Distributed ElGamal *à la* Pedersen - Application to Helios

Véronique Cortier
CNRS/LORIA
France

David Galindo
CNRS/LORIA
France

Stéphane Glondu
INRIA Nancy Grand-Est
France

Malika Izabachène
CNRS/LORIA
France

ABSTRACT

Real-world elections often require threshold cryptosystems so that any t out of ℓ trustees can proceed to tallying. This is required to protect the confidentiality of the voters' votes against curious authorities (at least $t+1$ trustees must collude to learn individual votes) as well as to increase the robustness of the election (in case some trustees become unavailable, $t+1$ trustees suffice to compute the election result). We describe a fully distributed (with no dealer) threshold cryptosystem suitable for the Helios voting system (in particular, suitable to partial decryption), and prove it secure under the Decisional Diffie-Hellman assumption. Secondly, we propose a fully distributed variant of Helios, that allows for arbitrary threshold parameters ℓ, t, together with a proof of ballot privacy when used for suffrage elections. Our modification of Helios can be seen as revision of the seminal multi-authority election system from Cramer, Gennaro and Schoenmakers, upon which the original Helios system is based. As such, our work implies, to our knowledge, the first formal proof of ballot privacy for the scheme by Cramer *et al.* Thirdly, we provide the first open-source implementation of Helios with a fully distributed key generation setup.

Keywords: voting protocols, Helios, ballot privacy, fully distributed cryptosystem, implementation.

1. INTRODUCTION

Ideally, a voting system should be both private and verifiable. Privacy ensures that no one knows that a certain voter has voted in a particular way, and verifiability ensures that everyone can trust the result, without having to rely on some authorities. One leading voting scheme that achieves both privacy and verifiability is Helios [4], based on a classical voting system proposed by Cramer *et al* [17] with variants proposed by Benaloh [6]. Helios is an open-source voting system that has been used several times to run real-world elections, including the election of the president of the University of Louvain-La-Neuve and the election of the 2010, 2011 and 2012 new board directors of the International Association for Cryptographic Research (IACR) [1]. Helios has been shown to ensure ballot privacy for successively stronger notions of privacy and more accurate implementations [15, 7, 9].

Yet none of the existing ballot privacy proofs [15, 7, 9, 11] for Helios considers a fully distributed setup phase with an arbitrary threshold t in the total number of trustees ℓ. That is, a setup phase where trustees generate the election public and secret keys without a trusted dealer while affording an arbitrary t-out-of-ℓ threshold parameters selection. We will refer to any variant of Helios enjoying this property as a fully distributed Helios, for short. Such a property is crucial towards achieving a reasonable level of confidentiality for the voters' votes and for the robustness of the election. Moreover, we are not aware of any publicly available open-source implementation of Helios enjoying a fully distributed key generation phase. The aim of this work is to give a solution to these two issues.

Our contributions. In order to build Helios upon the lightest and simplest fully distributed semantically secure cryptosystem, we show in Section 3 that the well-known Pedersen's [38] Distributed Key Generation (DKG) protocol applied to ElGamal can be proven semantically secure under the Decision Diffie-Hellman assumption, even if the resulting public key can not be guaranteed to be uniformly distributed at random [26, 28]. We do so by employing the techniques used in [27, 3, 28] to prove a similar result for fully distributed Schnorr signatures in the Random Oracle Model. Our analysis shows that those techniques can be safely used in scenarios other than digital signatures, where the adversary solves a decisional problem (in contrast to a search problem for digital signatures) and in the standard model (in contrast to the random oracle model).

We go further by providing the first detailed proof of ballot privacy for Helios affording arbitrary threshold without trusted dealer. The most accomplished proof of privacy for Helios-like voting schemes can be found in [9]. It proves privacy for the actual ElGamal scheme used in Helios and the actual tally function (that reveals partial decryption shares). It however abstracts away the distribution of the secret keys of the trustees that perform the tally and does not consider threshold decryption. In practice, threshold decryption is mandatory. Typically, three authorities detain the decryption shares and two among three authorities suffice, and are necessary, to perform the tally. This is crucial for the robustness of the election.

We note that the currently implemented version of Helios is subject to an attack against privacy [15]: an attacker

may (re)submit the ballot of an honest voter on his behalf without knowing the actual vote. A provably secure fix [7] consists in invalidating ballots that contain ciphertexts already present in the bulletin board, this is called *ciphertext weeding*. Interestingly, our fully distributed Helios avoids ciphertext weeding: we can provably avoid the submission of duplicated ballots by simply hashing the voter's identity in the non-interactive zero-knowledge proofs for ciphertext well-formedness used in Helios (wich in turn are obtained via the Fiat-Shamir transformation [20]). We provide a variant of this technique where identities are replaced by nonces; this variant is compliant with countries where identities cannot appear beside ballots, does not harm everlasting privacy and it might be incorporated in forthcoming versions of the official Helios system (see end of Section 4). Checking for duplicates is therefore alleviated to checking that the same identity (or credential thereof) does not appear twice in the board, a test which is needed anyway to avoid the same voter casting more than one ballot. Thus the complexity of the test depends only on the size of the number of voters. Moreover, election servers are typically replicated to handle more requests and to ensure better availability. Two replicated servers may therefore independently accept two ballots that contain duplicated encryption, allowing to mount the previously mentioned attack.

