \xleftarrow{\$} \mathscr{B}([1, n] \backslash A, n_C)$;
for $(i, \nu_i) \in B$ **do**
  $\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(i, \mathsf{PB})$;
  $\mathsf{PB} \longleftarrow \mathsf{PB} || \mathsf{Vote}_{\mathsf{pk}}(\nu_i, c_i)$;
$\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(i, \mathsf{PB}, \texttt{"end for"})$;
$b \xleftarrow{\$} \{0, 1\}$;
$\mathsf{PB} \longleftarrow \mathsf{PB} || \mathsf{Vote}_{\mathsf{pk}}(\nu_b, c_j)$;
$\mathbb{A}^{\mathcal{O}_{\mathsf{cast}}}(\mathsf{PB})$;
$r, \Pi \longleftarrow \mathsf{Tally}^{\mathbb{A}}(\mathsf{PB}, \{s_i\})$;
$b' \longleftarrow \mathbb{A}()$;
if $\nu_0, \nu_1 \in [1, n_C] \wedge b' == b$ **then**
  return 1 **else return** 0;

Algorithm 109: $\text{Ideal}^{\mathsf{Priv}}$

Requires: $\lambda, n_T, C_t, n, n_A, n_C, \mathscr{B}, \mathbb{A}$

;
;
;
$A \longleftarrow \mathbb{A}(\lambda)$;
$j, \nu_0, \nu_1 \longleftarrow \mathbb{A}()$;
(* chooses the voter to observe *);
if $|A| \neq n_A \vee j \notin [1, n] \backslash A$ **then**
  return 0;
$B \xleftarrow{\$} \mathscr{B}([1, n] \backslash (A \bigcup \{j\}), n_C)$;
$(\nu)_{i \in A} \longleftarrow \mathbb{A}(I)$;
$B \longleftarrow B || (i, \nu_i)_{i \in A, \nu_i \in [1, n_C]}$;
;
;
$b \xleftarrow{\$} \{0, 1\}$;
$B \longleftarrow B || (j, \nu_b)$;
;
$r \longleftarrow \mathsf{tally}(B)$;
$b' \longleftarrow \mathbb{A}(r)$;
if $\nu_0, \nu_1 \in [1, n_C] \wedge b' == b$ **then**
  return 1 **else return** 0;

Figure 47: Definition of privacy, λ is the security parameter, n_T the number of talliers, t the threshold, C_t the set of the corrupted talliers, n the number of voters, n_A the number of corrupted voters, n_C the number of voting options (excluding abstention) and $\mathscr{B}$ the distribution.

Game 5: In this game, the adversary no longer has access to the roster Π^R at line 4.

Just as in the transition to Game 5 in the proof of coercion-resistance, we can construct $\mathbb{A}_5$ which interacts with $\mathbb{A}_4$ by simulating Game 4, so that $|S_5 - S_4|$ is negligible.

$$|S_5 - S_4| \leq 2n_V \varepsilon_{\mathsf{PA0}}.$$

Game 6: In this game, before computing the tally, we decrypt every valid ballot sent by the adversary at lines 11, 13 and 16. If one of these ballots uses the same credential as a ballot sent by a honest voter (*i.e.* a ballot added to the board at line 12 for some (i, ν_i) with $i \in [1, n_V] \backslash A$), we abort the game and output a random bit.

This is the same transition as the one to Game 6 in the proof of coercion-resistance. Using the same arguments, we can show that $|S_6 - S_5|$ is negligible.

Game 7: In this game, we remove lines 11 and 13 so that the adversary can no longer insert its own ballots between two honest ballots. In addition, we give I to the adversary at line 16.

This transition is very similar to the transition to Game 7 in the proof of coercion-resistance. Similarly, we construct $\mathbb{A}_7$ which interacts with $\mathbb{A}_6$ by simulating Game 6. For this purpose, $\mathbb{A}_7$ gets PB and I at line 16 and creates a fake empty ballot box PB'. Then, in the kth iteration of the for loop, it gives the next entry of I to $\mathbb{A}_6$ as i, as well as the current PB'. Then, $\mathbb{A}_6$ casts some ballots and $\mathbb{A}_7$ adds the valid ones to PB'. Finally, to simulate the vote of i, $\mathbb{A}_7$ adds the next entry of PB to PB'. One the for loop has ended, $\mathbb{A}_7$ can similarly simulate line 13.

Clearly, $\mathbb{A}_7$ plays a perfect simulation of Game 6 if the result of the tally is the same. Besides, the latter can only differ if the credential of a ballot sent by $\mathbb{A}_6$ is the same as the credential of a ballot sent by some honest voter. In this case, both games abort with a random output and $\mathbb{A}_7$'s probability to win is the same as $\mathbb{A}_6$'s in Game 6. Consequently, $S_7 = S_6$.

Game 8: In this game, the adversary has no longer access to the ballot box PB at line 16 but is only given I. Using a similar argument as in the transition to Game 8 in the proof of coercion-resistance, we can show that $|S_8 - S_7|$ is negligible.

Game 9: The final game is the ideal game.

We construct an adversary $\mathbb{A}_9$ which interacts with $\mathbb{A}_8$ by simulating Game 8. For this purpose, $\mathbb{A}_9$ runs the setup and the registration honestly, by generating the secret key and the credentials. This allows $\mathbb{A}_9$ to get j, ν_0, ν_1 from $\mathbb{A}_8$, that it plays in the ideal game. Then, when given I in the ideal game, it forwards it to $\mathbb{A}_8$ which answers with a list of cast ballots $\boldsymbol{M}$ by interacting with the cast oracle. To deduce the corresponding voting options $(\nu_i)_{i \in A}$, $\mathbb{A}_9$ creates a hashmap with the keys $\{c_i; i \in A\}$, and values $(\nu_i)_{i \in A}$ (initially ϕ, for abstention). For each valid ballot cast by $\mathbb{A}_8$, $\mathbb{A}_9$ decrypts the ballot using the secret key and deduces (ν, c). Since the ballot is valid, by the soundness of the ZKP, c consists of λ bits and ν is a valid voting option. If c is a key of the hashmap, $\mathbb{A}_9$ changes the corresponding value to ν. (Otherwise, it ignores the ballot.) It plays the obtained values in Game 9 and receives the result of the tally which it forwards to $\mathbb{A}_8$. Finally, it outputs $\mathbb{A}_8$'s output. Remark that $\mathbb{A}_9$ played a perfect simulation of *Game* 8, so that $S_9 = S_8$.

□

Bibliography

[ABBT16] Roberto Araújo, Amira Barki, Solenn Brunet, and Jacques Traoré. Remote Electronic Voting Can Be Efficient, Verifiable and Coercion-Resistant. In *Financial Cryptography and Data Security - FC'16*. Springer, 2016.

[Adi08] Ben Adida. Helios: Web-based Open-Audit Voting. In *USENIX Security Symposium*. USENIX, 2008.

[AFG+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-Preserving Signatures and Commitments to Group Elements. In *CRYPTO'10*. Springer, 2010.

[AFT08] Roberto Araújo, Sébastien Foulle, and Jacques Traoré. A practical and secure coercion-resistant scheme for remote elections. In *Frontiers of Electronic Voting*. Schloss Dagstuhl, 2008.

[AGM+13] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptogr. Eng.*, 3(2):111–128, 2013.

[AMM22] Ferran Alborch, Ramiro Martínez, and Paz Morillo. R-LWE-Based Distributed Key Generation and Threshold Decryption. *Mathematics*, 10(5):728, 2022.

[ARR+10] Roberto Araújo, Narjes Ben Rajeb, Riadh Robbana, Jacques Traoré, and Souheib Yousfi. Towards Practical and Secure Coercion-Resistant Electronic Elections. In *Cryptology and Network Security - CANS'10*. Springer, 2010.

[AT13] Roberto Araújo and Jacques Traoré. A Practical Coercion Resistant Voting Scheme Revisited. In *E-Voting and Identify - VoteID'13*. Springer, 2013.

[AW07] Ben Adida and Douglas Wikström. How to Shuffle in Public. In *4th Theory of Cryptography Conference, TCC'07*. Springer, 2007.

[Bat68] Kenneth E. Batcher. Sorting Networks and Their Applications. In *Spring Joint Computer Conference*, American Federation of Information Processing Societies - AFIPS'68. ACM, 1968.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction. In *Symposium on Principles of Distributed Computing - PODC'89*. ACM, 1989.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More. In *Symposium on Security and Privacy S&P'18*. IEEE, 2018.

[BBE+13] Josh Benaloh, Michael D. Byrne, Bryce Eakin, Philip T. Kortum, Neal McBurnett, Olivier Pereira, Philip B. Stark, Dan S. Wallach, Gail Fisher, Julian Montoya, Michelle Parker, and Michael Winn. STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. In *Electronic Voting Technology Workshop / Workshop on Trustworthy Elections - EVT/WOTE'13*. USENIX, 2013.

[BBMP21] Sevdenur Baloglu, Sergiu Bursuc, Sjouke Mauw, and Jun Pang. Provably Improving Election Verifiability in Belenios. In *International Conference for Electronic Voting - E-Vote-ID'21*. Springer, 2021.

[BCC+22] Mikael Bougon, Hervé Chabanne, Véronique Cortier, Alexandre Debant, Emmanuelle Dottax, Jannik Dreier, Pierrick Gaudry, and Mathieu Turuani. Themis: An On-Site Voting System with Systematic Cast-as-intended Verification and Partial Accountability. In *Conference on Computer and Communications Security - CCS'22*. ACM, 2022.

