Université Paris-Saclay

Adding security properties to postal voting

Léo LOUISTISSERAND

Master thesis, under the supervision of Véronique Cortier and Pierrick Gaudry

March 2022 - August 2022

Contents

1 Introduction **2**
1.1 Properties . . . 2
1.1.1 Confidentiality properties . . . 2
1.1.2 Verifiability properties . . . 2
1.1.3 Other properties . . . 3
1.2 State of the art . . . 3
1.2.1 Postal voting for French people living abroad . . . 3
1.2.2 STROBE . . . 3
1.3 Our contribution . . . 4
1.3.1 Vote&Check . . . 4
1.3.2 Condor . . . 4
1.3.3 Formal verification . . . 5

2 Proposed protocols **5**
2.1 Existing protocols . . . 5
2.1.1 STROBE . . . 5
2.1.2 JCJ . . . 7
2.1.3 BeleniosVS . . . 7
2.2 Vote&Check . . . 8
2.2.1 Description of the protocol . . . 8
2.2.2 Comments on the design . . . 10
2.2.3 Security properties . . . 10
2.2.4 Possible attacks . . . 11
2.3 Condor . . . 11
2.3.1 Description of the protocol . . . 11
2.3.2 Comments on the design . . . 13
2.3.3 Security properties . . . 13
2.4 Comparative table . . . 13

3 Formal verification **14**
3.1 The ProVerif tool . . . 14
3.2 Condor's model . . . 15
3.3 Security properties . . . 17
3.3.1 Verifiability . . . 17
3.3.2 Privacy . . . 19

4 Conclusion **19**

1 Introduction

Elections are held in many different contexts: in politics, associations, enterprises, trade unions... Each context has its specific constraints, its requirements, its means and its stakes. There are several modalities for voting: at the ballot box, show of hands, remote e-voting, postal voting, etc. Voting at the ballot box offers all the guarantees of confidentiality and absence of fraud, but requires the voters to be present at a polling station. This might be unsuitable, for example if there are a little number of voters spread in a large area. Internet voting is attractive because of it seemingly simple procedure. It is widely studied in the literature and many protocols exist, tending to guarantee the same security properties as ballot box voting. Internet voting is used for some elections with very high stakes, often coupled with postal voting or voting at the ballot box. Internet voting is sometimes considered immature and fragile. Thus, postal voting will remain present, although it is also very vulnerable. The latest example is the 2022 election of the leader of the Tories. The voters had the choice between internet, ballot box and postal voting, but GCHQ raised an alert short before the election and the modalities have been modified [11]. Similarly, three months before the 2017 french legislative election for French people living abroad, the government removed the possibility of remote voting, fearing a cyberattack [4]. Hence, postal voting has still a long way to go, and should therefore be secured.

1.1 Properties

To compare the different voting protocols, there are several desirable properties. We distinguish between properties related to confidentiality, those related to verifiability and some additional properties. Except from the traditional voting protocol with booths, transparent ballot boxes and a public tally, no protocol satisfy all of them. More properties are detailed in [7].

1.1.1 Confidentiality properties

Ballot secrecy The vote of every voter must remain secret. We say that a protocol respects the *ballot secrecy* if it does not reveal any information about the votes except of course the result of the election.

Coercion Resistance The purpose of the ballot secrecy is to prevent the voters from being pressured by those around them, the candidates for the election, other voters... However, it is not enough to guarantee the sincerity of the vote. A protocol is said *coercion-resistant* if it is not possible for a voter to prove to anyone else for who they voted. This property avoid vote selling.

1.1.2 Verifiability properties

The verifications properties are those that guarantee that the result of the election is correct, independently of the assumptions on the authorities. Each voter should be able to check that their vote has been taken into account, and to convince themself that only valid ballots are in the ballot box.

Cast as Intended When the protocol allows the voters to make sure that the ballots they cast contain the votes they intended to cast, we say that it respects the *cast as intended* property. For example, if encrypted ballot are used, voters should be able to verify that the encryptions are actually encrypting the right votes. One possibility in this case is that the printer joins a sheet to the ballot containing all the random numbers used for the encryption.

Recorded as Cast When the protocol allows each voter to verify that their ballot is in the ballot box and that it contains the vote that they cast, we say that it respects the *recorded as cast* property.

Tallied as recorded When the protocol allows the voter to verify that the result of the election is actually corresponding to the ballots in the ballot box, we say that it respects the *tallied as recorded* property. This can be achieved thanks to cryptography, for example using a homomorphic scheme or a mixnet, and verifiable decryption.

Eligibility verifiability The ballot box must only contain ballots coming from eligible voters, and at most one per single voter. When a protocol allows the observers to verify that this statement is true, then we say that it satisfies the *eligibility verifiability* property.

1.1.3 Other properties

Ignorability Postal voting should not be an obstacle for the voter. We say that a protocol is *ignorable* if the voters can vote as usual and still have their ballot counted, although the verification properties no longer hold. More precisely, we say that a protocol is ignorable if the voters can cast their ballots without using a computer.

Accountability The verification properties allow to detect an possible fraud. We say that a protocol is *accountable* if it is possible to prove which entity is at fault, and if it is possible for an entity to prove their innocence. This property is usually not fully satisfied.

1.2 State of the art

There exist a lot of different protocols for postal voting, for example with barcodes, with mitigated results [5], or with cryptrography. We focused on two of them, that are the postal voting protocol for French people living abroad, used during the 2022 legislative election, and STROBE, presented by Josh Benaloh in [1].

1.2.1 Postal voting for French people living abroad

French people living abroad have several possibilities to vote: there are polling stations in embassies, they can vote by internet, by proxy or by postal mail [12]. To vote by mail, people have to register with the consulate. Before the election, each voter receives every possible bulletin and three envelops of different sizes: a small ballot envelop, a medium identification envelop and a big shipping envelop. They choose one bulletin, put it in the ballot envelop and close it. Then, they put this small envelop in the identification one, close it and write their name on it. Finally, they put the medium envelop and a copy of their identity card in the shipping one, close it and send it to the consulate. The consul is in charge of the receipt of the envelops. They open the shipping envelop and write on a list the name of the voter and the time the envelop was delivered. At the end of the voting phase, the consul gives all the identifications envelops and all the copies of identification cards to the talliers that verify the authentication of the voters. If they have not voted in the polling station, their ballot envelops are put in the ballot box with the other and tallied as usual.

This protocol has several drawbacks. It is not possible for the voters to verify that their vote has been recorded properly, or at least that their envelop has been received. Since there is no verification properties, the postal service or the consul can modify the cast votes without any risk of being detected. Furthermore, the privacy is very weak: anyone who have access to an identification envelop can learn the vote of its sender by simply reading the name on it and the ballot inside.

1.2.2 STROBE

The objective of STROBE is to provide an end-to-end verifiable scheme, even if the authorities are dishonest. It relies on cryptographic properties of the ElGamal encryption scheme. The talliers run a distributed key generation protocol and publish their public key. For each voter, the printer generates two encryptions of the identity matrix, each line corresponding to a candidate. They publishes those encryptions after having shuffled the lines, so the association between an encrypted line and a candidate remains secret. The printer prints a ballot containing the list of the candidates, the hash of the shuffled matrix id and the last byte b of the hash of each encrypted line, in front of the associated candidate. Each voter chooses one of the two ballots, ticks the box corresponding to the candidate of their choice and sends it to the cast officers. The cast officers, after receiving the ballot, publish id and the byte b of the choice of the voter. They can verify that their vote has been recorded as cast. Then, the printer reveals the random numbers used to encrypt the unused ballots, to convince the voter that the encryption on the used ballots are correct and that their votes have been cast as intended. The talliers sum all the chosen encryption and verifiably decrypt the result.

STROBE guarantees that the ballot are tallied as recorded and recorded as cast even if all authorities are dishonest. Each voter knows that their vote would have been cast as intended if they used the other ballot, and thus can be convinced that it is also the case with the one they chose. The main drawback of this scheme is that event a *honest but curious* printer knows the vote of every voter. Also, the printer has access to every bulletin and can use them to vote for the abstainers.