We have implemented our fully distributed version of Helios. Our implementation includes the suppression of ciphertext weeding and the fully distributed threshold cryptosystem. The code is openly accessible [29]. To our knowledge this is the first publicly available implementation of a variant of Helios using a fully distributed threshold cryptosystem.

Related work. In addition to Helios, several private and verifiable voting schemes have been proposed, including e.g. Civitas [14] and FOO [22]. Helios is currently the most usable (and used) remote voting scheme in practice.

The notion of ballot privacy or ballot secrecy has been extensively studied. Several privacy definitions for voting schemes have been proposed, from ballot privacy [30, 34, 7, 8, 9, 11] to coercion-resistance [31, 23, 33] and applied to voting schemes: Civitas has been shown to be coercion-resistant [14], while Helios has been shown to ensure ballot and vote privacy [15, 7, 9, 8].

There is an abundant literature regarding discrete log-based threshold cryptosystems (see e.g. [21, 36, 39, 12, 35]) but most of it assumes a trusted dealer, which amounts, in the case of voting protocols, to place the privacy of the election in the hands of a single authority. Propositions of distributed key generation (DKG) protocols with no dealer exist, such as [26, 41, 3, 28]. They focus on ensuring that the public key as output by the protocol is uniformly distributed as long as at least $t + 1$ honest parties cooperate. Compared to the classical DKG scheme of Pedersen [38], which was shown in [26] to output biased public keys in the presence of active adversaries, those stronger DKG protocols are more involved and expensive. Regarding voting systems with arbitrary threshold parameters, [19] describes an (non publicly-available) implementation of Civitas with a fully-distributed threshold cryptosystem using the distributed key generation protocol by Gennaro *et al.* [26]. Compared to ours, the latter key generation protocol is twice as complex.

As noted before, the currently implemented version of Helios is subject to an attack against privacy [15]: an attacker may (re)submit the ballot of an honest voter on his behalf without knowing the actual vote. The result of the election then counts the honest vote twice, which provides a bias to the attacker. In particular, in the case of three voters, the attacker knows the vote of the voter under attack. A provably secure fix [7] consists in invalidating ballots that contain ciphertexts already present in the bulletin board, this is called *ciphertext weeding*. This fix is conceptually simple but very heavy in practice. Indeed, upon receiving a ballot, the election server would need to access to the bulletin board and test whether any of the atomic ciphertexts that compose the ballot already appears in the bulletin board. The complexity of the verification test would therefore be a function of the number of voters times the number of candidates.

Helios [4] is in fact an implementation and simplification of a seminal work on multi-authority voting systems by Cramer, Gennaro and Schoenmakers [17]. The fully-distributed IND-CPA scheme that we propose and analyze in Section 3 was already proposed in [25, 17], but no formal proof of semantic security was given. Our fully-distributed version of Helios resembles almost exactly the voting scheme given in [17]. Thus our work can in particular be seen as a revision of the scheme by Cramer *et al.* inside a contemporary frame that validates all the security statements claimed by the authors back in 1997. We stress that in the original publication no security definitions nor formal proofs were given. Interestingly, [17] already proposed to avoid ballots duplication by adding the voter's identity *id* to the hash function in the NIZK proofs. This technique is sound, as proven in [24, 30] and indirectly here.

2. SYNTAX AND BALLOT PRIVACY FOR A VOTING SYSTEM

Election systems typically involve several entities:

1. *Registrars*: Denoted by $\mathcal{R} = \{R_1, \ldots, R_{n_R}\}$, is a set of entities responsible for registering voters.

2. *Trustees:* Denoted by $(\mathcal{T}_1, \ldots, \mathcal{T}_\ell)$, these authorities are in charge of producing the public and secret parameters of the election. They are responsible for tallying and publishing a final result.

3. *Voters*: The eligible voters $(V_1, \ldots, V_\tau)$ are the entities participating in a given election administered by $\mathcal{R}$. We let id_j be the public identifier of V_j.

4. *Bulletin board manager*: It is responsible for processing ballots and storing valid ballots in the bulletin board BB.

In an election system with interactive setup and tallying, the ℓ trustees can communicate via pairwise private authenticated channels. They have access to a commonly accessible append-only memory where every trustee can post messages, and these posts can be traced back to its sender. A setup interaction is then run between the ℓ trustees to build an election public key $\mathbf{pk}$ and corresponding private key $\mathbf{sk}$. Later, a tally interaction computes the final outcome result and a proof of valid tabulation Π.

We continue by describing the syntax for an electronic voting protocol that we will be using throughout the paper. A voting protocol $\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{VerifyVote}, \mathsf{Validate}, \mathsf{Box}, \mathsf{Tally}, \mathsf{Verify})$ consists of eight algorithms whose syntax is as follows:

$\mathsf{Setup}(1^\lambda, \ell)$ is a possibly interactive algorithm run by ℓ trustees. It takes as inputs the security parameter 1^λ and the total number ℓ of trustees. It outputs an election public key $\mathbf{pk}$, which includes the description of the set of admissible votes $\mathbb{V}$; a list of secret keys $\mathbf{sk}$. Each trustee only gets to see some part of the secret keys. We assume $\mathbf{pk}$ to be an implicit input of the remaining algorithms.