[BCG+15a] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs. In *Symposium on Security and Privacy - S&P'15*. IEEE, 2015.

[BCG+15b] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions. In *Symposium on Security and Privacy - S&P'15*. IEEE, 2015.

[BCK+14] Fabrice Benhamouda, Jan Camenisch, Stephan Krenn, Vadim Lyubashevsky, and Gregory Neven. Better Zero-Knowledge Proofs for Lattice Encryption and Their Application to Group Signatures. In *ASIACRYPT'14*. Springer, 2014.

[BCP+11] David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. Adapting helios for provable ballot privacy. In *ESORICS'11*. Springer, 2011.

[BDG+13] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In *Foundations of Security Analysis and Design - FOSAD'13, Tutorial Lectures*. Springer, 2013.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *CRYPTO'98*. Springer, 1998.

[Ben87] Josh Daniel Cohen Benaloh. *Verifiable secret-ballot elections*. PhD thesis, Yale University, 1987.

[Ben06] Josh Benaloh. Simple verifiable elections. In *Electronic Voting Technology Workshop - EVT'06*. USENIX, 2006.

[BFPV11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on Randomizable Ciphertexts. In *Public Key Cryptography - PKC'11*. Springer, 2011.

[BGdMM05] Lucas Ballard, Matthew Green, Breno de Medeiros, and Fabian Monrose. Correlation-Resistant Storage via Keyword-Searchable Encryption. http://eprint.iacr.org/2005/417, 2005. Last updated: 22-11-2005.

[BGG+18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold Cryptosystems from Threshold Fully Homomorphic Encryption. In *CRYPTO'18*. Springer, 2018.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *Symposium on Theory of Computing - STOC'88*. ACM, 1988.

[BJO+22] Carsten Baum, Robin Jadoul, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. Feta: Efficient Threshold Designated-Verifier Zero-Knowledge Proofs. In *Conference on Computer and Communications Security - CCS'22*, 2022.

[BK82] Richard P. Brent and H. T. Kung. A Regular Layout for Parallel Adders. *IEEE Trans. Computers*, 31(3):260–264, 1982.

[BL10] Michel Balinski and Rida Laraki. *Majority Judgment: Measuring Ranking and Electing*. MIT Press, 2010.

[BMN+09] Josh Benaloh, Tal Moran, Lee Naish, Kim Ramchen, and Vanessa Teague. Shufflesum: coercion-resistant verifiable tallying for STV voting. *IEEE Trans. Inf. Forensics Secur.*, 4(4):685–698, 2009.

[BMR07] Jens-Matthias Bohli, Jörn Müller-Quade, and Stefan Röhrich. Bingo Voting: Secure and Coercion-Free Voting Using a Trusted Random Number Generator. In *E-Voting and Identity - VOTE-ID'07*. Springer, 2007.

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In *Conference on Computer and Communications Security - CCS'16*. ACM, 2016.

[BMZ19] James Bartusek, Fermi Ma, and Mark Zhandry. The Distinction Between Fixed and Random Generators in Group-Based Assumptions. In *CRYPTO'19*. Springer, 2019.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In *ASIACRYPT'12*. Springer, 2012.

[BS99] Mihir Bellare and Amit Sahai. Non-malleable Encryption: Equivalence between Two Notions, and an Indistinguishability-Based Characterization. In *CRYPTO '99*. Springer, 1999.

[BT94] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Symposium on Theory of Computing - STOC'94*. ACM, 1994.

[Can00] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Paper 2000/067, 2000. Last edited: 12-02-2020.

[Can01] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Symposium on Foundations of Computer Science - FOCS'01*. IEEE, 2001.

[CC18] Pyrros Chaidos and Geoffroy Couteau. Efficient Designated-Verifier Non-interactive Zero-Knowledge Proofs of Knowledge. In *EUROCRYPT'18*. Springer, 2018.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Symposium on Theory of Computing - STOC'88*. ACM, 1988.

[CCFG16] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme. In *Conference on Computer and Communications Security - CCS'16*. ACM, 2016.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A Simpler Variant of Universally Composable Security for Standard Multiparty Computation. In *CRYPTO'15*. Springer, 2015.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a Secure Voting System. In *Symposium on Security and Privacy - S&P'08*. IEEE, 2008.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomorphic encryption from TFHE. In *ASIACRYPT'19*. Springer, 2019.

[CDDW18] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, and Bogdan Warinschi. Machine-Checked Proofs for Electronic Voting: Privacy and Verifiability for Belenios. In *Computer Security Foundations Symposium, CSF'18*. IEEE, 2018.

[CDG22] Véronique Cortier, Alexandre Debant, and Pierrick Gaudry. A privacy attack on the Swiss Post e-voting system. In *Real World Crypto Symposium- RWC'22*. IACR, 2022.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty Computation from Threshold Homomorphic Encryption. In *EUROCRYPT'01*. Springer, 2001.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO'94*. Springer, 1994.

[CEC+08] David Chaum, Aleksander Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan T. Sherman, and Poorvi L. Vora. Scantegrity: End-to-end voter-verifiable optical-scan voting. *IEEE Secur. Priv.*, 6(3):40–46, 2008.

[CFL19] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. BeleniosVS: Secrecy and Verifiability Against a Corrupted Voting Device. In *Computer Security Foundations Symposium - CSF'19*. IEEE, 2019.

[CGG19] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. Belenios: A simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning*. Springer, 2019.

[CGGI14] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election Verifiability for Helios under Weaker Trust Assumptions. In *ESORICS'14*. Springer, 2014.

[CGGI16a] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A Homomorphic LWE Based E-voting Scheme. In *Post-Quantum Cryptography - 7th International Workshop, PQCrypto'16*. Springer, 2016.

[CGGI16b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *ASIACRYPT'16*, 2016.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[CGK+16] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. SoK: Verifiability Notions for E-Voting Protocols. In *Symposium on Security and Privacy - S&P'16*. IEEE, 2016.

[CGY20] Véronique Cortier, Pierrick Gaudry, and Quentin Yang. How to fake zero-knowledge proofs, again. In *The International Conference for Electronic Voting - E-Vote-Id'20*, 2020.

[CGY21] Véronique Cortier, Pierrick Gaudry, and Quentin Yang. A toolbox for verifiable tally-hiding e-voting systems. Cryptology ePrint Archive, Paper 2021/491, 2021. https://eprint.iacr.org/2021/491.

[CGY22a] Véronique Cortier, Pierrick Gaudry, and Quentin Yang. A Toolbox for Verifiable Tally-Hiding E-Voting Systems. In *ESORICS'22*. Springer, 2022.

[CGY22b] Véronique Cortier, Pierrick Gaudry, and Quentin Yang. Is the JCJ voting system really coercion-resistant? Cryptology ePrint Archive, Paper 2022/430, 2022. https://eprint.iacr.org/2022/430.

[CH11] Jeremy Clark and Urs Hengartner. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In *Financial Cryptography and Data Security - FC'11*. Springer, 2011.

[Cha81] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[CKLM12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable Proof Systems and Applications. In *EUROCRYPT'12*. Springer, 2012.

[CL18] Véronique Cortier and Joseph Lallemand. Voting: You Can't Have Privacy without Individual Verifiability. In *Conference on Computer and Communications Security - CCS'18*. ACM, 2018.

[CLOT21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In *ASIACRYPT'21*. Springer, 2021.

[CLW20] Véronique Cortier, Joseph Lallemand, and Bogdan Warinschi. Fifty Shades of Ballot Privacy: Privacy against a Malicious Board. In *Computer Security Foundations Symposium - CSF'20*. IEEE, 2020.

[Con85] Nicolas de Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Paris : Imprimerie Royale, 1785.

[CPP13] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. Election Verifiability or Ballot Privacy: Do We Need to Choose? In *ESORICS'13*. Springer, 2013.

[CPP17] Geoffroy Couteau, Thomas Peters, and David Pointcheval. Removing the Strong RSA Assumption from Arguments over the Integers. In *EUROCRYPT'17*, 2017.

[CPST18] Sébastien Canard, David Pointcheval, Quentin Santos, and Jacques Traoré. Practical Strategy-Resistant Privacy-Preserving Elections. In *ESORICS'18*. Springer, 2018.

[CS98] Ronald Cramer and Victor Shoup. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In *CRYPTO'98*. Springer, 1998.

[CS14] Chris Culnane and Steve A. Schneider. A Peered Bulletin Board for Robust Use in Verifiable Voting Systems. In *Computer Security Foundations Symposium - CSF'14*. IEEE, 2014.

[CYLR18] Marwa Chaieb, Souheib Yousfi, Pascal Lafourcade, and Riadh Robbana. Verify-Your-Vote: A Verifiable Blockchain-Based Online Voting Protocol. In *European, Mediterranean, and Middle Eastern Conference on Information Systems - EMCIS'18*. Springer, 2018.

[CZZ+16] Nikos Chondros, Bingsheng Zhang, Thomas Zacharias, Panos Diamantopoulos, Stathis Maneas, Christos Patsonakis, Alex Delis, Aggelos Kiayias, and Mema Roussopoulos. D-DEMOS: A Distributed, End-to-End Verifiable, Internet Voting System. In *International Conference on Distributed Computing Systems - ICDCS'16*. IEEE, 2016.