1.3 Our contribution

Our contribution consists of the design of two postal voting protocols and the formal analysis of one of them. Both of them are fully ignorable. The first protocol, Vote&Check, is inspired by STROBE and tries to bring confidentiality while keeping the end-to-end verifiability. The second one, Condor, is inspired by the JCJ voting scheme and tries to make the previous one coercion resistant. Our formal analysis revealed an attack on the verifiability property on Condor.

1.3.1 Vote&Check

The main idea of Vote&Check is to split the printer into two entities sharing the secret used by the voter to verify their vote. Therefore, they will have to collude to break the privacy property. We assume the existence of a private and authenticated channel between the registrar and each voter.

The printer and the registrar agree on a pseudonym a for each voter. The registrar sends a and a tracking number t to the voter by the private channel. They send $(h(a), t)$ to the cast officer. The printer sends a, a voting credential c and a signature of (a, c) on a ballot to the voter by postal mail. The voter adds a number of their choice n on the ballot, ticks the box of the candidate of their choice and send the filled ballot to the cast officer. The cast officer verifies that the printer's signature of (a, c) is valid and that they received $h(a)$ from the registrar. In this case, they publish $(c \oplus t, v, n)$ on the bulletin board with $\oplus$ the exclusive or. The voter can find their ballot on the board and verify that it has been recorded as cast. The arbitrary number n avoids the clash attacks [10].

This protocol satisfies all the individual verification properties without any assumption, respects the eligibility if the registrar is honest or if the printer and the cast officer are honest, and the guarantees the confidentiality if at most one authority is dishonest. It is neither accountable nor coercion resistant. One nice feature it that it only uses very basic cryptographic primitives: xor and hash function.

1.3.2 Condor

To achieve coercion resistance, we designed a more complex voting scheme. It uses cryptographic primitives that will be described with more details in the next section. The different authorities are the registrar, the printer, the cast officer that is in charge of the ballot box and the talliers. We assume the existence of a private and authenticated channel between the registrar and each voter and between the registrar and the printer.

The talliers run a distributed key generator and publish their public key pk_T. The registrar generates for each voter a couple of keys $(sk1, pk1)$ and sends the public key with the name of the voter to the printer and the private key to the voter, using the secure channels. The printer generates another couple of keys $(sk2, pk2)$, a credential σ, two encryptions S and $\tilde{S}$ of σ under pk_T and a *designated verifier zero-knowledge proof* (DVZKP) using pk_1 and showing that S and $\tilde{S}$ encrypt the same message or that sk_1 is known. They publish S and send their material to the voter: σ, the random r used to compute $\tilde{S}$, the DVZKP, sk_2 and a ballot containing $\tilde{S}$, $PK = pk_1 \times pk_2$, a zero-knowledge proof of knowledge of r and one box for each candidate. They also publish PK. The voter verifies that their material is correct, ticks the box of their choice and send the ballot to the cast officer. The cast officer verifies that the ZKP and the key PK are valid, in this case they publish $\tilde{S}, PK$, the ZKP, an encryption of the vote v under PK, an other encryption of v under pk_T and a ZKP that the two encryptions are encrypting the same plaintext. After the voting phase, the talliers verify all ZKP and remove the lines with invalid ones. Then, they perform verifiable *plaintext equivalence tests* (PET) on the different $\tilde{S}$ to remove the duplicate ballots. All ballots are shuffled using a verifiable mixnet. After that, the talliers perform PET between the list of S and the list of $\tilde{S}$ and keep only the lines with a valid credential. Finally, they decrypt all the votes and proclaims the result.

This protocol is resistant to vote-buying: a voter knows the private key sk_1 so they can sell a random number instead of $\tilde{S}$ and a DVZKP that this fake credential is equivalent to s or that sk_1 is known. An attacker cannot distinguish between this and valid material.

1.3.3 Formal verification

Many security properties have been cited in the previous section. These properties may or may not be true depending on the assumptions made about the different agents. For example, classical postal voting verifies all of them in the case where all entities are honest. Finding the minimal assumptions requires to analyze many cases and to produce as many proofs: with n entities, there are 2^n cases to deal with. In most of these cases, the proofs will be similar, so writing them by hand is likely to be error-prone, as well as tedious. This is why we use automatic verification tools. Here, we have chosen ProVerif. We need to model a protocol to transform it into something on which ProVerif can apply logical reasoning. In doing so, we move away from reality, the challenge being to approximate it as much as possible. Some constructions (encryption, hashing, ZKP, private channels) are classical, others are not and need to be designed from scratch. We present here a model of the postal channel, on which anyone can write but only the legitimate recipient can read messages. The modeling of the Condor protocol revealed an important attack on verifiability.

2 Proposed protocols

2.1 Existing protocols

In this section, I will present three protocols that inspired us in our work. STROBE [1] is a verifiable postal voting protocol, JCJ [8] is an electronic voting protocol providing coercion resistance and BeleniosVS [6] is an electronic voting protocol that guaranties both verifiability and privacy, even if one authority is dishonest or if one of the voter's devices is corrupt.

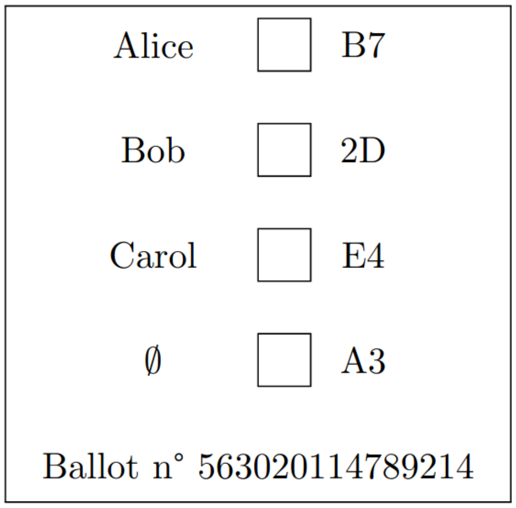
2.1.1 STROBE

The STROBE voting scheme is a postal voting scheme designed by Josh Benaloh. Its objective is to provide verifiability while staying as close as possible to traditional vote by mail. The main idea is that the printer will send two ballots to each voter who will cast one of them.

The different entities involved are the printer, the postal service, the cast officer and each voter. There is a public board where the printer and the cast officer can write any message.

Protocol STROBE uses probabilistic homomorphic encryption (for example ElGamal). For each voter, the printer encrypts the identity matrix of size one plus the number or possible votes. Each line of the encrypted matrix is hashed, only the last byte of the hash is kept as a short code. This is repeated until all short codes are different. Lines of the matrix are permuted such that the short codes increase. The ballot identifier is the hash of the permuted matrix. The permuted matrix, the short codes and the ballot identifier are published on the board, as well as zero-knowledge proofs that the matrix is the encryption of a permutation matrix. Ballots are paired and the pairings are made public. Each voter receives by postal mail two paired ballots (see Figure 1). They can verify that the bulletins are well-formed by looking if the informations on the board correspond to what they have received. Then they simply choose one of the bulletin, tick the box of their choice and send it by postal mail to the cast officer. The cast officer publishes the ballot identifier and the chosen short code, that a conscientious voter can verify. Finally, the printer multiplies all the selected encryption and decrypts the product, giving the result of the election. They also reveal the random numbers used to encrypt the unused bulletins, proving that they were well formed.

$$A = enc\left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\right) = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$
$$b_1 = h(a_{11}||a_{12}||a_{13}||a_{14}) = B7\ldots$$
$$b_2 = h(a_{21}||a_{22}||a_{23}||a_{24}) = 2D\ldots$$
$$b_3 = h(a_{31}||a_{32}||a_{33}||a_{34}) = E4\ldots$$
$$b_4 = h(a_{41}||a_{42}||a_{43}||a_{44}) = A3\ldots$$
$$A' = \begin{bmatrix} a_{21} & a_{22} & a_{23} & a_{24} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ a_{11} & a_{12} & a_{13} & a_{14} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}$$
$$id = h(A')$$
A’, the ZKP and the pairings are published on the board.