$\mathsf{Register}(1^\lambda, id)$ captures the registration phase that is intuitively inherent to any voting system. On inputs the security parameter 1^λ and a unique identifier id for the user, it outputs the secret part of the credential usk_{id} and the public part of the credential upk_{id}. We assume the list $L = \{\mathsf{upk}_{id}\}$ of legitimate public credentials to be included in the public key $\mathbf{pk}$. If no credentials are needed, upk_{id} is set to be id and usk_{id} is empty.

$\mathsf{Vote}(id, \mathsf{upk}, \mathsf{usk}, v)$ is used by voter id to cast his choice $v \in \mathbb{V}$ for the election. It outputs a ballot b, which may/may not include the identifier id and or upk_{id}.

$\mathsf{VerifyVote}(\mathsf{BB}, id, \mathsf{upk}, \mathsf{usk}, b)$ is a typically light verification algorithm intended to the voters, for checking that their ballots will be included in the tally. It takes as input the bulletin board BB, a ballot b, and the voter's credentials $\mathsf{usk}, \mathsf{upk}$ and performs some validity checks, returning `accept` or `reject`.

$\mathsf{Validate}(b)$ takes as input a ballot b and returns `accept` or `reject` for well/ill-formed ballots.

$\mathsf{Box}(\mathsf{BB}, b)$ takes as inputs the bulletin board BB and a ballot b. If $\mathsf{Validate}(b)$ accepts, it adds b to BB; otherwise, it lets BB unchanged.

$\mathsf{Tally}(\mathsf{BB}, \mathbf{sk})$ takes as input $\mathsf{BB} = \{b_1, \ldots, b_\tau\}$ and the secret key $\mathbf{sk}$, where τ is the number of ballots cast. It outputs the tally result, together with a proof of correct tabulation Π. Possibly, $\mathsf{result} = \texttt{invalid}$, meaning the election has been declared invalid.

$\mathsf{Verify}(\mathsf{BB}, \mathsf{result}, \Pi)$ takes as input the bulletin board BB, and a result/proof pair (result, Π) and checks whether Π is a valid proof of correct tallying for result. It returns `accept` if so; otherwise it returns `reject`.

Next we define the minimal procedural requirement, called *correctness*, that every such voting protocol must satisfy. It simply requires that honestly cast ballots are accepted to the BB (and pass the verification checks) and that, in an honest setting, the tally procedure always yields the expected outcome (that is, the result function). A voting scheme is *correct* if for $i = 1$ to τ it holds: (1) $\mathsf{Box}(\mathsf{BB}, b_i) = \mathsf{BB} \cup \{b_i\}$ where we let $b_i \leftarrow \mathsf{Vote}(id_i, \mathsf{upk}_i, \mathsf{usk}_i, v_i)$ for some $v_i \in \mathbb{V}$; (2) $\mathsf{Validate}(b_i) = \texttt{accept}$ and $\mathsf{VerifyVote}(\mathsf{Box}(\mathsf{BB}, b_i), \mathsf{upk}_i, \mathsf{usk}_i, b_i) = \texttt{accept}$; (3) $\mathsf{Tally}(\{b_1, \ldots, b_\tau\}, \mathbf{sk})$ outputs $(\rho(v_1, \ldots, v_\tau), \Pi)$; (4) $\mathsf{Verify}(\{b_1, \ldots, b_\tau\}, \rho(v_1, \ldots, v_\tau), \Pi) = \texttt{accept}$. The above properties can be relaxed to hold only with overwhelming probability. Possibly, one might need to include additional trust assumptions in the consistency definition. For instance, if the voting scheme has threshold parameters (t, ℓ), then it is likely that correctness can only be guaranteed if at least $t + 1 > \lceil \ell/2 \rceil$ trustees cooperate to compute Tally.

2.1 Ballot privacy

Ballot privacy requires that any coalition of at most t trustees together with any coalition of corrupted users cannot infer information from ballots cast by honest voters, even after the election result is announced. In the following, we extend the computational game-based definition from [9] in two ways: we take into account a fully distributed key generation phase with arbitrary threshold; we take into consideration the necessary voters' registration phase.

Formally, two experiments are considered: one in which tally is left to a simulator and another one in which tally is done as normal. In both of them, the adversary acts on behalf of corrupted trustees and users. As usual in the electronic voting protocol literature, we assume static corruption of the trustees; however users can be adaptively corrupted.

The challenger maintains two bulletin boards BB_0 and BB_1. It randomly chooses $\beta \xleftarrow{R} \{0, 1\}$, and the adversary will be given access to the left board if $\beta = 0$, or the right board if $\beta = 1$. Thus the board $\mathsf{BB}_{1-\beta}$ is invisible to the adversary. The adversary is given access to oracles $\mathcal{O}\mathsf{register}, \mathcal{O}\mathsf{corruptU}$ and $\mathsf{OvoteLR}$ as follows:

$\mathcal{O}\mathsf{register}(id)$: invokes algorithm $\mathsf{Register}(\lambda, id)$, it returns upk_{id} and keeps usk_{id} secret. It also updates given lists $L = L \cup \{\mathsf{upk}_{id}\}$ and $\mathcal{U} = \mathcal{U} \cup \{(id, \mathsf{upk}_{id}, \mathsf{usk}_{id})\}$.