[Dav22] David Mestel and Johannes Müller and Pascal Reisert. How Efficient are Replay Attacks against Vote Privacy? A Formal Quantitative Analysis. In *Computer Security Foundations Symposium - CSF'22*. IEEE, 2022.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In *ASIACRYPT'02*. Springer, 2002.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation. In *Theory of Cryptography Conference - TCC'06*. Springer, 2006.

[DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir Transformation in the Quantum Random-Oracle Model. In *CRYPTO'19*. Springer, 2019.

[DH22] Alexandre Debant and Lucca Hirschi. Reversing, breaking, and fixing the french legislative election e-voting protocol. *IACR Cryptol. ePrint Arch.*, 2022.

[DJN10] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *Int. J. Inf. Sec.*, 9(6):371–385, 2010.

[DKR06] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Computer Security Foundations Workshop - CSFW'06*. IEEE, 2006.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *J. Comput. Secur.*, 17(4):435–487, 2009.

[DLP22] Julien Devevey, Benoît Libert, and Thomas Peters. Rational Modular Encoding in the DCR Setting: Non-interactive Range Proofs and Paillier-Based Naor-Yung in the Standard Model. In *Public-Key Cryptography - PKC'22*. Springer, 2022.

[dMPQ09] Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios. In *Electronic Voting Technology Workshop / Workshop on Trustworthy Elections - EVT/WOTE'09*. USENIX, 2009.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption. In *CRYPTO'03*. Springer, 2003.

[dPLNS17] Rafaël del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. Practical Quantum-Safe Voting from Lattices. In *Conference on Computer and Communications Security - CCS'17*. ACM, 2017.

[DPP22a] Henri Devillez, Olivier Pereira, and Thomas Peters. How to Verifiably Encrypt Many Bits for an Election? In *ESORICS'22*. Springer, 2022.

[DPP22b] Henri Devillez, Olivier Pereira, and Thomas Peters. Traceable receipt-free encryption. In *ASIACRYPT'22*. Springer, 2022.

[ESLL19] Muhammed F. Esgin, Ron Steinfeld, Joseph K. Liu, and Dongxi Liu. Lattice-Based Zero-Knowledge Proofs: New Techniques for Shorter and Faster Constructions and Applications. In *CRYPTO'19*. Springer, 2019.

[ESWV22] Piret Ehin, Mihkel Solvak, Jan Willemson, and Priit Vinkel. Internet voting in Estonia 2005-2019: Evidence from eleven elections. *Gov. Inf. Q.*, 39(4):101718, 2022.

[FLM11] Marc Fischlin, Benoît Libert, and Mark Manulis. Non-interactive and Re-usable Universally Composable String Commitments with Adaptive Security. In *ASIACRYPT'11*. Springer, 2011.

[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6), 1962.

[FPS00] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing Decryption in the Context of Voting or Lotteries. In *Financial Cryptography - FC'00*. Springer, 2000.

[FS86] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86*. Springer, 1986.

[FS01] Jun Furukawa and Kazue Sako. An Efficient Scheme for Proving a Shuffle. In *CRYPTO'01*. Springer, 2001.

[Fuc11] Georg Fuchsbauer. Commuting Signatures and Verifiable Encryption. In *EUROCRYPT'11*. Springer, 2011.

[Geh81] William V. Gehrlein. The expected probability of condorcet's paradox. *Economics Letters*, 7(1):33–37, 1981.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Symposium on Theory of Computing - STOC'09*. ACM, 2009.

[GGP15] David Galindo, Sandra Guasch, and Jordi Puiggali. 2015 neuchâtel's cast-as-intended verification mechanism. In *E-Voting and Identity - VoteID'15*. Springer, 2015.

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *EUROCRYPT'99*. Springer, 1999.

[Gjø11] Kristian Gjøsteen. The norwegian internet voting protocol. In *E-Voting and Identity - VoteID'11*. Springer, 2011.

[GL17] Rajeev Goré and Ekaterina Lebedeva. Simulating STV Hand-Counting by Computers Considered Harmful. In *International Joint Conference in Electronic Voting - E-Vote-ID'16*. Springer, 2017.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect Non-interactive Zero Knowledge for NP. In *EUROCRYPT'06*. Springer, 2006.

[Gro05] Jens Groth. Non-interactive Zero-Knowledge Arguments for Voting. In *Applied Cryptography and Network Security - ACNS'05*, 2005.

[Gro16] Jens Groth. On the Size of Pairing-Based Non-interactive Arguments. In *EUROCRYPT'16*. Springer, 2016.

[GS07] Jens Groth and Amit Sahai. Efficient Non-interactive Proof Systems for Bilinear Groups. https://eprint.iacr.org/2007/155, 2007. Last updated: 11-04-2016.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT'08*. Springer, 2008.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed Output Delivery Comes Free in Honest Majority MPC. In *CRYPTO'20*. Springer, 2020.

[gua22] 'People's primary' backs Christiane Taubira as unity candidate of French left. theguardian.com, 2022. Accessed: 16-03-23.

[Hel] Helios Voting. https://vote.heliosvoting.org/. Accessed: 03-03-2023.

[HHHH18] Friorik P. Hjalmarsson, Gunnlaugur K. Hreioarsson, Mohammad Hamdaqa, and Gísli Hjálmtýsson. Blockchain-Based E-Voting System. In *International Conference on Cloud Computing, CLOUD'18*. IEEE, 2018.

[HHK+21] Fabian Hertel, Nicolas Huber, Jonas Kittelberger, Ralf Kuesters, Julian Liedtke, and Daniel Rausch. Extending the Tally-Hiding Ordinos System: Implementations for Borda, Hare-Niemeyer, Condorcet, and Instant-Runoff Voting. In *International Conference for Electronic Voting - E-Vote-ID'21*. University of Tartu Press, 2021.

[Hir10] Martin Hirt. Receipt-Free K-out-of-L Voting Based on ElGamal Encryption. In *Towards Trustworthy Elections, New Directions in Electronic Voting*. Springer, 2010.

[HKK+22] Nicolas Huber, Ralf Küsters, Toomas Krips, Julian Liedtke, Johannes Müller, Daniel Rausch, Pascal Reisert, and Andreas Vogt. Kryvos: Publicly Tally-Hiding Verifiable E-Voting. In *Conference on Computer and Communications Security - CCS'22*. ACM, 2022.

[HKLD17] Rolf Haenni, Reto E. Koenig, Philipp Locher, and Eric Dubuis. CHVote Protocol Specification. https://eprint.iacr.org/2017/325, 2017. Last edited: 29-11-2022.

[HLPT20] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *Symposium on Security and Privacy - S&P'20*. IEEE, 2020.

[HMMP23] Thomas Haines, Rafieh Mosaheb, Johannes Müller, and Ivan Pryvalov. Sok: Secure e-voting with everlasting privacy. *Proc. Priv. Enhancing Technol.*, 2023(1):279–293, 2023.

[HMRT12] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting. In *Topics in Cryptology - CT-RSA'12*. Springer, 2012.

[HPT19] Thomas Haines, Dirk Pattinson, and Mukesh Tiwari. Verifiable Homomorphic Tallying for the Schulze Vote Counting Scheme. In *Verified Software. Theories, Tools, and Experiments - VSTTE'19*. Springer, 2019.

[HRT10] James Heather, Peter Y. A. Ryan, and Vanessa Teague. Pretty good democracy for more expressive voting schemes. In *ESORICS'10*. Springer, 2010.

[HS00] Martin Hirt and Kazue Sako. Efficient Receipt-Free Voting Based on Homomorphic Encryption. In *EUROCRYPT'00*. Springer, 2000.

[HS19] Thomas Haines and Ben Smyth. Surveying definitions of coercion resistance. Cryptology ePrint Archive, Report 2019/822, 2019. https://ia.cr/2019/822.

[HSB21] Lucca Hirschi, Lara Schmid, and David A. Basin. Fixing the Achilles Heel of E-Voting: The Bulletin Board. In *Computer Security Foundations Symposium - CSF'21*. IEEE, 2021.

[HT15] J. Alex Halderman and Vanessa Teague. The New South Wales iVote System: Security Failures and Verification Flaws in a Live Online Election. In *E-Voting and Identity - VoteID'15*. Springer, 2015.

[HW14] Sven Heiberg and Jan Willemson. Verifiable internet voting in estonia. In *International Conference on Electronic Voting: Verifying the Vote, EVOTE'14*. IEEE, 2014.

[IRRR17] Vincenzo Iovino, Alfredo Rial, Peter B. Rønne, and Peter Y. A. Ryan. Using selene to verify your vote in JCJ. In *Financial Cryptography and Data Security - FC'17 International Workshops*, Lecture Notes in Computer Science. Springer, 2017.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Workshop on Privacy in the Electronic Society - WPES'05*, 2005.

[JJ00] Markus Jakobsson and Ari Juels. Mix and Match: Secure Function Evaluation via Ciphertexts. In *ASIACRYPT'00*. Springer, 2000.

[JSI96] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated Verifier Proofs and Their Applications. In *EUROCRYPT'96*. Springer, 1996.

[Key] BlueKrypt Cryptographic Key Length Recommendation. https://www.keylength.com/. Accessed: 23-02-2023.

[KKW06] Aggelos Kiayias, Michael Korman, and David Walluck. An Internet Voting System Supporting User Privacy. In *Computer Security Applications Conference - ACSAC'06*. IEEE, 2006.