Alice ☐ B7

Bob ☐ 2D

Carol ☐ E4

∅ ☐ A3

Ballot n° 563020114789214

Figure 1: Construction of a STROBE ballot

Security properties STROBE is an *individual verifiable* scheme under no assumption. Each voter can verify that their vote has been *recorded as cast* thanks to the public information. They can verify that the ballot they did not used was correct and therefore be convinced that their vote has been *cast as intended.* The decryption is verifiable so every vote is *tallied as recorded.* The printer can illegitimately cast the ballots of the voters, but will be detected if these voters later do the verification, so STROBE respects *eligibility.* However, this protocol does not respect *confidentiality.* A *honest but curious* printer knows the vote of every voter. STROBE is not *coercion-resistant*, since a coercer can require the voting material of a voter, verify that they are valid and later that the voter did not try to fool them and to cast a ballot.

2.1.2 JCJ

The JCJ voting scheme is named after its authors Juels, Catalano and Jakobsson. It provides coercion resistance without compromising verifiability. It is a protocol for internet voting, where every voter needs a computer, not only to verify that their ballot has been counted properly, but also to cast their ballot. Moreover, each voter is supposed to have a secret key, and the associated public key is known by the registrar.

The different entities involved are the registrar, the talliers and the voters. There is a public board where anyone can write any message with authentication but where no message can be deleted. The different channels of communication are supposed to be private.

Protocol The talliers run a distributed key protocol and publish the associated public key $pk_{\mathcal{T}}$ on the board. Each voter publishes their public key pk_i. The registrar generates for each voter V_i a voting credential σ_i and sends it to the legitimate voter. They publish on the board $S_i = enc(\sigma_i, pk_{\mathcal{T}}, r_i)$ for a random r_i. The registrar also sends to V_i a *designated verifier zero-knowledge proof* that S_i is an encryption of σ_i or that the private key associated to pk_i is known to ensure the voter that the credential is valid.

During the voting phase, V_i publishes on the board an encryption of their credential $\tilde{S}_i = enc(\sigma_i, pk_{\mathcal{T}}, s_i)$, an encryption of their vote $c_i = enc(v_i, pk_{\mathcal{T}}, t_i)$ and a zero-knowledge proof of knowledge of σ_i.

During the tallying phase, the talliers verify all ZKP and discards the ballots with invalid ones. Then, they perform *plaintext equivalence tests* on the $\tilde{S}_i$ to remove all duplicates and keep at most one ballot per credential. Afterwards, the tally shuffles and re-encrypts the ballots with a verifiable mixnet, and do the same to the list of S_i. Then they perform plaintext equivalence tests between the S_i and the $\tilde{S}_i$ to remove the bulletin without a valid credential. Finally, the talliers decrypt all the remaining c_i.

Evasion strategy Intuitively, the coercion resistance is the ability of a voter to fool an attacker that tries to influence their vote. The attacker can force the voter to reveal any private information they have. The evasion strategy is what the voter can do to satisfy their coercer and still have their ballot counted the way they want. Here, the evasion strategy consist of giving the private key sk_i such that $pk_i = pk(sk_i)$, and random number $\tilde{\sigma}_i$ and a DVZKP that S_i is an encryption of $\tilde{\sigma}_i$ or that sk_i is known. The coercer cannot distinguish between this situation and the case where the voter really gives their credential.

2.1.3 BeleniosVS

BeleniosVS is an electronic voting protocol that provides privacy and verifiability if only one authority is dishonest or if one of the voter's devices is corrupt. The main idea is to re-encrypt the voter's ballot several times to make sure that nobody can deduce their votes. The primitive used, called *signatures on randomisable ciphertexts* [3] allows anyone to compute, from a signed encryption, a new encryption of the same message and a new valid signature of the new ciphertext for the same key. The different entities involved are the registrar, the voters, their voting and auditing devices, the voting server and the tallying authority. Each voter is supposed to have a password to authenticate themself to the voting server. There is a public board as usual.

Protocol The talliers run a distributed key generation protocol and publish the associated public key $pk_{\mathcal{T}}$. The registrar generates for each voter V_i a signature key sk_i and the associated verification key vk_i. They encrypt each possible vote v under $pk_{\mathcal{T}}$ and sign this encryption with sk_i. They send a voting sheet to V_i containing sk_i, the encrypted and signed votes and the random numbers used. They publish (V_i, vk_i). The voter can verify that this voting sheet is correct using their auditing machine, i.e. a computer with a scanner. Then, they scan the part of the ballot containing the encrypted and signed vote of their choice with the voting machine (an other computer with a scanner) and give their password to the voting machine. The voting machine re-encrypts the ciphertext and the signature, authenticates to the voting server and sends the re-encrypted ballot. The voting server checks that this voter has not already voted and that the signature is valid with the verification key vk_i. In this case, they re-re-encrypt the vote and the signature and publish the ballot on the board. The voter verifies that there is a ballot signed with their signature key sk_i.

Security properties BeleniosVS guaranties privacy and individual verifiability if at most one entity is dishonest or if one of the voter's devices is corrupt. Even though it uses paper-based voting material, this protocol requires a lot of computations from the voter that are mandatory, and, for this reason, is very difficult to change into a full paper version.

2.2 Vote&Check

2.2.1 Description of the protocol

The goal of this protocol is to keep the verifiability properties from STROBE while getting the privacy properties even if one entity is dishonest. The main idea is to split the voting authority in a printer and a registrar that are sharing the secret used by the voter to verify their vote.

The different entities involved are a registrar, a printer, the cast officer and every voter. The registrar, the printer and the cast officer have each private and public keys to sign messages. There is a public bulletin board where the cast officer and the registrar can write any message. We assume that there exists a private and authenticated channel between the registrar and the printer, as well as between the registrar and each voter. The channel between the printer and each voter and between each voter and the cast officer is the postal service. We suppose that when the post is honest, the message on those channels can only be read by their legitimate recipient, and that anyone can write on it. The protocol will be described in the next paragraph. It is summarised in Figure 2. The election runs in three phases.

Setup During this phase, the registrar and the printer send the material to the voters. For each voter V_i:

- The registrar creates an alias a_i by choosing a random 128-bits bitstring. They share this information with the printer. Both of them commit on this information by signing it and exchanging these signatures. The registrar publishes the voters' list and the printer verifies that it corresponds to what they have agreed on.
- The registrar creates a 128-bits verification token t_i. They send $(h(a_i), t_i)$ and a signature to the cast officer, with h a hash function. The cast officer countersigns this message and sends the signature to the registrar. The registrar also sends (a_i, t_i) and a signature to the voter V_i by the secure channel.
- The printer creates a 128-bits voting credential c_i. They send V_i their bulletin by postal mail. The bulletin contains a_i, c_i and the signature (see Figure 3).

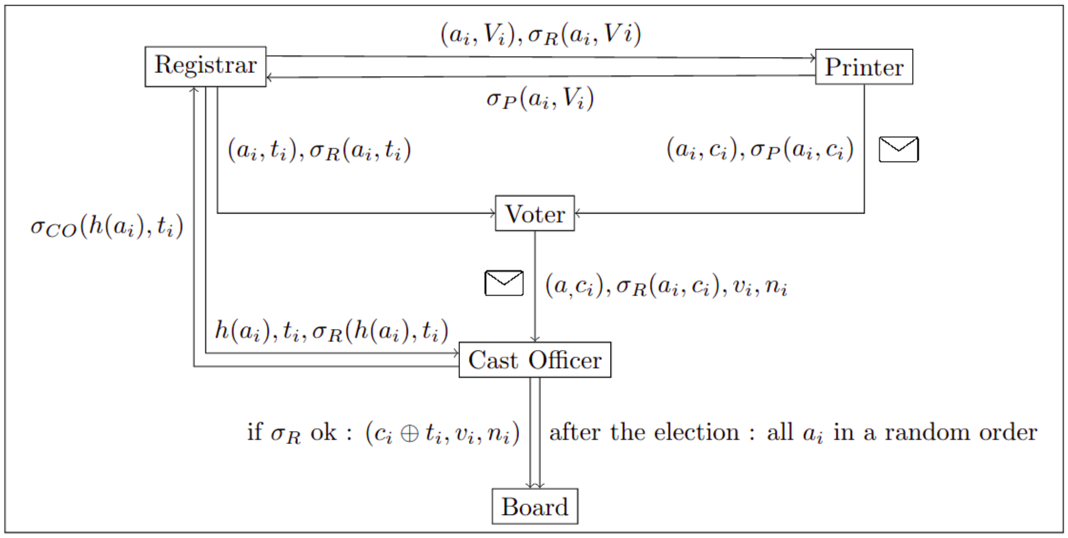

Registrar
Printer
$(a_i, V_i), \sigma_R(a_i, Vi)$
$\sigma_P(a_i, V_i)$
$(a_i, t_i), \sigma_R(a_i, t_i)$
$(a_i, c_i), \sigma_P(a_i, c_i)$
Voter
$\sigma_{CO}(h(a_i), t_i)$
$(a_, c_i), \sigma_R(a_i, c_i), v_i, n_i$
$h(a_i), t_i, \sigma_R(h(a_i), t_i)$
Cast Officer
if σ_R ok : $(c_i \oplus t_i, v_i, n_i)$
after the election : all a_i in a random order
Board

Figure 2: Description of Vote&Check

- (optional) A conscientious voter may verify that the material they received is correct by checking the signatures and that the two aliases a_i are the same. This part requires the use of a computer.