$\mathcal{O}\mathsf{corruptU}(id)$: firstly, it checks if an entry $(id, *, *)$ appears in $\mathcal{U}$; if not, it halts. Else, it outputs the credential's secret key usk_{id} associated to upk_{id} and updates $\mathcal{CU} = \mathcal{CU} \cup \{(id, \mathsf{upk}_{id})\}$.

$\mathsf{OvoteLR}(id, v_0, v_1)$: if id was previously queried, or $\mathsf{upk}_{id} \notin \mathcal{U} \setminus \mathcal{CU}$, or $v_0 \notin \mathbb{V}$, or $v_1 \notin \mathbb{V}$, it halts. Else, it updates $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \cup \{\mathsf{Vote}(id, v_0)\}$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \cup \{\mathsf{Vote}(id, v_1)\}$.

We define two procedures Init and Main that help defining the view of the adversary in the ballot privacy game.

- $\mathsf{Init}(\lambda, \ell)$: it is run interactively by a challenger and the adversary. The challenger starts by picking a random bit β, and it sets up two bulletin boards BB_0 and BB_1, initialized at empty. The adversary is given access to BB_β. The lists $L, \mathcal{U}, \mathcal{CU}$ are initialized at empty. In a first phase, the adversary is given access to $\mathcal{O}\mathsf{register}()$. At the end of this phase, the lists $L, \mathcal{U}, \mathcal{CU}$ have been populated and will be fixed for the rest of the game. In a second phase, the setup algorithm is run and $(\mathbf{pk}, \mathbf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, t, \ell)$. Eventually we might allow the adversary to corrupt a subset of trustees. In that case, when running $\mathsf{Setup}()$, the adversary will control the corrupted trustees (in particular the adversary might deviate from the algorithm specification). At the end of this second phase, the public key $\mathbf{pk}$ is published, and it includes the set of admissible choices $\mathbb{V}$ and the list of public credentials L. The adversary ends with knowledge of the secret keys belonging to the corrupted trustees, whenever they are defined.
- $\mathsf{Main}(\mathsf{BB}_0, \mathsf{BB}_1, \mathbf{pk}, \mathbf{sk})$: If $\beta = 0$, the challenger sets $(\mathsf{result}, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$. If $\beta = 1$, the challenger sets $(\mathsf{result}, \Pi') \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$ and $\Pi \leftarrow \mathsf{SimTally}(\mathsf{BB}_0, \mathsf{BB}_1, \mathbf{pk}, \texttt{info})$, where `info` contains any information known to the challenger. The output is $(\mathsf{result}, \Pi,$

β'), where β' is the guess for β made by the adversary $\mathcal{A}$. If the adversary is allowed to corrupt a subset of trustees, then Tally and $\mathsf{SimTally}$ are jointly run between the challenger and the adversary, the challenger playing the role of the honest trustees and the adversary playing the role of the corrupted trustees.

Experiment $\mathsf{Exp}^{\mathsf{priv}}_{\mathcal{A}}(\lambda)$

(1) $\mathsf{Init}(\lambda) \rightarrow (\mathbf{pk}, \mathbf{sk}, \beta)$

(2) $\mathcal{A}^{\mathcal{O}^{\mathsf{corruptU}(),\mathsf{OvoteLR}()}}(\mathsf{BB}_\beta) \rightarrow (\mathsf{BB}_\beta, \mathsf{BB}_{1-\beta})$

(3) $\mathsf{Main}(\mathsf{BB}_0, \mathsf{BB}_1, \mathbf{pk}, \mathbf{sk}) \rightarrow (\mathsf{result}, \Pi, \beta')$

`return` $\beta = \beta'$

Figure 1: Ballot privacy

If $\beta = 1$ then the announced result will likely not match the result corresponding to the ballots on the board. Ballot privacy asks that an adversary cannot distinguish the real votes contained in the ballots from any other selection of votes.

Formally, we say that a voting protocol $\mathcal{V}$ has *ballot privacy* if there exists an efficient simulator $\mathsf{SimTally}$ such that no PPT algorithm can tell whether it interacts with the real tally algorithm or a simulator, i.e. there is a negligible function $\nu(\lambda)$ such that, for any PPT adversary $\mathcal{A}$, it holds that $\mathsf{Succ}^{\mathsf{priv}}(\mathcal{A}) = \left| \Pr\left[\mathsf{Exp}^{\mathsf{priv}}_{\mathcal{A}}(\lambda) = 1\right] - 1/2 \right| < \nu(\lambda)$, where $\mathsf{Exp}^{\mathsf{priv}}_{\mathcal{A}}$ is defined in Figure 1.