[KLM+20] Ralf Küsters, Julian Liedtke, Johannes Müller, Daniel Rausch, and Andreas Vogt. Ordinos: A Verifiable Tally-Hiding E-Voting System. In *European Symposium on Security and Privacy - EuroS&P'20*. IEEE, 2020.

[KLO+19] Michael Kraitsberg, Yehuda Lindell, Valery Osheter, Nigel P Smart, and Younes Talibi Alaoui. Adding distributed decryption and key generation to a ring-LWE based CCA encryption scheme. In *Australasian Conference on Information Security and Privacy - ACISP'19*. Springer, 2019.

[KMST16] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A Lightweight Verifiable Remote Voting System. In *Computer Security Foundations Symposium - CSF'16*. IEEE, 2016.

[Knu73] Donald Knuth. *The Art Of Computer Programming, vol. 3: Sorting And Searching*. Addison-Wesley, 1973.

[Kra03] Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols. In *CRYPTO'03*. Springer, 2003.

[KRS10] Steve Kremer, Mark Ryan, and Ben Smyth. Election Verifiability in Electronic Voting Protocols. In *ESORICS'10*. Springer, 2010.

[KTV10a] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. A Game-Based Definition of Coercion-Resistance and its Applications. In *Computer Security Foundations Symposium - CSF'10*. IEEE, 2010.

[KTV10b] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In *Conference on Computer and Communications Security - CCS'10*. ACM, 2010.

[KTV11] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. In *Symposium on Security and Privacy - S&P'11*. IEEE, 2011.

[KTV12] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash Attacks on the Verifiability of E-Voting Systems. In *Symposium on Security and Privacy - S&P'12*. IEEE, 2012.

[KTV14] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Formal analysis of chaumian mix nets with randomized partial checking. In *Symposium on Security and Privacy - S&P'14*, pages 343–358. IEEE, 2014.

[KY02] Aggelos Kiayias and Moti Yung. Self-tallying Elections and Perfect Ballot Secrecy. In *International Workshop on Practice and Theory in Public Key Cryptosystems - PKC'02*. Springer, 2002.

[KZZ15] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-End Verifiable Elections in the Standard Model. In *EUROCRYPT'15*. Springer, 2015.

[LHK16] Philipp Locher, Rolf Haenni, and Reto E. Koenig. Coercion-Resistant Internet Voting with Everlasting Privacy. In *Financial Cryptography and Data Security - FC'16 International Workshops*. Springer, 2016.

[Lin11] Yehuda Lindell. Highly-Efficient Universally-Composable Commitments Based on the DDH Assumption. In *EUROCRYPT'11*, 2011.

[Lip03] Helger Lipmaa. On Diophantine Complexity and Statistical Zero-Knowledge Arguments. In *ASIACRYPT'03*. Springer, 2003.

[LNP22] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon. Lattice-Based Zero-Knowledge Proofs and Applications: Shorter, Simpler, and More General. In *CRYPTO'22*. Springer, 2022.

[LP11] Yehuda Lindell and Benny Pinkas. Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer. In *Theory of Cryptography Conference - TCC*. Springer, 2011.

[LPJY13] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Linearly Homomorphic Structure-Preserving Signatures and Their Applications. In *CRYPTO'13*. Springer, 2013.

[LQT20] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. VoteAgain: A scalable coercion-resistant voting system. In *USENIX Security Symposium*. USENIX, 2020.

[LS12] Mark Lindeman and Philip B. Stark. A Gentle Introduction to Risk-Limiting Audits. *IEEE Secur. Priv.*, 10(5):42–49, 2012.

[LT13] Helger Lipmaa and Tomas Toft. Secure Equality and Greater-Than Tests with Sublinear Online Complexity. In *International Colloquium on Automata, Languages, and Programming - ICALP'13*. Springer, 2013.

[LWB05] Helger Lipmaa, Guilin Wang, and Feng Bao. Designated Verifier Signature Schemes: Attacks, New Security Notions and a New Construction. In *International Colloquium on Automata Languages and Programming - ICALP'05*. Springer, 2005.

[Mee69] B. L. Meek. Une nouvelle approche du scrutin transférable. *Mathématiques et Sciences humaines*, 25:13 – 23, 1969.

[MF21] Arno Mittelbach and Marc Fischlin. *The Theory of Hash Functions and Random Oracles - An Approach to Modern Cryptography*. Information Security and Cryptography. Springer, 2021.

[mie] Mieux voter. https://mieuxvoter.fr/qui-sommes-nous. Accessed: 16-03-2023.

[MM06] Ülle Madise and Tarvi Martens. E-voting in Estonia 2005. The first Practice of Country-wide binding Internet Voting in the World. In *International Workshop in Electronic Voting - EVOTE'06*. GI, 2006.

[MN06] Tal Moran and Moni Naor. Receipt-Free Universally-Verifiable Voting with Everlasting Privacy. In *CRYPTO'06*. Springer, 2006.

[MPT20] Eleanor McMurtry, Olivier Pereira, and Vanessa Teague. When Is a Test Not a Proof? In *ESORICS'20*. Springer, 2020.

[MSH17] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. A Smart Contract for Boardroom Voting with Maximum Voter Privacy. In *Financial Cryptography and Data Security - FC'17*. Springer, 2017.

[MZR+21] Karola Marky, Marie-Laure Zollinger, Peter Roenne, Peter YA Ryan, Tim Grube, and Kai Kunze. Investigating usability and user experience of individually verifiable internet voting schemes. *Transactions on Computer-Human Interaction - TOCHI'21*, 28(5):1–36, 2021.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Conference on Computer and Communications Security - CCS'01*. ACM, 2001.

[Nie02] Jesper Buus Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In *CRYPTO'02*. Springer, 2002.

[Nie03] Jesper Buus Nielsen. *On Protocol Security in the Cryptographic Model*. PhD thesis, University of Aarhus, 2003.

[NS10] Takashi Nishide and Kouichi Sakurai. Distributed Paillier Cryptosystem without Trusted Dealer. In *International Workshop on Information Security Applications - WISA'10*. Springer, 2010.

[NSW19] NSWEC – Election results. NSW Electoral Commision, pastvtr.elections.nsw.gov.au, 2019. Accessed: 2020-08-05.

[NV12] Stephan Neumann and Melanie Volkamer. Civitas and the real world: problems and solutions from a practical point of view. In *International Conference on Availability, Reliability and Security - ARES'12*. IEEE, 2012.

[Ord] Ordinos Extension (E-Vote-ID 2021). https://github.com/JulianLiedtke/ordinos. Accessed: 27-02-2023.

[Pai66] Claude Pair. Sur des algorithmes pour des problèmes de cheminement dans les graphes finis. In *Théorie des graphes, journées internationales d'études*. Dunod (Paris), 1966.

[Pai99] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT'99*. Springer, 1999.

[Ped91a] Torben P. Pedersen. A Threshold Cryptosystem without a Trusted Party (Extended Abstract). In *EUROCRYPT'91*. Springer, 1991.

[Ped91b] Torben P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91*. Springer, 1991.

[PG12] Jordi Puigalli and Sandra Guasch. Cast-as-Intended Verification in Norway. In *International Conference on Electronic Voting - EVOTE'12*. GI, 2012.

[Pol60] Maurice Pollack. The Maximum Capacity through a Network. *Operations Research*, 8(5):733–736, 1960.

[PS96] David Pointcheval and Jacques Stern. Security Proofs for Signature Schemes. In *EUROCRYPT'96*. Springer, 1996.

[RCPT19] Kim Ramchen, Chris Culnane, Olivier Pereira, and Vanessa Teague. Universally Verifiable MPC and IRV Ballot Counting. In *Financial Cryptography and Data Security - FC'19*. Springer, 2019.

[RRI16] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In *Financial Cryptography and Data Security - FC'16 International Workshops, BITCOIN, VOTING, and WAHC*. Springer, 2016.

[RS62] J Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6(1):64–94, 1962.

[RS06] Peter Y. A. Ryan and Steve A. Schneider. Prêt à voter with re-encryption mixes. In *ESORICS'06*. Springer, 2006.

[Rya05] Peter Y. A. Ryan. A variant of the chaum voter-verifiable scheme. In *Workshop on Issues in the Theory of in the Theory of Security - WITS'05*. ACM, 2005.

[SBWP03] Ron Steinfeld, Laurence Bull, Huaxiong Wang, and Josef Pieprzyk. Universal Designated-Verifier Signatures. In *ASIACRYPT'03*. Springer, 2003.

[Sch80] Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4):701–717, 1980.

[Sch89] Claus-Peter Schnorr. Efficient Identification and Signatures for Smart Cards. In *CRYPTO'89*. Springer, 1989.

[Sha71] Daniel Shanks. Class number, a theory of factorization and genera. *Proc. symp. Pure Math.*, 20:415 – 440, 1071.

[SHKS11] Michael Schläpfer, Rolf Haenni, Reto E. Koenig, and Oliver Spycher. Efficient Vote Authorization in Coercion-Resistant Internet Voting. In *E-Voting and Identity - VoteID'11*. Springer, 2011.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT'97*. Springer, 1997.

[SK95] Kazue Sako and Joe Kilian. Receipt-Free Mix-Type Voting Scheme - A Practical Solution to the Implementation of a Voting Booth. In *EUROCRYPT'95*. Springer, 1995.