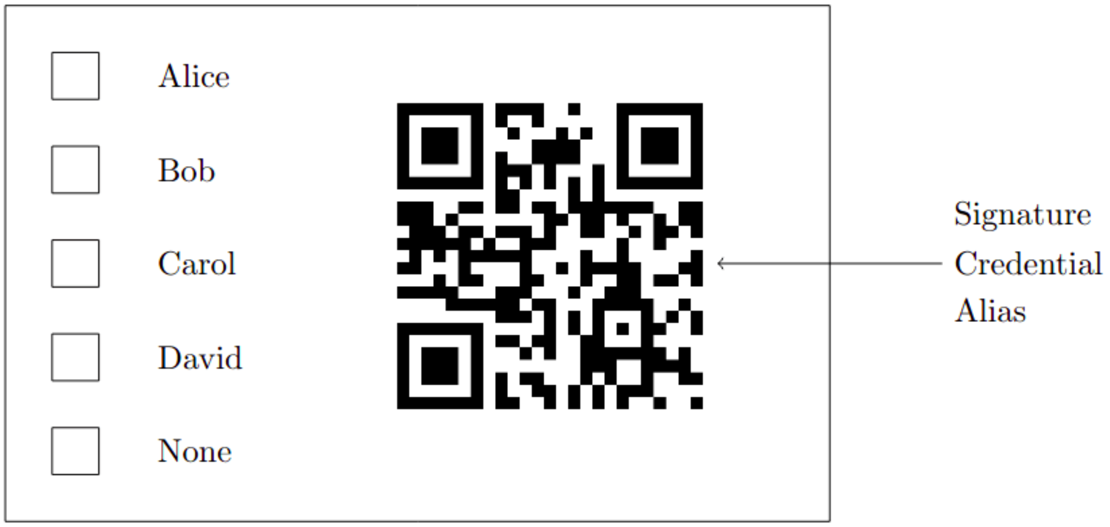


Figure 3: Ballot

Vote During this phase, voters cast their ballots by postal mail. The cast officer publishes the information that voters need to verify that their ballots have been recorded correctly.

- Each voter V_i ticks the box corresponding to their vote v_i and write on the ballot a number n_i of their choice. Then they mail their ballot to the cast officer.

- When the cast officer receives a ballot, they check that the signature is correct and that the voting credential c_i has not been used. In this case, they find t_i associated to a_i by look for the message $(h(a_i), t_i)$ sent by the registrar. If such a message does not exist, they raise an alert. Else, the ballot is valid and the cast officer publishes $(c_i \oplus t_i, v_i, n_i)$.
- (optional) A conscientious voter may verify that their vote has been correctly recorded by looking for a line starting with $c_i \oplus t_i$. Because of the size of the bitstrings, there should not be any collision so the voter can be sure that this line corresponds to their ballot. Then they only have to check that the vote v_i and the number n_i are the ones that they have chosen. During this phase, the voter needs their computer to receive t_i from the registrar, to read the results on the public board and to compute the xor.

Tally During this phase, the result of the election is computed as well as the list of the voters who indeed voted.

- Every ballot published by the cast officer is valid and votes are not encrypted so anyone can compute the result of the election by simply counting votes.
- Once all ballots have been received, the cast officer publishes the shuffled list of all received aliases a_i. From this list, the registrar finds and publishes the list of voters who have cast a ballot and the printer verify that it is correct.

2.2.2 Comments on the design

- Contrary to STROBE, the printer and the registrar are two different entities. This avoid that an *honest but curious* authority learns the vote of every voter without having to do anything.
- The number n_i that the voter V_i has to choose and write on their ballot is there to make *clash attacks* a lot riskier. If both the registrar and the printer are corrupted and collude, and if they are sure that two different voters will cast the same vote, they may send them twice the same material and use the other ballot to cast an illegitimate valid ballot. The two victims cannot realise it, even if they do all possible verifications, and only one of their votes would be counted. Thanks to the number n_i, they may notice it if they choose different numbers: the cast officer will only publish one of them since the aliases are the same, so one of the voter can verify, notice the problem and will raise an alert.
- The different signatures fail to provide total accountability but they prevent the authorities from doing anything without any risk. For example, the registrar commits to the printer on the list of the aliases so they cannot send a wrong alias to a voter.
- The cast officer has to publish a number of valid aliases a_i equal to the number of ballots they published. Their possible ways to learn valid aliases are reading the ballots that they receive or corrupting either the registrar, the printer or the post service. This means that the cast officer alone cannot vote for the abstainers. The printer can but they cannot know a priori which voter will abstain.

2.2.3 Security properties

Verifiability Vote&Check is end-to-end verifiable even if all participants are dishonest. The conscientious voter will look for a line containing their identifier $c_i \oplus t_i$ and check that their vote has been properly tallied nearby. This line must correspond to the voter because the identifier is unique. If it is not the case, for example if the registrar and the printer collude and send the same material to different voters, those voters will notice it if they choose different n_i and verify. Hence, a voter that sees a line on the bulletin board corresponding to their ballot can be convinced that their vote has been *recorded as cast.* Using only plaintext votes guarantees that the vote are *cast as intended* and *tallied as recorded.*

Eligibility The only undetectable ballot stuffing involves at least the registrar and one entity out of the printer or the cast officer. If an illegitimate ballot is recorded, either the cast officer is dishonest or received an illegitimate valid ballot. To get an illegitimate valid ballot, the printer can forge one and both the postal service and the printer can both steal legitimate voter's ballots. Furthermore, the registrar establishes the list of the voters that have actually voted, so either the registrar is corrupted or a voter that has not voted will be on this list. If a voter has been stolen their ballot or counted dishonestly, they may notice it and complain, even if it is less likely in the case of an abstaining.

Privacy The post service can learn the vote of any voter by reading their mail. Any 2-coalition between the cast officer, the printer and the registrar can learn the opinion of every voter by sharing the informations they have: the registrar knows (V_i, t_i), the printer knows (V_i, c_i) and the polling station knows (t_i, c_i, v_i). In addition, $(t_i \oplus c_i, v_i)$ is public. Hence, any 2-coalition can find (V_i, v_i).

Coercion resistance This protocol is not coercion-resistant. It is possible to check whether a ballot is valid or not and has been counted or not so it is possible to sell a vote.

Accountability This protocol is not accountable. If a voter received aliases from the registrar and the printer that are different, they may raise an alert. The registrar can prove that they sent the right alias (if they did) by showing the printer signature of a_i, but it is impossible to know if the printer sent a wrong alias, if the post service swaps the voter's mail with an other ballot or even if the voter who complains exchanged their ballot with someone else to create chaos. Similarly, if a voter disagrees with what the cast officer published, the cast officer may be in charge or the post service or it may be the voter that raises a false alert.

2.2.4 Possible attacks

Let us imagine that a voter is not going to verify their vote and that a dishonest printer or a dishonest post service knows it. This can happen for example if this voter did not vote during the last few elections. Then, the attacker can give the voter a fake ballot with an invalid signature and use the legitimate valid ballot to vote. If the voter unexpectedly verifies and raises an alert, the printer and the post service can blame each other or accuse the voter of lying since they have no proof. A malicious voter could have raised a false alert with the only objective of creating chaos.

Similarly, a corrupted cast officer can modify any ballot that they receive and blame the post service or the voter if the voter verifies and complains.