3. A FULLY DISTRIBUTED THRESHOLD CRYPTOSYSTEM FROM DECISION DIFFIE-HELLMAN

Our aim is to build a Helios-like voting protocol where to compute $\mathsf{Tally}()$ we shall require that at least $t+1$ trustees cooperate and follow the protocol specification, without assuming the existence of a trusted dealer. We need what is called a *fully distributed* (t, ℓ)*-threshold cryptosystem* or *threshold cryptosystem without trusted dealer*. In such a cryptosystem, there exist ℓ servers which can communicate via pairwise private authenticated channels. They have access to an append-only public board where every server can post messages, and these posts can be traced back to its sender. A setup interaction is then run between the ℓ servers to build a public key pk, servers' private keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell$, and eventually verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell$. The secret and verification keys will later enable any set of $(t+1)$ servers to non-interactively decrypt ciphertexts computed under the public key pk. On the contrary, any set of at most t servers can not learn any information on the plaintext embedded on any given ciphertext C. Rigorously, a fully distributed t-out-of-ℓ threshold cryptosystem with non-interactive threshold decryption consists of the following algorithms:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ is a fully distributed *key generation* algorithm that takes as input a security parameter 1^λ, the number of decryption servers ℓ, and the threshold parameter t; it outputs a public key pk, a list $\mathbf{sk} = \{\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell\}$ of servers' private keys, a list $\mathbf{vk} = \{\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell\}$ of verification keys.

$\mathsf{Enc}(\mathsf{pk}, m)$ is an *encryption algorithm* that takes as input the public key pk and a plaintext m, and outputs a ciphertext C. We may write $\mathsf{Enc}(\mathsf{pk}, m; r)$ when we want to make explicit the random coins used for encrypting.

$\mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{vk}_i, C)$ is a *share decryption algorithm* that takes as input the public key pk, the private key sk_i, the verification key vk_i, a ciphertext C, and outputs a decryption share (i, c_i).

$\mathsf{Rec}(\mathsf{pk}, \mathbf{vk}, C, \mathcal{C})$ is a *recovery* algorithm that takes as input the public key pk, a ciphertext C, and a list $\mathcal{C}$ of $t+1$ decryption shares, together with the verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell$, and outputs a message m or `reject`.

We recall the definition of *completeness* and *robustness* and *IND-CPA* security for a threshold cryptosystem:

Completeness: for any integers $1 \leq t \leq \ell$, for every admissible plaintext m, we require $\mathsf{Rec}(\mathsf{pk}, \mathbf{vk}, C, \mathcal{C}) = m$, where (1) $(\mathsf{pk}, \mathbf{sk}, \mathbf{vk}) \leftarrow \mathsf{DistKG}(1^\lambda, t, \ell)$; (2) $C = \mathsf{Enc}(\mathsf{pk}, m)$; (3) $(i, c_i) \leftarrow \mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{vk}_i, C)$; and (4) $\mathcal{C} \subseteq \{c_1, \ldots, c_\ell\}$ is any subset of $t+1$ elements.

Robustness: against active cheating adversaries means that for any ciphertext C and any two $(t+1)$-subsets of decryption shares $\mathcal{C} \neq \mathcal{C}'$ such that $\mathsf{Rec}(\mathsf{pk}, C, \mathcal{C}) \neq \mathtt{reject} \neq \mathsf{Rec}(\mathsf{pk}, C, \mathcal{C}')$ it holds that $\mathsf{Rec}(\mathsf{pk}, C, \mathcal{C}) = \mathsf{Rec}(\mathsf{pk}, C, \mathcal{C}')$.

IND-CPA security - Static corruptions: On input the total number of decryption servers ℓ and threshold t, an IND-CPA adversary $\mathcal{A}$ chooses t servers to be corrupted. We assume wlog. that $\mathcal{A}$ chooses servers 1 to t. From this point on $\mathcal{A}$ acts on behalf of corrupted servers, while the challenger acts on behalf of the remaining servers, which behave honestly (namely they follow the protocol specification).

The adversary $\mathcal{A}$ and the challenger run together the algorithm $\mathsf{DistKG}(1^\lambda, t, \ell)$, at the end of which the adversary learns $\mathsf{sk}_1, \ldots, \mathsf{sk}_t$. The public encryption and verification keys pk, vk are obtained. $\mathcal{A}$ chooses two admissible messages m_0, m_1 with equal length. The challenger chooses $\beta \xleftarrow{R} \{0,1\}$ and sends the adversary the encryption $\mathsf{Enc}(\mathsf{pk}, m_\beta)$. Finally $\mathcal{A}$ outputs its guess $\beta' \in \{0,1\}$. Let us define the output of the IND-CPA experiment as 1 if $\beta' = \beta$ and 0 otherwise.

We say that a fully distributed cryptosystem

$$\mathcal{E} = (\mathsf{DistKG}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{Rec})$$

has *IND-CPA security against static corruptions* if no PPT algorithm $\mathcal{A}$ can tell apart encryptions of m_0 and m_1, i.e. there is a negligible function $\nu(\lambda)$ such that, for any PPT adversary $\mathcal{A}$, it holds that $\mathsf{Succ}^{\mathsf{cpa}}(\mathcal{A}) = |\ \Pr\left[\mathsf{Exp}^{\mathsf{cpa}}_{\mathcal{A}}(\lambda) = 1\right] - 1/2\ | < \nu(\lambda)$, where $\mathsf{Exp}^{\mathsf{cpa}}_{\mathcal{A}}$ is the experiment above.

3.1 IND-CPA Fully Distributed (t, n)-Threshold Cryptosystem from DDH in the Standard Model

Few electronic voting works detail how to generate the election public and secret keys in a fully distributed manner with arbitrary threshold parameters (t, ℓ) such that $0 \leq t \leq \ell - 1$. One notable exception is the work by Juels, Catalano and Jakobsson [32], whose idea has been detailed and implemented by Davis, Chmelev and Clarkson [19]. They propose to use a computationally secure distributed simulation of the

process of generating a public key $Y = g^x, x \stackrel{R}{\leftarrow} \mathbb{Z}_q$. In particular, they propose the DKG scheme by Gennaro, Jarecki, Krawczyk and Rabin in [28].