[SKHS11] Oliver Spycher, Reto E. Koenig, Rolf Haenni, and Michael Schläpfer. A New Approach towards Coercion-Resistant Remote E-Voting in Linear Time. In *Financial Cryptography and Data Security - FC'11*. Springer, 2011.

[SKHS12] Oliver Spycher, Reto E. Koenig, Rolf Haenni, and Michael Schläpfer. Achieving Meaningful Efficiency in Coercion-Resistant, Verifiable Internet Voting. In *International Conference on Electronic Voting - EVOTE'12*. GI, 2012.

[SKM03] Shahrokh Saeednia, Steve Kremer, and Olivier Markowitch. An Efficient Strong Designated Verifier Signature Scheme. In *Information Security and Cryptology - ICISC'03*. Springer, 2003.

[Smi07] Warren D. Smith. Three voting protocols: Threeballot, vav, and twin. In *Electronic Voting Technology Workshop - EVT'07*. USENIX, 2007.

[sou22] Source code of prototype implementation of condorcet-schulze. Available at `https://gitlab.inria.fr/gaudry/THproto`, 2022.

[ST04] Berry Schoenmakers and Pim Tuyls. Practical Two-Party Computation Based on the Conditional Gate. In *ASIACRYPT'04*. Springer, 2004.

[Swi] swisspost-evoting. `https://gitlab.com/swisspost-evoting`. Accessed: 11-04-2023.

[Tid87] T Nicolaus Tideman. Independence of clones as a criterion for voting rules. *Social Choice and Welfare*, 4(3):185–206, 1987.

[tre] Receipt-free trenc implementation. `https://github.com/receiptfreevoting/trenc_implem`. Accessed: 17-04-2023.

[TW10] Björn Terelius and Douglas Wikström. Proofs of Restricted Shuffles. In *Progress in Cryptology - AFRICACRYPT'10*. Springer, 2010.

[Ubu12] Ubuntu IRC council position. `https://lists.ubuntu.com/archives/ubuntu-irc/2012-May/001538.html`, 2012. Accessed: 04/01/2022.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully Homomorphic Encryption over the Integers. In *EUROCRYPT'10*. Springer, 2010.

[Ver] Open Verificatum. `https://www.verificatum.org/`. Accessed: 07-02-2023.

[Vot23] Online votes make up two-thirds of Reform, less than third of EKRE votes. news.err.ee, 2023. Accessed: 13-04-2023.

[War62] Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1), 1962.

[WB08] Roland Wen and Richard Buckland. Mix and Test Counting in Preferential Electoral Systems. Technical report, University of New South Wales, 2008.

[Wik04] Douglas Wikström. A Universally Composable Mix-Net. In *Theory of Cryptography Conference - TCC'04*, Lecture Notes in Computer Science. Springer, 2004.

[Wik05] Douglas Wikström. A Sender Verifiable Mix-Net and a New Proof of a Shuffle. In Bimal K. Roy, editor, *ASIACRYPT'05*. Springer, 2005.

[Wik09] Douglas Wikström. A commitment-consistent proof of a shuffle. In *Australasian Conference on Information Security and Privacy - ACISP'09*. Springer, 2009.

[Yao86] Andrew Chi-Chih Yao. How to Generate and Exchange Secrets (Extended Abstract). In *Symposium on Foundations of Computer Science - FOCS'86*. IEEE, 1986.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation - EUROSAM'79*. Springer, 1979.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates. In *EUROCRYPT'15*, Lecture Notes in Computer Science. Springer, 2015.

Résumé

Le vote est la base de toutes les démocraties : c'est la clef de voûte qui légitime les actions d'un gouvernement, et il est décrit par Lyndon B. Johnson, le 58ème président des États-Unis, comme « l'instrument le plus puissant jamais conçu par l'homme pour briser l'injustice ». Récemment, le vote électronique est apparu comme un moyen d'améliorer les systèmes de vote existants. Tout d'abord, le vote électronique peut permettre aux citoyens de voter par internet, qui est plus accessible qu'un bureau de vote dépendant de la liste électorale. Cela pourrait représenter une alternative intéressante au vote par correspondance ou par procuration, en particulier pour les personnes expatriées, les handicapées et les étudiants. Deuxièmement, le vote électronique peut être une alternative nécessaire en cas de confinement de longue durée, comme cela a été le cas lors de la pandémie de COVID. Enfin, l'utilisation d'ordinateurs peut faciliter le décompte des voix, et contribuer à limiter l'utilisation de bulletins de vote en papier, qui ont un impact négatif sur l'environnement.

En raison de ces avantages, le vote par Internet a été utilisé pour des élections politiques dans plusieurs pays, comme par exemple l'Australie, le Canada, la France, la Norvège et la Suisse. D'autres formes de vote électronique, basées sur des machines à voter, ont été utilisées, par exemple, au Bangladesh, au Brésil, en Namibie, en Nouvelle-Zélande, au Pakistan, en Corée du Sud et aux États-Unis. L'exemple le plus remarquable est l'Estonie, où le vote par internet est possible depuis 2005, et où la proportion de votants par internet est passée d'un faible pourcentage à 51% en 2023 [ESWV22, Vot23]. D'autre part d'autre part, certains pays, tels que l'Allemagne, l'Italie et le Royaume-Uni, ont interdit les machines à voter, estimant qu'elles peuvent être sujettes à la fraude ou qu'un électeur doit comprendre toutes les étapes de son vote, même sans connaissances techniques.

L'une des principales raisons de se méfier du vote électronique est le risque d'une attaque informatique : en exploitant une vulnérabilité, un pirate peut être en mesure d'interrompre le service, de récupérer les bulletins choisis par les électeurs ou de truquer le résultat de l'élection, ce qui mettrait en péril la souveraineté du pays et remettrait en cause la légitimité des élus. C'est pourquoi il est extrêmement important que le système de vote garantisse la confidentialité et la vérifiabilité. Intuitivement, le *secret du vote* est atteint si personne ne peut connaître le choix d'un électeur donné. Quant à *vérifiabilité*, elle est souvent décomposée en plusieurs propriétés, à savoir la vérifiabilité individuelle, la vérifiabilité universelle et l'éligibilité. Informellement, l'éligibilité stipule que seuls les personnes éligibles peuvent voter, et qu'au maximum un bulletin est compté par personne. La vérifiabilité individuelle signifie qu'un votant est en mesure de vérifier que le bulletin de vote a bien été déposé dans l'urne t qu'il contient bien l'option de vote choisie (par exemple, le nom du candidat). Enfin, la vérifiabilité universelle indique que n'importe qui peut vérifier que le résultat de l'élection est conforme à l'urne. Pour obtenir ces propriétés, il est habituel de faire plusieurs *hypothèses de confiance*. Par exemple, dans le cas du vote sur papier, on suppose que l'urne est sécurisée, de sorte que personne ne peut malicieusement retirer ou ajouter des bulletins, ni consulter le contenu d'un bulletin spécifique. En outre, il est généralement difficile de garantir plusieurs propriétés simultanément. Par exemple, déclarer que Kim Jong-un est le vainqueur respecterait parfaitement le sevret du vote, puisque personne ne donneraient aucune information sur ses préférences. Au contraire, le vote à main levée pourrait garantir la vérifiabilité, mais pas le secret du vote.

En vote électronique, il existe plusieurs moyens d'assurer la confidentialité et la vérifiabilité. En ce qui concerne la confidentialité, la principale stratégie consiste à *chiffrer* les bulletins de vote. Ainsi, si le serveur est compromis, l'attaquant - également appelé *adversaire* - ne peut récupérer que des données chiffrées, qui ne permettent pas de savoir quel électeur a choisi quel candidat (ou, plus généralement, quelle option de vote). Pour la vérifiabilité universelle, la principale primitive cryptographique est la *preuve à divulgation nulle* (on utilisera l'abréviation ZKP, pour Zero Knowledge Proof), qui permet de prouver que le protocole de décompte, utilisé pour calculer le résultat à partir des bulletins chiffrés, a été correctement exécuté. Comme son nom l'indique, une ZKP ne révèle rien sur les secrets utilisés pour produire la preuve ; ainsi, aucune information autre que le résultat n'est révélée, et le secret du vote est préservé.

Le chiffrement et les ZKP sont très courants en vote électronique ; cependant, des systèmes tels que sElect [KMST16] ne s'appuient sur les ZKP pour la vérifiabilité, et les systèmes basés sur des urnes au dépouillement autonome (*e.g.*, [KY02]) ne s'appuient pas sur le chiffrement pour le secret du vote. Des exemples de systèmes de vote académiques qui combinent chiffrement et ZKP sont Adder [KKW06], Helios [Adi08, dMPQ09] et Belenios [CGG19], qui sont très similaires dans leur conception. Cependant, il existe de nombreuses autres propositions. Par exemple, des schémas tels que D-Demos [CZZ+16] et BeleniosVS [CFL19] peuvent garantir la confidentialité, même contre un appareil de vote malicieux ; et des systèmes tels que [CPP13] permettent d'obtenir un secret du vote intemporel (voir [HMMP23] pour une étude sur le sujet). D'autres systèmes, tels que le Prêt-à-Voter [Rya05, RS06], ThreeBallot [Smi07] et Scantegrity [CEC+08], se concentrent sur le vote papier. Lorsque du vote papier est utilisé, le résultat peut être calculé à partir des bulletins physiques ou électroniques ; des exemples de systèmes qui comptent les bulletins électroniques sont STAR-Vote [BBE+13] et Bingo Voting [BMR07]. L'intérêt de disposer à la fois de bulletins électroniques et de bulletins papier est qu'une technique connue sous le nom de *audit de limitation des risques* [LS12] peut être utilisée pour réduire le risque d'erreur ou de manipulation.