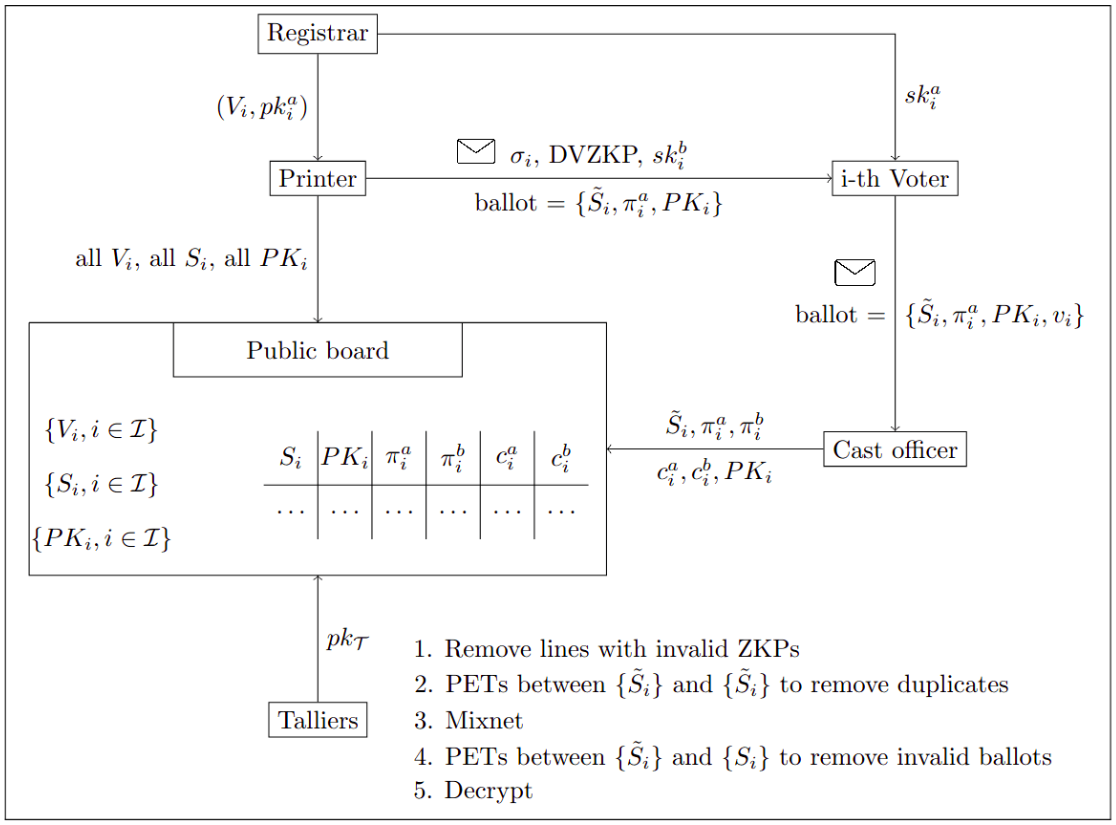
2.3 Condor

This protocol is largely inspired by JCJ. Our goal was to make Vote&Check coercion resistant. The different entities involved are the registrar, the printer, the cast officer, the trustees and every voter. There is a public public board where the printer, the cast officer and the trustees can write any message. We suppose that private and authenticated channels exist between the registrar and each voter and between the registrar and the printer. The protocol will be described in the next section and is summarised in Figure 4.

2.3.1 Description of the protocol

Setup During this phase, the registrar and the printer send the material to the voters. First of all, the trustees run a distributed key generation protocol and publish the associated public key $pk_{\mathcal{T}}$ on the public board. Then, for each voter V_i:

- the registrar generates a private key sk_i^a associated to the public key pk_i^a. They send (V_i, pk_i^a) to the printer and sk_i^a to V_i.



Registrar
(V_i, pk_i^a)
sk_i^a
σ_i, DVZKP, sk_i^b
Printer
i-th Voter
ballot $= \{\tilde{S}_i, \pi_i^a, PK_i\}$
all V_i, all S_i, all PK_i
ballot $= \{\tilde{S}_i, \pi_i^a, PK_i, v_i\}$
Public board
$\{V_i, i \in \mathcal{I}\}$
$\{S_i, i \in \mathcal{I}\}$
$\{PK_i, i \in \mathcal{I}\}$
S_i PK_i π_i^a π_i^b c_i^a c_i^b
$\tilde{S}_i, \pi_i^a, \pi_i^b$
c_i^a, c_i^b, PK_i
Cast officer
$pk_\mathcal{T}$
Talliers
1. Remove lines with invalid ZKPs
2. PETs between $\{\tilde{S}_i\}$ and $\{\tilde{S}_i\}$ to remove duplicates
3. Mixnet
4. PETs between $\{\tilde{S}_i\}$ and $\{S_i\}$ to remove invalid ballots
5. Decrypt

Figure 4: Description of the Condor protocol

- the printer generates a voting credential σ_i and two encryptions $S_i = enc(\sigma_i, pk_\mathcal{T}, r_i)$ and $\tilde{S}_i = enc(\sigma_i, pk_\mathcal{T}, s_i)$ with two different random numbers r_i and s_i. They generate a public key pk_i^b associated to a private key sk_i^b. They generate a *designated verifier zero-knowledge proof* of knowledge of sk_i^a such that $pk_i^a = pk(sk_i^a)$ or of r_i, s_i and σ_i such that $S_i = enc(\sigma_i, pk_\mathcal{T}, r_i)$ and $\tilde{S}_i = enc(\sigma_i, pk_\mathcal{T}, s_i)$. Finally, they generate a zero-knowledge proof of knowledge π_i^a of (σ_i, s_i) such that $\tilde{S}_i = enc(\sigma_i, pk_\mathcal{T}, s_i)$. They send to V_i by postal mail σ_i, sk_i^b, the DVZKP, s_i and a ballot containing $\tilde{S}_i$, π_i^a and $PK_i = pk_i^a \times pk_i^b$. There is also a checkbox for each possible vote.

After that, the printer publishes on the public board the shuffled list of the voters $V_i, i \in \mathcal{I}$, the shuffled encrypted list of the valid credentials $S_i, i \in \mathcal{I}$ and the shuffled list of the legitimate public keys $PK_i, i \in \mathcal{I}$.

Vote Each voter V_i receives their voting material from the printer and their verification material from both the printer and the registrar. The first step is optional: the voter may verify that their material is correct. It is the case if $\tilde{S}_i = enc(\sigma_i, pk_\mathcal{T}, s_i)$, $pk_i^a = pk(sk_i^a)$, $pk_i^b = pk(sk_i^b)$, $PK_i = pk_i^a \times pk_i^b$, if the DVZKP holds for a S_i published on the board and if π_i^a holds. The voting step itself does not require the use of a computer. The voter simply ticks the box corresponding to their choice and send their ballot by post to the polling station.

When the cast officer receives a ballot, they check that its structure is correct, i.e. that it contains a ciphertext, a zero-knowledge proof and a public key from the list. In this case, they verify that the ciphertext $\tilde{S}_i$ has not already been used and that π_i^a holds. If this verification succeeds, then they publish on the public board $\tilde{S}_i$, PK_i, π_i^a, $c_i^a = enc(v_i, PK_i, t_i)$, $c_i^b = enc(v_i, pk_\mathcal{T}, u_i)$ with v_i the cast vote and t_i and u_i two random numbers. They also publish a zero-knowledge proof of knowledge π_i^b of (v_i, t_i, u_i) such that $c_i^a = enc(v_i, PK_i, t_i)$ and $c_i^b = enc(v_i, pk_{\mathcal{T}}, u_i)$.

Each voter can verify that there is a line on the board with their encrypted credential $\tilde{S}_i$. Then, they may decrypt the associated c_i^a with their secret key $sk_i^a + sk_i^b$ and verify that they get their vote v_i.

Tally When the casting phase is over, the result of the election is computed as well as the list of the voters who indeed voted. The trustees extract the lines corresponding to ballots from the public board where both zero-knowledge proofs are valid. They perform verifiable plaintext equivalence tests on the encrypted credentials and remove the duplicates. They shuffle and re-encrypt the list of $(\tilde{S}_i, c_i^b)$ using a verifiable mixnet. They performs plaintext equivalence tests between the S_i and the $\tilde{S}_i$ and verifiably decrypt the votes that are next a valid credential. Anyone can compute the result of the election and the printer can compute the voter's list knowing which S_i made a plaintext equivalence text successful.