In this section we give a full description and proof of a fully distributed cryptosystem compatible with Helios, which is in fact obtained by coupling Pedersen's DKG [38] with ElGamal. This cryptosystem has been previously described in [25, 17]. However we shall notice that no proof of semantic security has been given in these works, nor elsewhere to our knowledge. The latter is somehow unsatisfactory, since as it is common wisdom for cryptographic protocols, the devil is in the details. In fact Pedersen's protocol is shown in [28] to not always output uniformly random public keys, so in principle it does not seem to be a good choice to distribute ElGamal while retaining the DDH assumption (we refer to [40] for a definition of DDH) for semantic security. Pedersen's DKG is known to be sound in conjunction with Schnorr signatures in the Random Oracle Model, as shown in [27, 28]. Here we apply the same techniques to prove that Perdersen's DKG + ElGamal gives fully distributed semantically secure encryption under DDH. In this case the result is in principle more challenging, as [28] seems to indicate, since the adversary solves a decisional problem (in contrast to a search problem, as in the case of digital signatures) in the standard model (in contrast to the random oracle model). Still the same techniques used in the Schnorr case can be applied to the ElGamal case. Let $\mathcal{D} = (\mathsf{DistKG}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{Rec})$ be then the threshold cryptosystem [25, 17, 19]:

$\mathsf{DistKG}(1^\lambda, t, \ell)$

1. Each party P_i chooses a random t-degree polynomial $f_i(x) = a_{i0} + a_{i1}x + \ldots + a_{it}x^t \in \mathbb{Z}[x]$ and broadcasts $A_{ik} = g^{a_{ik}}$ for $k = 0, \ldots, t$. Denote the secret held by P_i as $s_i = f_i(0)$ and let $Y_i = g^{f_i(0)}$. Each party P_i computes shares $s_{ij} = f_i(j) \mod q$ of its own secret s_i for $j = 1, \ldots, \ell$ and sends $s_{ij} \in \mathbb{Z}_q$ secretly to party P_j.
2. Each party P_j verifies the shares he received by checking for $i = 1, \ldots, \ell$:

$$g^{s_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k} \quad (1)$$

   If a check fails for an index i then P_j broadcasts a *complaint* against P_i.
3. Party P_i reveals share $s_{ij} \in \mathbb{Z}_q$ if it receives a complaint against him by party P_j. If any of the revealed shares s_{ij} fails to satisfy Equation 1, then P_i is disqualified. Let us define the set $\mathrm{QUAL} \neq \emptyset$ as the set of qualified players.
4. The public key is computed as $\mathsf{pk} = \prod_{i \in \mathrm{QUAL}} Y_i$. Each P_j sets his share of the secret key as $x_j = \sum_{i \in \mathrm{QUAL}} s_{ij} \mod q$. The virtual decryption key $x = \sum_{i \in \mathrm{QUAL}} s_i \mod q$ is not needed to be known to be able to decrypt. The public verification keys are computed as $\mathsf{vk}_j = \prod_{i \in \mathrm{QUAL}} g^{s_{ij}}$ for $j = 1, \ldots, \ell$.

$\mathsf{Enc}(\mathsf{pk}, m)$ outputs $C = (R, S) = (g^r, Y^r \cdot m)$ for a plaintext $m \in \mathbb{G}$ and randomness $r \stackrel{R}{\leftarrow} \mathbb{Z}_q$.

$\mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{vk}_i, C)$ outputs $(i, c_i = R^{x_i})$.

$\mathsf{Rec}(\mathsf{pk}, \mathbf{vk}, C, \mathcal{C})$ parses $C = (R, S), \mathcal{C} = \{c_{i_1}, \ldots, c_{i_{t+1}}\}$ and outputs $m = S \cdot \left(\prod_{j \in \mathcal{I}} c_j^{\lambda_j^{\mathcal{I}}} \right)^{-1}$ with $\mathcal{I} = \{i_1, \ldots, i_{t+1}\}$, where the $\lambda_j^{\mathcal{I}}$'s are the Lagrange coefficients, $\lambda_j^{\mathcal{I}} = \prod_{k \in \mathcal{I} \setminus \{j\}} \frac{k}{k-j} \in \mathbb{Z}_q^*$. We thus have that $\sum_{j \in \mathcal{I}} f(j)\lambda_j^{\mathcal{I}} = f(0)$ for any polynomial f of degree at most t.