Bien que le secret du vote et la vérifiabilité universelle soient deux exigences fondamentales en matière de sécurité dans le cadre du vote électronique, elles ne sont pas ne sont pas considérées comme suffisantes pour des élections à fort enjeu. Une première difficulté est liée à la vérifiabilité individuelle : étant donné que les bulletins de vote publiés dans l'urne sont chiffrés, il est difficile pour les électeurs de savoir si leur bulletin contient effectivement l'option de vote souhaitée. Cela peut poser un problème, par exemple si l'appareil de vote est compromis par un logiciel malveillant. Pour pallier cette éventualité, il existe plusieurs stratégies permettant d'assurer une propriété appelée *cast-as-intended verification*. Une approche populaire est basée sur les codes de retour : chaque option de vote est associée à un code secret, qui est unique pour chaque électeur. À un moment donné après la phase de vote, l'électeur reçoit un code de retour calculé à partir du bulletin chiffré, et le compare au code attendu. Si les codes correspondent, l'électeur est convaincu que le contenu fu bulletin chiffré n'a pas été modifié. C'est l'approche adoptée, par exemple, par [HRT10] et CHVote [HKLD17]. Les codes de retour ont notamment été utilisés en Norvège [Gjø11, PG12] et en Suisse [GGP15], où le vote par internet a été autorisé de 2003 à 2019 et reprendra en juin 2023.

En dehors des codes de retour, une autre stratégie populaire est le défi de Benaloh [Ben06], qui est utilisé dans Helios. Une alternative récente au est proposée dans Themis [BCC+22]. En Estonie, cependant, la vérification est effectuée grâce à un serveur tiers, que l'électeur peut interroger à l'aide d'un reçu cryptographique [HW14]. Il existe d'autres approches ; par exemple, Selene [RRI16] est basée sur des données de suivi. Pour une comparaison et une catégorisation plus exhaustive, voir [MZR+21].

Vote sans reçu et résistance à la coercition

Outre la vérification de l'intention de vote, une deuxième difficulté, liée au secret du vote, est le risque de l'*achat de vote*. En effet, dans un système de vote classique tel qu'Helios, l'électeur produit un bulletin chiffré qui contient son choix. Or, en utilisant un algorithme de vote *ad hoc*, l'électeur peut produire un bulletin dont il connaît l'aléatoire utilisé pour le chiffrement. Cet aléa peut être utilisé comme *reçu* pour convaincre un acheteur de vote que le bulletin chiffre un choix spécifique. Pour tenir compte de cette menace, la notion de *vote sans reçu* (receipt-freeness) a été proposée [BT94].

Il existe plusieurs notions de receipt-freeness dans la littérature. Intuitivement, un système de vote est *sans reçu* si l'électeur ne peut pas convaincre un tiers qu'il a voté d'une certaine manière, même s'il est prêt à renoncer au secret du vote ou s'il suit un certain jeu d'instructions. Pour parvenir à l'absence de reçu tout en préservant la vérifiabilité, il existe deux approches principales. Tout d'abord, le paradigme du *revote silencieux* (deniable revoting) consiste à permettre aux électeurs de revoter. Lors d'un revote, le vote précédent est annulé, mais un observateur externe n'est pas en mesure de dire si un bulletin donné est annulé ou non. Ainsi, même si l'électeur prouve qu'il a voté d'une manière spécifique, l'acheteur du vote n'aurait aucune garantie que le vote n'a pas été annulé par un revote. Des exemples basés sur ce paradigme sont, par exemple, [LHK16] et VoteAgain [LQT20].

Une autre approche est basée sur le *rechiffrement*, où les électeurs ne peuvent pas directement déposer leur bulletin dans l'urne publique. Au lieu de cela, le bulletin est envoyé à un *serveur de rechiffrement*, auquel on fait confiance pour ce qui concerne la receipt-freeness. Le serveur rechiffre le bulletin de manière à ce qu'il devienne indistinguable d'un bulletin aléatoire. Ainsi, même si le bulletin a été créé de manière malveillante par l'électeur (ou s'il a été donné par l'acheteur du vote), il n'est plus possible de savoir si le bulletin rechiffré contient l'option de vote voulue. Néanmoins, la vérifiabilité individuelle est toujours assurée, ce qui signifie que le votant a la garantie que le contenu de son bulletin n'a pas été modifié. Afin que cette garantie ne puisse être transmise à l'acheteur de vote, [Hir10] propose d'utiliser des ZKP à vérifieur désigné (DVZKP) [JJ00], qui ne peuvent convaincre que le votant. Cependant, l'utilisation des DVZKP nécessite une mise en place qui soulève des problèmes pratiques. Plus tard, la nécessité de DVZKP a été levée grâce aux couplages bilinéaires, qui introduisent la possibilité de rechiffrer des signatures [BFPV11]. L'idée a été développée dans BeleniosRF [CCFG16], qui fournit également une définition moderne de l'absence de reçu.

Une menace liée à l'achat de vote est celle de la *coercition*. Un attaquant, le coerciteur, demande à un électeur de voter d'une manière spécifique, en utilisant une menace ou une récompense. Par rapport à l'absence de reçu, la résistance à la coercition suppose un adversaire plus fort, qui peut être actif pendant la phase de vote et demander au votant de lui donner son matériel de vote pour pouvoir voter à sa place. Lorsqu'une solution de vote électronique est utilisée sans contre-mesure contre la coercition, le coerciteur peut contraindre un plus grand nombre d'électeurs ou s'assurer - grâce au mécanisme de vérifiabilité - que les électeurs contraints ont bien obéi. En Estonie, le principal moyen d'atténuer la coercition est de permettre aux électeurs de revoter, afin qu'ils puissent d'abord obéir au coerciteur et ensuite revoter avec l'option de vote souhaitée, lorsqu'ils disposent d'un moment d'intimité. Bien que le revote soit une contre-mesure intuitive contre la coercition, cela suppose que l'électeur est capable de revoter *après* le coerciteur, ce qui n'est pas nécessairement justifié car ce dernier peut attendre le dernier moment pour voter. En outre, si le coerciteur demande à l'électeur son matériel de vote, ce dernier ne peut pas savoir quand le coerciteur va l'utiliser.

La principale solution académique pour prévenir la coercition est le protocole JCJ, proposé

dans [JCJ05], qui formalise également la notion de résistance à la coercition. L'idée est qu'un électeur est capable de donner un faux matériel de vote au coerciteur. Ce dernier peut émettre un bulletin de vote avec le matériel donné, et le bulletin sera ajouté à l'urne indépendamment de la validité du matériel de vote utilisé. Cependant, les bulletins de vote émis avec un mauvais matériel de vote sont retirés après la phase de vote. La principale propriété de sécurité du protocole JCJ est que le coerciteur n'est pas en mesure de distinguer un vrai matériel de vote d'un faux, ou de dire si un bulletin de vote donné a été retiré ou non. Dans la littérature, de nombreux schémas ultérieurs ont été basés sur le paradigme du faux matériel de vote, et peuvent être considérés comme des itérations permettant d'améliorer certains points du protocole JCJ. L'exemple le plus connu est Civitas [CCM08], qui propose une phase d'enregistrement explicite. D'autres contributions, par exemple, se sont concentrées sur l'amélioration de la gestion du matériel de vote [CH11, NV12]. Enfin, de nombreuses propositions visent à améliorer l'efficacité du protocole : voir, par exemple, les schémas de Spycher *et al.* [SKHS11, SHKS11, SKHS12] et les schémas d'Araújo *et al.* [AFT08, ARR+10, AT13, ABBT16].

La prévention des attaques à l'italienne

Une menace majeure qui n'est pas prise en compte dans la résistance à la coercition ou dans l'absence de reçu est celle des attaques *à l'italienne*, qui sont basées sur les informations disponibles dans le résultat du décompte. En effet, l'une des principales stratégies pour déterminer le résultat de l'élection à partir des bulletins chiffrés consiste à utiliser un *mixnet* [Cha81], qui révèle la liste de toutes les options de vote choisies par les électeurs, mais dans un ordre aléatoire. En général, cela donne plus d'informations que le simple résultat (typiquement, le nom du ou des vainqueurs, le nombre de bulletins dépouillés et le nombre de bulletins exprimés), et ces informations peuvent être utilisées par un coerciteur pour décider si un votant placé sous la contrainte a obéi ou non. Par exemple, dans le cas du vote préférentiel, un choix peut être n'importe quelle permutation des candidats, de sorte qu'il peut y avoir beaucoup plus de choix possibles qu'il n'y a d'électeurs. En Australie, les élections législatives de 2019 en Nouvelle-Galles du Sud, qui ont utilisé un système de vote préférentiel connu sous le nom de vote unique transférable (STV), ont compté plusieurs centaines de candidats [NSW19]. Dans une telle situation, il est possible de demander à l'électeur de classer d'abord le candidat préféré du coerciteur, puis d'utiliser une permutation très spécifique et improbable des autres candidats, par exemple en alternant plusieurs partis opposés. Si l'électeur n'obéit pas, il est fort probable que personne d'autre ne soumettra un tel bulletin, si bien que le coerciteur pourra déduire, en observant le résultat du décompte où toutes les permutations choisies sont révélées, que le votant a désobéi. De cette manière, le coerciteur peut contraindre un grand nombre d'électeurs et savoir exactement lesquels ont obéi.