2.3.2 Comments on the design

- Like in Vote&Check, the information used by a voter to verify their vote is split between the registrar and the printer to prevent a *honest but curious* authority from breaking the privacy.
- The designated verifier zero-knowledge proof is the key to achieve coercion resistance: the printer does not know sk_i^a so it still ensures that the related credential is correct. On the other hand, the voter knows it so they can make a valid DVZKP using a fake credential, give this fake credential to a coercer and use the real one to vote.

2.3.3 Security properties

- **Verifiability:** The individual verifiability is true if the registrar is honest. Every conscientious voter can verify that their vote has been counted: they have to verify that their material is correct, that their ballot has been published and well encrypted and that it has not been considered as a duplicate. If the registrar is honest, their vote will be tallied as intended if these tests are successful.
- **Eligibility:** the only attacker that can cast illegitimate ballots with the certainty of not being detected is the coalition between the registrar and either the printer or the postal service. However, a dishonest printer can use the credential they generated to cast illegitimate ballots, a dishonest tally can do the same by decrypting the encrypted list of valid credentials published by the printer and a dishonest cast officer can use the credential on the ballot they receives instead of publishing them.
- **Privacy:** the privacy holds if the post is honest and if the printer is honest or if the registrar, the polling station and the tally are honest.
- **Coercion resistance:** this scheme resists to coercion. Each voter can forge one fake ballot indistinguishable from a valid one and give it to the coercer. Then, the voter can use their real ballot to vote and have it counted the way they wanted.

2.4 Comparative table

The following table lists the properties of the different postal voting protocols. Each cell shows which entities have to be honest for a property to be true. The formula $A \wedge B$ indicates that both A and B have to be honest for the property to be true, and $A \vee B$ that either A or B have to be honest for the property to be true. If there is no parenthesis, conjunctions should be applied before disjunctions.

	Postal voting	STROBE	Vote&Check	Condor
Cast as intended	✓	✓	✓	✓
Recorded as cast	Po ∧ C	✓	✓	R
Tallied as recorded	✓	✓	✓	✓
Eligibility	Po ∧ Id	Pr ∧ Po	Po ∧ (R ∨ Pr ∧ C)	Pr ∧ Po ∧ C ∧ T
Ballot secrecy	Po ∧ C	Pr ∧ Po	Po ∧ (R ∧ Pr ∨ Pr ∧ C ∨ R ∧ C)	Po ∧ (Pr ∨ R ∧ C ∧ T)
Coercion Resistance	x	x	x	✓

Table 1: Comparative table of the different postal voting protocols

The property is true if those entities are honest :
R = Registrar
Pr = Printer
Po = Post service
C = Cast officer
T = Talliers
Id = anyone that know the identity card of an other voter.
The ✓ symbol means that the property is unconditionally true. The x symbol means that even if all entities are honest, the property is still false.

3 Formal verification

There are two different ways to analyse protocols : the symbolic model or the cryptographic model. In this section, I will present a symbolic analysis of Condor. I will describe the ProVerif tool [2] that we used, how we modeled the protocol and the security properties we got.

3.1 The ProVerif tool

ProVerif is a software that automatically analyses security protocols. It can answer two different kind of questions : equivalence properties and reachability queries. In the first ones, the attacker tries to distinguish between two different situations, for example Alice votes A, Bob votes B and Alice votes B, Bob votes A. It is used to model the voter privacy. In the second case, the attacker tries to reach a given state in the protocol, for example having an illegitimate ballot counted or making a verification succeed despite their ballot has been altered. It is used to model eligibility or individual verifiability.

The messages exchanged during the protocol are modeled by *terms*. Terms are defined recursively from a set $\mathcal{X}$ of *variables*, a set $\mathcal{N}$ of *names* and a set $\mathcal{C}$ of *constructors*: a term can be a variable, a name or the image of some other terms by a constructor. An *expression* represents a computation. It is defined recursively from terms, constructors and a set $\mathcal{D}$ of *destructors*: an expression can be a term, the image of other expressions by a constructor or a destructor or fail. When evaluation an expression E, we write $E \Downarrow T$ if E reduces to a term T and $E \Downarrow fail$ in the other case. *Formulas* are defined from expressions: a formula ϕ can be true, false, the conjunction of two formulas, their disjunction, the negation of a formula or $E_1 = E_2$ where E_1 and E_2 are two expressions. The formula $E_1 = E_2$ is $\top$ if E_1 and E_2 evaluates to the same term and $\bot$ in the other case.

Example 3.1. Asymmetric encryption of a message m is modeled by `enc(m, pkey, r)` where `enc` is a constructor, `m` a variable of type bitstring, `pkey` a variable of type public key and `r` a variable of type random. The decryption is modelled by the rewriting rule `dec(enc(m, pk(sk), r), sk) = m` where `dec` is a destructor and `pk` a constructor.
Similarly, we define the signature and its verification, different zero-knowledge proofs and their verifications, plaintext equivalence test, proof of correct decryption and its verification...

The actions that follow each other during the protocol are modelled by *processes*. Again, processes are defined recursively. The null process 0 does nothing. Processes $out(C, M); P$ and $in(C, x : T); P$ output the term M on the *channel* C or input on C a message of type T and store

Terms: T, $T_1 \dots T_k$ ::= x | n | f($T_i \dots T_k$), where $x \in \mathcal{X}, n \in \mathcal{N}$ and $f \in \mathcal{C}$.
Expressions: E, $E_1 \dots E_k$ ::= T | g($E_1 \dots E_k$) | `fail` where $g \in \mathcal{C} \cup \mathcal{D}$.
Formulas: ϕ, ϕ_1, ϕ_2 ::= $E_1 = E_2$| `true` | `false` |$\phi_1 \wedge \phi_2$|$\phi_1 \vee \phi_2$| $\neq \phi$
Processes: P, Q :: = 0 | out(T_1,T_2); P | in($T_1, x : t$); P | (P | Q) | !P | if ϕ then P else Q | let x = E in P else Q | event(T); P

Figure 5: ProVerif's syntax

it in the variable x, and then proceed with the process P. The parallel composition of processes P and Q is written $P|Q$, while $!P$ is the unbounded replication of P. If ϕ is a formula, the conditional process if ϕ then P else Q takes the branch P if ϕ evaluates to $\top$ and the branch Q otherwise. If E is an expression, the assignment let $x = E$ in P else Q runs the process P with x bounds to E if E reduces to a term and runs the process Q in the other case.

Some security properties are modelled in ProVerif using *events* and *queries*. One type of query is attacker(s), meaning that the attacker must not learn the value of s. If M is a term and P a process, event(M); P is a process that triggers the event M and then behaves like P. It means that a specific state has been reach, for example event(SENT(c, m)) can mean that the message m has been sent on the channel c. The attacker can not trigger events on their own. The query event(E1(a)) $\Longrightarrow$ event(E2(b)) means that in every execution of the protocol where the event E1 triggers with the value a, then the event E2 must have triggered before with the value b.

Example 3.2. Here is an example of the modelling of a very simple protocol where Alice has a secret key skA and Bob wants to send Alice a message m on a public channel c.

```
1. out(c, pk(skA));                (* Alice publishes her public key *)
2. in(c, x: bitstring);            (* Alice receives Bob's message *)
3. let m' = dec(x, skA);
4. event(RECEIVED(m'))
5. |
6. in(c, pkA: pkey);               (* Bob receives Alices's public key *)
7. new r: rand;
8. event(SENT(m));
9. out(c, enc(m, pkA, r)).         (* Bob sends his message *)
```

In this example, the attacker may send his own public key on line 6 or learn the public key of Alice on line 1 and send her a message of their choice on line 2. Thus, the query attacker(m) that models the secrecy of the communication would be false, as well as the query event(RECEIVED(m)) $\Longrightarrow$ event(SENT(m)) that model its integrity. This man-in-the-middle attack is the typical type of attacks that can be found by ProVerif.

The other tool we use to model security property is the *equivalence*. Two processes are said equivalent if the attacker cannot distinguish between them. For example, to model the resistance to vote buying, the process where a voter sells their own material should be equivalent to the process where the voter sells forged material.