Let us see that the above cryptosystem is complete. Indeed, let $C = (R, S) = (g^r, Y^r \cdot m)$. Consider the equation $\sum_{j \in \mathcal{I}} \lambda_j^{\mathcal{I}} x_j = \sum_{j \in \mathcal{I}} \lambda_j^{\mathcal{I}} \left(\sum_{i \in \mathrm{QUAL}} s_{ij} \right) = \sum_{i \in \mathrm{QUAL}} \left(\sum_{j \in \mathcal{I}} \lambda_j^{\mathcal{I}} s_{ij} \right) = \sum_{i \in \mathrm{QUAL}} \left(\sum_{j \in \mathcal{I}} \lambda_j^{\mathcal{I}} f_i(j) \right) = \sum_{i \in \mathrm{QUAL}} s_i$. Then

$$\prod_{j \in \mathcal{I}} c_j^{\lambda_j^{\mathcal{I}}} = \prod_{j \in \mathcal{I}} (R^{x_j})^{\lambda_j^{\mathcal{I}}} = R^{\left(\sum_{j \in \mathcal{I}} \lambda_j^{\mathcal{I}} x_j\right)} = R^x$$

and completeness follows.

Theorem 3.1 *The above scheme is* IND-CPA *secure under the* DDH *assumption.*

Proof. The reduction we show next is based on the ideas used by Gennaro *et. al* [28] to prove that Pedersen's key generation protocol produces hard instances of the dlog problem. Let (g, g^a, g^b, h) be the instance of the DDH problem that we need to distinguish. That is, we need to distinguish between the case $h = g^{ab}$ or $h \stackrel{R}{\leftarrow} \mathbb{G}$. To this end, we will use the IND-CPA adversary against the cryptosystem from Section 3. We simulate the IND-CPA game to the IND-CPA adversary $\mathcal{A}$. Let B be the set of parties corrupted by $\mathcal{A}$. Let G denote the set of honest decryption servers that will be simulated by our reduction. Wlog let us assume that the ℓ-th server is honest, i.e. $T_\ell \in G$.

What does the adversary expect to see? In the first place, the adversary chooses before the start of the IND-CPA game (static corruption) the set $B \subset \{1, \ldots, \ell\}$ of players that it will corrupt, $|B| \leq t$. For each $i \in B$ the adversary plays the role of the i-th server T_i. At the end of the distributed key generation phase, the adversary learns the public and verification keys $\mathsf{pk}, \mathbf{vk}$. Next, adversary $\mathcal{A}$ will choose two different plaintexts $m_0, m_1 \in \mathbb{G}$ and asks to see $\mathsf{Enc}(\mathsf{pk}, m_\beta)$ for a random coin $\beta \stackrel{R}{\leftarrow} \{0, 1\}$. His goal is to learn β with probability significantly away from $1/2$.

We start our simulation of the IND-CPA game by running a regular instance of $\mathsf{DistKG}(1^\lambda, t, \ell)$, except that for server T_ℓ we cheat without $\mathcal{A}$ noticing, and this results in a simulation that provides g^a as T_l's contribution to the jointly computed public key pk. That is, for party T_ℓ we choose t values $s_{lj} \stackrel{R}{\leftarrow} \mathbb{Z}_q$ for $j \in B$ and we send it to corrupted server $T_j \in B$. Notice that there exists a unique polynomial $f_\ell(z)$ of degree t such that $f_\ell(0) = a$ and $f_\ell(j) = s_{lj}$ for $j \in B$. Let $f_\ell(z) = a_{\ell 0} + a_{\ell 1}z + \ldots + a_{\ell t}z^t \in \mathbb{Z}[z]$. Then it is known that there exists a proper set of efficiently computable Lagrange coefficients $\lambda_{\ell j}$ such that $a_{\ell i} = \lambda_{\ell 0}a + \sum_{j=1}^{t} \lambda_{\ell j} s_{\ell j}$, which are defined as a function of the values $s_{lj} \stackrel{R}{\leftarrow} \mathbb{Z}_q$ for $j \in B$ as indicated in [28]. We can not explicitly compute them, but instead we are able to compute an implicit representation $A_{\ell i} = g^{a_{\ell i}} = (h)^{\lambda_{\ell 0}} \prod_{j=1}^{t} g^{s_{\ell j}\lambda_{\ell j}}$. Finally we broadcast $A_{\ell 0}, \ldots, A_{\ell t}$ on behalf of T_ℓ.

Let pk be the public key output by the DistKG algorithm. [28] shows that pk can be written as $\mathsf{pk} = g^a \cdot Y_G \cdot Y_B$, where Y_G is the contribution of servers in $G \setminus \{\ell\}$, and Y_B is the contribution of parties in the set $B \cap \mathrm{QUAL}$. Furthermore, if we write $Y_G = g^{x_G}$ and $Y_B = g^{x_B}$, the simulator can explicitly compute both x_G and x_B. Indeed, as [28] argues, the simulator chose x_G on behalf of the honest servers. On the other hand, the contribution of each server in $i \in B$ that has not been disqualified is the free term of a polynomial $f_i(z) \in \mathbb{Z}_q[x]$ of degree t, and the simulator holds at least $t+1$ points on this polynomial. If follows that the simulator can compute each of these contributions and hence the value $x_B \in \mathbb{Z}_q$.

Now, let $m_0, m_1 \in \mathbb{G}$ be the plaintexts chosen by $\mathcal{A}$. Then we build the challenge ciphertext $C_\beta = (g^b, h \cdot g^{(x_G+x_B)b} \cdot m_\beta)$ for $\beta \xleftarrow{R} \{0,1\}$. Notice that if $h = g^{ab}$ then $h \cdot g^{(x_G+x_B)b} = \mathsf{pk}^b$; else if $h = g^r$ for $r \xleftarrow{R} \mathbb{Z}_q$ then $h \cdot g^{(x_G+x_B)b}$ is uniformly distributed at random in $\mathbb{G}$ as long as $(x_G+x_B)b \neq -r$. The latter can be discarded, as this only happens with negligible probability $1/q$.