Il n'existe pas beaucoup de contre-mesures contre les attaques italiennes dans la littérature ; l'approche la plus prometteuse repose sur le *tally-hiding*. Dans un système partiellement tally-hiding, le protocole de dépouillement fuite certaines informations supplémentaires, mais pas nécessairement toutes les options de vote choisies par les électeurs. Dans [RCPT19], un protocole MPC est proposé afin de dépouiller un système de vote nommé IRV (un cas spécifique de STV où il n'y a qu'un seul vainqueur) est proposé. Ce protocole permet de calculer un décompte IRV sans révéler toutes les permutations choisies par les électeurs. Cependant, il révèle certaines informations sur le déroulement du protocole. Pour les méthodes de Condorcet, qui sont plusieurs méthodes de comptage s'appliquant au vote préférentiel respectant un critère introduit par Condorcet [Con85], la principale stratégie consiste à représenter le choix d'un électeur sous la forme d'une matrice, de sorte que les bulletins peuvent être additionnés. C'est, par exemple, l'approche proposée dans [HPT19]. Un autre exemple qui utilise une forme de tally-hiding par-

tielle est Shuffle-sum [BMN+09], qui vise à atténuer le risque d'une attaque italienne dans STV en cachant les informations les plus cruciales.

Dans Kryvos [HKK+22], une solution basée sur le tally-hiding *public* est proposée. L'idée est que le public n'a accès qu'au résultat de l'élection, tandis que les dépouilleurs apprennent plus d'informations. Le principal problème de cette approche est qu'elle ne protège pas les électeurs contre une coercition de la part d'un dépouilleur.

Il est également possible de concevoir un protocole complètement tally hiding. C'est ce qui a été fait, par exemple, dans [CPST18] ; toutefois, cette solution a été proposée pour le jugement majoritaire, pour lequel il est possible d'utiliser un décompte homomorphe, de sorte que le risque d'une attaque à l'italienne est faible. Parallèlement à cette thèse, le travail indépendant d'Ordinos [KLM+20] a été proposé pour réaliser un tally hiding complet. Ordinos a été étendu dans [HHK+21] pour couvrir diverses fonctions de comptage, y compris certaines variantes de la méthode de Condorcet. La solution proposée par Ordinos est coûteuse pour l'électeur et ne lui permet pas de classer plusieurs candidats à égalité, ce qui est restrictif dans le contexte du vote de Condorcet. Dans les deux propositions, la solution proposée repose sur le calcul multi-partie (MPC) basé sur le schéma de chiffrement de Paillier, qui, comparé au plus populaire chiffrement d'ElGamal, a la propriété d'être *additivement* homomorphe. Cependant, le chiffrement de Paillier nécissite une longueur de la clef beaucoup plus importante, de sorte que le calcul d'un chiffrement est plus coûteux que pour le chiffrement d'ElGamal. Par rapport à une solution de type Helios, le coût du chiffrement d'un bulletin peut être supérieur de plusieurs ordres de grandeur lorsque l'on utilise le chiffrement de Paillier, ce qui soulève des questions pratiques. En outre, comme le système de chiffrement de Paillier n'est pas aussi largement utilisé que celui d'ElGamal, il n'existe pas de bibliothèque aussi bien étudiée et largement déployée. Cela est d'autant plus préjudiciable en vote électronique que nous avons besoin de deux bibliothèques, l'une du côté du serveur et l'autre du côté de l'électeur.

Les preuves de sécurité

L'utilisation de primitives cryptographiques bien étudiées, telles que le chiffrement et le ZKP, ne suffit pas à garantir la sécurité d'un protocole. Idéalement, ce dernier devrait être analysé à plusieurs niveaux d'abstraction. Premièrement, les concepteurs du protocole devraient fournir une preuve de sécurité cryptographique (ou formelle) ; ensuite, la spécification du protocole devrait faire l'objet d'un audit pour s'assurer qu'il n'y a pas de vulnérabilités ; enfin, l'implémentation devrait également être auditée, par exemple dans le cadre d'un programme de prime aux bogues. Pour que ces preuves et ces audits soient significatifs, la communauté académique recommande que les spécifications du protocole soient publiques, afin que l'ensemble de la communauté puisse l'analyser. En Suisse, l'audit public du système de vote de La Poste Suisse, dont les spécifications sont disponibles à [Swi], a permis de détecter certaines vulnérabilités avant la publication, comme décrit dans [HLPT20] et [CDG22]. En revanche, il a été révélé que le système de vote utilisé en Australie présentait certains problèmes de sécurité [HT15], de même que celui utilisé en France [DH22]. Ces problèmes auraient pu être évités si la communauté du vote électronique avait eu l'occasion d'auditer ces systèmes *avant* leur déploiement. Ces échecs montrent qu'il n'est pas facile de concevoir un système de vote électronique sécurisé, et encore moins d'évaluer sa sécurité. C'est pourquoi il est habituel de fournir une preuve calculatoire ou formelle que les propriétés souhaitées sont vérifiées.

Dans une preuve calculatoire, l'adversaire est modélisé comme une *machine de Turing*, qui a une puissance de calcul limitée (polynomiale) mais qui peut effectuer des calculs arbitraires. La principale stratégie consiste à présenter une réduction polynomiale d'un problème connu, tel que

la factorisation des entiers ou le problème du logarithme discret. En d'autres termes, une preuve cryptographique est une preuve mathématique qui stipule que, si une propriété de sécurité est violée, il existe une machine de Turing explicite polynomiale (c'est-à-dire un algorithme efficace) qui résout un problème calculatoire considéré comme difficile. Pour un problème bien étudié tel que le logarithme discret, cela signifie que la propriété de sécurité est vérifiée.

Dans une preuve formelle, un modèle mathématique et symbolique est conçu pour représenter le protocole et la propriété de sécurité souhaitée. souhaitée. Dans un tel modèle, les primitives cryptographiques utilisées, ainsi que les actions possibles que l'adversaire peut effectuer, sont idéalisées, par exemple à l'aide de règles de réécriture ou de théories équationnelles. Une fois le modèle créé, la preuve proprement dite consiste à démontrer que la propriété de sécurité peut ou ne peut pas être violée. Généralement, une preuve formelle est obtenue grâce à un outil entièrement automatique ou interactif basé sur des techniques de déduction. Par rapport à une preuve calculatoire qui repose sur une hypothèse de calcul bien étudiée, une preuve formelle suppose que la cryptographie est parfaite et ne peut être violée. Cependant, elle peut couvrir plus de scénarios d'attaque.

Avant de fournir une preuve, une étape clef consiste à modéliser les propriétés de sécurité souhaitées et à en donner une définition formelle. Or, les définitions des notions de sécurité dans le domaine du vote électronique ne sont pas stabilisées. Par exemple, l'une des premières définitions pour le secret du vote a été donnée par Benaloh [Ben87], et a été utilisée ou étendue dans divers travaux ultérieurs (*e.g.*, [KZZ15, CL18]). Cependant, cette définition comporte plusieurs limites, de sorte que d'autres définitions ont été proposées. En particulier, la notion de *ballot privacy* [BCP+11, BPW12] a convergé vers la définition BPRIV, donnée dans [BCG+15b]. Cette définition a été étendue dans [CLW20], pour modéliser modéliser la présence d'une urne malveillante.

Pour la receipt-freeness, deux définitions modernes peuvent être trouvées dans [CCFG16] et [KZZ15]. Par rapport à la définition de Kiayias *et al.*, la définition de Chaidos *et al.* ne tient pas compte du mécanisme de vérifiabilité individuel. Cependant, la définition de Kiayias *et al.* ne prend pas en compte le fait que le votant peut construire son bulletin de manière malveillante, ce qui est restrictif. Dans [DPP22b], une version modifiée de la définition de Chaidos *et al.* a été proposée, où le protocole d'inscription n'est plus pertinent. Cela a été fait dans le but d'atteindre l'absence de reçu (presque) indépendamment du reste du protocole, ce qui permet une analyse de sécurité plus modulaire et rend le système de vote plus adaptable.

Pour la résistance à la coercition, la principale définition académique est celle de [JCJ05], qui reste une référence sur le sujet. Intuitivement, cette définition compare un jeu réel à un jeu idéal : dans le jeu réel, l'adversaire observe le protocole réel ; dans le jeu idéal, l'adversaire n'a pas d'autres informations que celles contenues dans le résultat final ; dans les deux jeux, le but de l'adversaire est de deviner si le votant a obéi ou non. Cette comparaison est faite car, en observant le résultat, on peut obtenir des informations sur le choix de l'électeur soumis à la contrainte : c'est l'idée qui sous-tend les attaques à l'italienne. Dans [HS19], on remarque que la définition de JCJ est défectueuse et ne peut pas être réalisée par un système de vote avec une urne publique. La raison principale est que le jeu idéal ne fournit aucune information sur l'urne. Par conséquent, en observant la taille de l'urne dans le jeu réel et en la comparant au résultat de l'élection (qui indique le nombre de bulletins dépouillés), on peut inférer si le bulletin soumis avec le matériel de vote du votant a été retiré ou non. Pour corriger ce défaut, [HS19] propose de modifier le jeu idéal et d'ajouter l'information sur la taille de l'urne. Cependant, la définition qui résulte reste incomplète, car elle ne prend pas en compte les revotes.