To answer the questions of equivalence, correspondence and secrecy, ProVerif uses the following modelling. The attacker is able to compute any terms from what they already know: names, variables and other terms. They can read every message and send any term they know on any channel they have access (i.e. any channel that is not explicitly private). When several processes are running in parallel, they can choose in which order they will execute. The answer can be true if the attacker cannot win in any execution, false if there is an execution where the attacker can win or cannot be proved if ProVerif's approximations lead it to find a false attack.

3.2 Condor's model

In this section, I will describe several choices we made in our model.

Mailboxes ProVerif's modelling of communication channels is the following: any entity that has access to a private channel can send any message on it and read any message that has been sent there. This is not fully satisfactory in the case of postal voting. The printer sends the voting material to the voter by postal mail and they receive it in their mailbox. Only the recipient has access to the messages they receive, but anyone can send them messages. Also, the recipient cannot know the origin of a message. Thus, we have adopted the following model :

```
1. new Mailbox_i: channel;             (* A private channel for voter V_i *)
2. !(in(public, message: bitstring);
3.   out(Mailbox_i, message)).
```

It means that the attacker can send any message to a voter. It ensures that the individual verifiability property still holds even if a dishonest entity sends forged material to a voter and that this voter uses it instead of the real one.

Mixnet Modeling a mixnet is not an easy task. Since its cryptographic implementation is verifiable and thus the trustees cannot cheat during this phase, we decided to omit it for the verification model. However, this conventional choice is not possible for the privacy model, where the mixnet plays a major role. Therefore, we used a very simplified model the mixnet, that only shuffles Alice's and Bob's ballots.

```
1. Voter(Alice, choice[0, 1]) |       (* Alice votes 0 (resp. 1) *)
2. Voter(Bob, choice[1, 0])           (* Bob votes 1 (resp. 0)   *)
...
1'. let ballot_1 = choice[ballot_Alice, ballot_Bob] in
2'. let ballot_2 = choice[ballot_Bob, ballot_Alice] in
    (* The two ballots are tallied *)
3'. out(public, ballot_1) | out(public, ballot_2).
```

It means that the attacker has to distinguish between two cases. In the first case, Alice has voted 0 and Bob has voted 1 earlier in the process. The ballot of Alice is published on the left side of the vertical bar and Bob's one on the right side. In the second case, Bob has voted 0 and Alice has voted 1, Bob's ballot is published on the left side and Alice's on the right side.

If we only write `out(public, ballot_Alice) | out(public, ballot_Bob)`, ProVerif fails to prove the equivalence. The attacker would only execute the left side of the vertical bar (what cannot be done in practice), deduce Alice's vote and perform a distinguisher test. The lines 1' and 2' are here to avoid this false attack. Thanks to the commutativity of the vertical bar that ProVerif somehow fails to use in its internal reasoning, the model is still correct.

Different types of voters Honest voters can do a lot of verifications but they are not mandatory. It is possible for a lazy voter to cast a ballot without carrying any of them. Hence, there should be tree different types of voter : the conscientious ones, the lazy ones and the corrupt ones. Conscientious voters are doing all the possible verifications, lazy voters are only casting their ballots and corrupt voters collude with the attacker. In the model, every corrupt voter publishes all their private channels on the public channel so that the attacker can read and write on them.

Public board In the protocol, we suppose that there is a public board where different entities can write messages. Those messages cannot be removed and anyone can read them and see their sender. In our model, each entity that has to write on the board has a signature key. A public channel is used as a public board, where entities publish their signed messages.

Zero-knowledge proofs Zero-knowledge proofs are modeled in a very abstract manner. For example, to prove the knowledge of the plaintext associated to a ciphertext, one only needs the plaintext, the ciphertext, the random number and the private key used for the encryption. To verify this proof, one needs the proof, the ciphertext and the public key. This leads to the following model:

```
1. fun ZKP1(bitstring, bitstring, pkey, random): bitstring.
2. reduc forall message: bitstring, r: rand, k: pkey;
      verify_ZKP1(ZKP1(enc(message, k, r), message, k, r),
      enc(message, k, r), k) = true.
```

The designated-verifier ZKP is slightly more complex. Since the proof must be valid in two different cases (the credential is valid or the private key is known), we need to use two rewriting rules. The syntax is then different.

```
1. fun DVZKP(skey, pkey, bitstring, bitstring, pkey, rand): bitstring.
2. fun verify_DVZKP(bitstring, pkey, bitstring, bitstring, pkey): bool
      reduc forall sk1: skey, pk1: pkey, x: bitstring, pk2: pkey, r: rand;
          verify_DVZKP(DVZKP(sk1, pk1, enc(x, pk2, r), x, pk2, r), pk1,
          enc(x, pk2, r), x, pk2) = true
      otherwise forall sk1: skey, x: bitstring, y: bitstring, pk2: pkey,
          r: rand; verify_DVZKP(DVZKP(sk1, pk(sk1), y, x, pk2, r),
          pk(sk1), y, x, pk2) = true.
```

The DVZKP proves that a private key associated to a first given public key is known or that a given ciphertext is the encryption of a given plaintext under a second given public key. To create the proof, one needs all the given informations, a private key and a random number. The proof is true if and only if the first public key is associated to the private key or if the ciphertext is the encryption of the plaintext under the second public key using the random. To verify the proof, one need both public keys, the plaintext, the ciphertext and the proof.

Unicity of recorded ciphertexts According to the protocol, when the cast officer receives a ballot with an encryption of a credential that was already used, they must not publish it. This kind of instruction is not well supported in ProVerif. Instead, we used a restriction: only the cases where all the published encryptions are unique are computed. Other cases are not relevant since if the cast officer publishes the same encryption twice, it will immediately be detected.

```
1. restriction date1: stamp, date2: stamp, cred: bitstring;
      event(USED_VALID_CRED(cred, date1))
          && event(USED_VALID_CRED(cred, date2)) ==> date1 = date2.
```

Every time the cast officer publish a ballot containing the encrypted credential `cred`, it triggers an event `USED_VALID_CRED(cred, date)` where `date` is an unique time stamp. Then, the restriction makes it impossible for the cast officer to publish twice the same credential.

3.3 Security properties

In order to analyse the security of our protocol, we designed two different models: one for the correspondence properties and one for the equivalence properties. In this section, I will present those two models.

3.3.1 Verifiability

Individual verifiability and *eligibility* are correspondence properties. They are modeled in ProVerif by *queries*. Informally, queries are statements of the form "if this event occurs during the execution

of the process, then this other event has already occurred before". The syntax is as follows:
`query x1: t1 ... xn: tn; event(E1(M)) ==> query(E2(N)).`
where `M` and `N` are terms obtained by applying constructors to the variables `x1 ... xn` of types `t1 ... tn` and `E1` and `E2` are events. This query is false if there exist a trace where `E2` is executed and `E1` is not and true if in every trace where `E2` is executed, `E1` has been executed too. Sometimes, ProVerif cannot determine if the query is true or false. A query can also include formulas on one side or the other of the `==>` symbol.

Individual verifiability Individual verifiability is usually divided in *cast as intended*, *recorded as cast* and *tallied as recorded.* In our case, every ballot is cast as intended since the vote is not encrypted. Also, tally's proofs of correct decryption proves that the ballots are tallied as recorded. The following query asks for the recorded as cast:

```
1. query S_i: bitstring, Stilde_i: bitstring, v: bitstring, c: bitstring,
       sk_T: skey, r1: rand, r2: rand, r3: rand, cred: bitstring;
       event(HAPPY_VOTER(S_i, Stilde_i, v, c)) ==> event(ELECTION_KEY(sk_T))
       && c = enc(v, pk(sk_T), r1) && S_i = enc(cred, pk(sk_T), r2)
       && Stilde_i = enc(cred, pk(sk_T), r3).
```

`HAPPY_VOTER(S_i, Stilde_i, v, c)` is an event triggered when a conscientious voter finishes their verification after having voted v, using the encrypted credential `Stilde_i`, equivalent to the public encrypted credential `S_i`, and that their ballot as been encrypted as `c`. We want that if it is reached, then the voter's ballot has been recorded properly. The event `ELECTION_KEY(sk_T)` means that `sk_T` is the private key of the election. Following formulas imply that the voter's credential is valid and that their vote has been encrypted as they wanted.

If the query is true, then the voter can be convinced that there is a ballot on the board that has been recorded with the credential they received and encrypting their vote. This ballot comes either from this voter or from an other voter that cast the same vote with the same $\tilde{S}_i$, the same key and the same encryption of the same credential. This is possible only if the registrar is dishonest (the message sk_i^a is sent on a private and authenticated channel so it cannot be modified).

After running the program for each possible configuration of honest and corrupt authorities, we obtained the following results: the property of individual verifiability is true if and only if the registrar is honest.

Eligibility The property of eligibility states that each valid ballot in the ballot box must come from an eligible voter, each voter being the sender of at most one valid ballot. The following query asks for the eligibility:

```
1. query encrypted_vote: bitstring, cred: bitstring, encrypted_cred: bitstring,
   vote: bitstring, V: voter, r: rand, sk: skey;
       event(ABOUT_TO_BE_TALLIED(encrypted_vote)) ==>
           (event(ELECTION_KEY(sk)) &&
            event(BALLOT_CAST(V, cred, encrypted_cred, vote)) &&
            event(BALLOT_RECORDED(encrypted_cred, encrypted_vote)) &&
            encrypted_vote = enc(vote, pk(sk), r))
           || (event(BALLOT_RECORDED(encrypted_cred, encrypted_vote)) &&
               event(CORRUPT_CRED(cred)) &&
               encrypted_cred = enc(cred, pk(sk), r)).
```

`ABOUT_TO_BE_TALLIED(encrypted_vote)` is an event triggered when the talliers are about to decrypt a ballot. The event `ELECTION_KEY(sk)` means that the trustee's secret key is `sk`. The event `BALLOT_CAST(V, cred, encrypted_cred, vote)` is triggered when the voter `V` uses the encryption `encrypted_cred` of the credential `cred` to cast a ballot containing the vote `v`. The event `BALLOT_RECORDED(encrypted_cred, encrypted_vote)` means that the cast officer has recorded a ballot with this encrypted credential, and that `encrypted_vote` is the encryption of the vote under $pk_{\mathcal{T}}$. Finally, the event `CORRUPT_CRED(cred)` means that this credential has been sent to a corrupt eligible voter.

If this query is true, then each valid ballot on the box come either from a dishonest eligible voter or from an honest eligible voter that intended to cast this vote.

After running the program for each possible configuration of honest and corrupt authorities, we obtained the following result: the property of eligibility is true if and only if the printer, the cast officer and the talliers are honest.

Computed attack The automated verification has shown that the registrar can fool the voters and make their verification inappropriately succeed, while the printer, the cast officer and the talliers can stuff the ballot box. The attacks on the eligibility were predictable since they know all the valid credentials: the printer generated them, the talliers can decrypt them from the board and the cast officer can read them on the received ballots. Then, they can use them to cast valid and illegitimate ballots. The registrar's attack on the verifiability, however, was more surprising. I will describe it in this section.

In this attack, the registrar is dishonest and at least a voter is dishonest too. Let us assume that the attacker can block messages on the postal channel. The registrar send the same private key sk^a to all the targeted voters and to the corrupt voter. They say to the printer that pk^a is the public key of the corrupt voter and that random keys are the ones of the targeted voters. When the printer mails the material to the voters, the attacker blocks every message sent to the targeted voters and reads the private key sk^b of the corrupt voter. Now, the registrar knows the private key associated to a valid public key, and therefore can forge invalid material indistinguishable from the valid one, using the evasion strategy. They generate a random number for each targeted voter, and a DVZKP that this number is a valid credential or that $SK = sk^a + sk^b$ is known (what is the case). Then, they send a fake credential, a DVZKP and a fake ballot to each targeted voter, pretending to be the printer. Those voters will have their ballot spoiled without leaving them a chance to realize it.

3.3.2 Privacy

We used the definition of privacy of [9]. The *ballot secrecy* is modeled by an equivalence property. We assume that there are at least two honest voters, Alice and Bob. The attacker has to distinguish between two processes. In the first one, Alice tries to vote 0 and Bob tries to vote 1 while in the second one, Bob tries to vote 0 and Alice tries to vote 1. We write `Alice(0)` | `Bob(1)` $\approx$ `Alice(1)` | `Bob(0)`. For example, the attacker can provide an invalid ballot to Bob. If Bob uses it, only Alice's vote will be tallied so the attacker will learn her vote and distinguish between the two processes.

Unfortunately, we did not obtain any results from ProVerif with this model. Either the computation time is too long or the calculations do not end, but whatever the cause, we never saw the end. Even a reduced model without any corrupt voter did not terminate after 3 days of calculation.

4 Conclusion

We have proposed two different protocols for postal voting. The fist one, Vote&Check, has better properties than the classic postal voting, while being ignorable by the voters who do not care about verifiability and prefer to vote as usual. The second one, Condor, was an attempt to bring coercion resistance, what seems contradictory with the individual verifiability, and even more in a context of remote voting, where the voter can give all their material to the coercer. In doing so, we have unfortunately degraded the security properties compared to the classic postal voting scheme, and made it vulnerable to an undetectable attack with only one dishonest entity.

Studying a protocol versus all possible corruption cases requires a lot of attention to detail, and the risk of making a mistake increase with its complexity. For this reason, we used an automated verifier. Some unusual aspects of the protocol, for example the postboxes, required a specific modeling to be as close as possible of what they represent.

The natural continuation of this work would be to modify the Condor protocol to avoid the attack of a corrupt registrar, and refining the ProVerif model of the tally.

References

[1] Josh Benaloh. Strobe-voting: Send two, receive one ballot encoding. In *Electronic Voting: 6th International Joint Conference, E-Vote-ID 2021, Virtual Event, October 5–8, 2021, Proceedings*, page 33–46, Berlin, Heidelberg, 2021. Springer-Verlag.

[2] Bruno Blanchet. Using Horn Clauses for Analyzing Security Protocols. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*, pages 86 – 111. IOS Press, 2011.

[3] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on Randomizable Ciphertexts. In Rosario Gennaro, editor, *PKC 2011 - Proceedings of the 2011 International Conference on Practice and Theory in Public Key Cryptography*, volume 6571 of *LNCS - Lecture Notes in Computer Science*, pages 403–422, Taormina, Italy, March 2011. Springer.

[4] Alexandre Boudet and Geoffroy Clavel. Législatives 2017: pas de vote par internet en raison du risque de cyberattaques. https://www.huffingtonpost.fr/politique/article/legislatives-2017-pas-de-vote-par-internet-en-raison-du-risque-de-cyberattaques_95623.html.

[5] Véronique Cortier, Jérémie Detrey, Pierrick Gaudry, Frédéric Sur, Emmanuel Thomé, Mathieu Turuani, and Paul Zimmermann. Ballot stuffing in a postal voting system. In *2011 International Workshop on Requirements Engineering for Electronic Voting Systems*, pages 27–36, 2011.

[6] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. BeleniosVS: secrecy and verifiability against a corrupted voting device. Research report, CNRS, Inria, LORIA ; Orange Labs, May 2019.

[7] Braden L. Crimmins, Marshall Rhea, and J. Alex Halderman. Remotevote and SAFE vote: Towards usable end-to-end verification for vote-by-mail. *CoRR*, abs/2111.08662, 2021.

[8] Ari Juels, Dario Catalano, and Markus Jakobsson. *Coercion-Resistant Electronic Elections*, pages 37–63. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[9] Steve Kremer and Mark D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In Mooly Sagiv, editor, *Programming Languages and Systems — Proceedings of the 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200, Edinburgh, Scotland, UK, April 2005. Springer.

[10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash attacks on the verifiability of e-voting systems. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 395–409. IEEE Computer Society, 2012.

[11] Ben Riley-Smith. Tory leadership vote delayed after gchq hacking alert. https://www.telegraph.co.uk/politics/2022/08/02/tory-leadership-voting-delayed-gchq-hacking-warning/.

[12] Code électoral. Vote par correspondance sous pli fermé (articles r176-4 à r176-4-7). https://www.legifrance.gouv.fr/codes/id/LEGISCTA000024372228/.