En effet, dans le contexte de la résistance à la coercition, il est naturel de permettre le revote, qui peut constituer une première contre-mesure face aux influences implicites d'un membre de la

famille ou d'un employeur. Supposons, par exemple, qu'une petite-fille explique à son grand-père comment voter en ligne. Pour ce faire, elle lui demande de s'identifier à la plateforme de vote et de procéder étape par étape, tandis qu'elle reste derrière lui pour clarifier chaque étape. Dans ce scénario, le grand-père peut se sentir obligé de choisir le parti démocratique alors qu'il aurait préféré choisir le parti républicain. Lorsque le revote est autorisé, le grand-père peut choisir n'importe quel candidat (ou même le candidat suggéré par la petite-fille). Par la suite, lorsque la petite-fille n'est plus là, il peut revoter avec son choix personnel. Autre exemple : un employé est encouragé à voter *au travail*, en utilisant un appareil qui peut être surveillé par l'employeur. Pour éviter tout conflit, le votant peut être tenté de voter dans un premier temps pour le parti conservateur, puis revoter pour le parti travailliste lorsqu'il est chez lui. Il est donc important que les définitions de la résistance à la coercition prennent en compte le revote.

Nos contributions

1. Nous proposons une boîte à outils MPC basée sur le schéma de chiffrement ElGamal, qui peut être utilisé pour réaliser un tally hiding complet.

Notre boîte à outils, présentée au chapitre 5, est basée sur la primitive de porte conditionnelle [ST04]. Elle offre une alternative intéressante au cadre de Paillier, qui permet notamment de diminuer les coûts du côté du votant sans trop détériorer la compléxité du côté du serveur. Cette boîte à outils fournit de nombreux protocoles MPC pour réaliser diverses opérations sur des données cryptées, telles que des opérations arithmétiques et des comparaisons, mais aussi des opérations plus complexes comme celles liées au tri. De plus, nous proposons plusieurs compromis entre le coût calculatoire et le nombre de communications, qui peuvent être déployés pour atténuer le fait que le chiffrement d'ElGamal nécessite plus de communications. Dans le chapitre 6, nous appliquons notre boîte à outils pour concevoir un protocole complètement tally hiding pour les méthodes de Condorcet-Schulze, STV, le jugement majoritaire et la méthode D'Hondt. Dans le cas de la méthode de Condorcet, nous avons découvert une violation de la vie privée dans la solution de [HS19], qui se produit lorsqu'un électeur donne le même rang à deux candidats.

Dans la section 6.1.2, nous proposons une nouvelle façon pour l'électeur de soumettre un bulletin de vote pour le vote de Condorcet, qui permet le vote blanc et est compatible avec le dépouillement homomorphique. Pour le jugement majoritaire, nous avons remarqué un défaut dans la solution proposée dans [CPST18], qui utilise une heuristique connue sous le nom de *majority gauge*. En effet, cette heuristique ne garantit pas de produire systématiquement un résultat. Enfin, nous avons également découvert un problème avec la solution proposée dans [KLM+20], qui a été conçue pour révéler les noms des s candidats ayant reçu le plus de votes, où s est un paramètre quelconque (par exemple, le nombre de sièges). En effet, en cas d'égalité, il est possible que leur solution produise en fait plus de s gagnants. Nous proposons un moyen non intrusif d'inclure un mécanisme de départage dans la solution proposée par Ordinos ; cela préserve la propriété de tally hiding et ne détériore pas l'efficacité.

Nous prouvons la sécurité de notre boîte à outils dans le cadre de sécurité de [CCL15], qui est une variante plus simple du cadredriciel universellement composable de [Can01] (en bref, nous utilisons l'abréviation SUC pour désigner ce cadriciel). Pour ce faire, nous avons modifié le protocole de la porte conditionnelle et prouvé sa sécurité SUC du protocole modifié en Section 4.4. Comme le cadre SUC fournit un *théorème de composition*, la sécurité SUC de la primitive principale peut être utilisée pour prouver les propriétés de sécurité souhaitées, telles que la confidentialité et la vérifiabilité, ce qui est fait dans la section 6.6.

2. Nous dévoilons une fuite dans le schéma JCJ qui peut compromettre sa résistance à la coercition lorsque le revote est autorisé.

Lorsque le revote est autorisé, nous avons découvert que les informations supplémentaires révélées pendant la phase de décompte du protocole JCJ peuvent être exploitées par le coerciteur pour déduire le comportement de l'électeur soumis à la contrainte, en utilisant l'inférence bayésienne. Plus précisément, nous avons identifié la nature exacte de la fuite dans le protocole JCJ : Par rapport au pur résultat de l'élection, qui contiendrait des informations sur le nombre total de bulletins retirés, le protocole JCJ divulgue le nombre de votants ayant revoté k fois, pour tout $k \geq 1$, ainsi que le nombre de bulletins qui utilisent un matériel de vote invalide. Pour évaluer l'impact de cette fuite d'informations, nous avons utilisé le cadre formel de [KTV10a] qui donne une définition quantitative de la résistance à la coercition. Dans ce cadre, il est possible de comparer le *niveau de coercition* du protocole réel à celui du protocole idéal, qui ne souffrirait d'aucune fuite. En utilisant ce cadre, nous proposons plusieurs scénarios réalistes où la différence entre les niveaux de coercition (idéal et réel) n'est pas négligeable.

L'une des raisons pour lesquelles le défaut du protocole JCJ n'a pas été remarqué jusqu'à présent est peut-être que la définition de JCJ ne tient pas en compte du revote. En outre, on sait que, dans un système de type JCJ, un nombre imprévisible de bulletins doit être retiré pendant le décompte. Autrement, le coerciteur s'apercevrait que le bulletin déposé avec le matériel de vote fourni par l'électeur a été retiré. C'est pourquoi il est nécessaire de modéliser la présence de bulletins utilisant un matériel de vote invalide, mais qui ne sont pas des bulletins soumis par le coerciteur : ils sont appelés bulletins *fictifs*. Dans la définition JCJ, ces bulletins sont censés provenir des électeurs honnêtes, qui doivent pour cela sacrifier leur propre vote. Cette modélisation n'est pas réaliste et ne permet pas d'envisager une situation où des bulletins fictifs supplémentaires seraient déposés, par exemple par une tierse partie qui n'est pas un votant éligible. Pour ces raisons, nous avons conçu une nouvelle définition de la résistance à la coercition, qui tient mieux compte de la présence de bulletins fictifs et de la possibilité de revoter.

Comme le protocole JCJ ne vérifie pas notre définition de la résistance à la coercition, nous proposons CHide, une variante du protocole JCJ qui utilise la boîte à outils afin d'empêcher la fuite présente dans le schéma JCJ. Ceci est fait dans le chapitre 8, et montre que notre définition de résistance à la coercition peut être satisfaite en pratique. Pour rendre le protocole pratique pour des paramètres réalistes, nous avons conçu une nouvelle phase de nettoyage qui s'appuie sur le tri, et qui et qui est plus robuste vis-à-vis du nombre de bulletins soumis. Nous prouvons que la confidentialité, la vérifiabilité et la résistance à la coercition sont atteintes par CHide sous les mêmes hypothèses de confiance que JCJ.

3. Nous étudions la notion d'absence de reçu et proposons une solution qui peut constituer un premier pas pratique vers la résistance à la coercition.

En collaboration avec Henri Devillez, Olivier Pereira et Thomas Peters, nous proposons une nouvelle définition de la notion de receipt-freeness qui ne fait aucune hypothèse sur la phase d'enregistrement ou le mécanisme d'éligibilité. Par rapport à la définition de [KZZ15], notre définition prend en compte le fait que l'électeur peut utiliser *n'importe quel* algorithme pour produire son bulletin de vote, y compris un algorithme qui peut être fourni par l'acheteur de votes. Par rapport à la définition de [CCFG16], notre définition permet à l'adversaire de donner une instruction *quelconque* à l'électeur, et pas seulement un bulletin chiffré. En outre, elle tient compte du fait que l'électeur peut se voir remettre un reçu pendant la phase de vote, en raison du mécanisme de vérifiabilité individuelle. Dans l'ensemble, notre définition de l'absence de reçu

est plus proche de l'intuition de l'achat de votes, et la réalisation de cette définition peut être un premier pas vers la résistance à la coercition. En outre, sur la base des travaux antérieurs de [DPP22b], nous proposons une stratégie modulaire qui permet de construire un système de vote sans reçu, en fournissant un ensemble de conditions faciles à vérifier concernant le schéma de chiffrement, le protocole de comptage et la phase de vote.

Cela rend la réalisation de l'absence de reçu plus modulaire et plus indépendante des spécificités du protocole. Nous fournissons également un nouveau schéma de chiffrement qui satisfait aux propriétés requises par notre stratégie, de sorte qu'il peut être instancié. Par rapport au schéma proposé dans [DPP22b], ce nouveau schéma supporte les preuves 0/1 (c'est-à-dire qu'il est possible de prouver que le bulletin de vote chiffre un message d'une forme spécifique), ce qui est extrêmement intéressant dans le contexte du vote électronique. En outre, la génération des paramètres nécessaires à ce schéma de chiffrement peut être faite à l'aide d'aléa publiques, ce qui signifie que nous avons besoin de moins d'hypothèses de confiance. Enfin, par rapport au schéma proposé dans [CCFG16], le nôtre n'est pas limité au chiffrement de petites chaînes de bits.