Thus, using the standard reduction argument, we can conclude that any IND-CPA adversary against the above fully distributed threshold cryptosystem implies a DDH solver. □

3.2 A Transformation for NM-CPA Threshold Cryptosystems in the Random Oracle Model

We know from previous work that ballot private Helios-like voting protocols are tightly related to NM-CPA cryptosystems [9, 10]. Next we state that and IND-CPA fully distributed (t,ℓ)-threshold cryptosystem can be converted into a NM-CPA Fully Distributed (t,ℓ)-Threshold Cryptosystem by applying the transformation in [10, 9]. The definitions of Σ-protocols and the Fiat-Shamir transform are well-known and are recalled in Appendixes A and B.

Theorem 3.2 *Let $\mathcal{D}' = (\mathsf{DistKG}', \mathsf{Enc}', \mathsf{ShareDec}', \mathsf{Rec}')$ be an IND-CPA fully distributed (t,ℓ)-threshold cryptosystem and let $(\mathsf{Prove}, \mathsf{Verify})$ be a Σ-protocol for the language*

$$R((\mathsf{pk}, C), (m, r)) = 1 \iff C = \mathsf{Enc}'(\mathsf{pk}, m; r)$$

with special soundness, special honest-verifier zero knowledge, unique responses and an exponentially large (in the security parameter) challenge space $C \in \mathrm{Ch}$. Let $H : \{0,1\}^* \to \mathrm{Ch}$ be a random oracle. Then the following construction $\mathcal{D} = (\mathsf{DistKG}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{Rec})$, that uses the Fiat-Shamir transformation provides a NM-CPA fully distributed (t,ℓ)-threshold cryptosystem. The definitions of Σ-protocols and the Fiat-Shamir transform are well-known and are recalled in Appendixes A and B.

$\mathsf{DistKG}(\lambda)$*: is defined as* $\mathsf{DistKG}'(\lambda)$

$\mathsf{Enc}(\mathsf{pk}, m; r)$*: computes* $C' \leftarrow \mathsf{Enc}'(\mathsf{pk}, m; r)$ *and runs* Prove *on input* $((\mathsf{pk}, C'), (m, r))$ *until it outputs commitment com; it computes challenge* $ch \leftarrow H(\mathsf{pk}, C, com)$ *and sends this to* Prove*; obtains the response res from* Prove *and returns the ciphertext* $C \leftarrow (C', com, res)$.

$\mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{vk}_i, C)$*: parses C as* (C', com, res)*. Then if* $\overline{\mathsf{Verify}}((\mathsf{pk}, C'), com, ch, res) = 0$*, it returns* `reject`*. Else, it outputs whatever* $\mathsf{ShareDec}'(\mathsf{sk}_i, \mathsf{vk}_i, C')$ *outputs.*

$\mathsf{Rec}(\mathsf{pk}, \mathbf{vk}, C, \mathcal{C})$*: parses C as* (C', com, res)*. It returns* `reject` *if* $\overline{\mathsf{Verify}}((\mathsf{pk}, C'), com, ch, res) = 0$*. Else, it outputs whatever* $\mathsf{Rec}'(\mathsf{pk}, \mathbf{vk}, C', \mathcal{C})$ *outputs.*

Proof. The proof is obtained by replacing the algorithms corresponding to an IND-CPA PKE in either Theorem 5.1 in [10] or Theorem 2 in [9] by the algorithms corresponding to a fully distributed threshold cryptosystem. Additionally, algorithms $\mathsf{ShareDec}$ and Rec have to be adapted. We do not give full details in here, as they are cumbersome and the proof from [10, 9] can be adapted with no additional difficulties. □

4. FULLY DISTRIBUTED HELIOS WITH ARBITRARY THRESHOLD

In this section we present a detailed fully distributed version of Helios [4] with arbitrary threshold parameters, for the first time with formal proofs. As an added value, our modification also avoids ciphertext weeding while achieving ballot privacy. It takes into account the specification of Helios variants such as [15, 9].

4.1 Building blocks

Firstly, we need the fully distributed IND-CPA cryptosystem $\mathcal{D} = (\mathsf{DistKG}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{Rec})$ from Section 3.

Secondly we need a couple of NIZK proof systems, namely $\mathsf{DisjProof}^{\mathsf{upk}}(g, \mathsf{pk}, R, S)$ and $\mathsf{EqDl}(g, R, \mathsf{vk}, c)$. The first proof system allows to non-interactively prove in zero-knowledge that a ciphertext (R, S) from cryptosystem $\mathcal{D}$ encrypts g^0 or g^1. The second proof system $\mathsf{EqDl}(g, R, \mathsf{vk}, c)$ allows to prove in zero-knowledge that $\log_g \mathsf{vk} = \log_R c$ for $g, R, \mathsf{vk}, c \in \mathbb{G}$. These proof systems are described next.

Let $\mathbb{G}$ a cyclic group of order q and $g_1, g_2 \in \mathbb{G}$. We define the language $\mathcal{L}_{\mathsf{EqDl}} = \{(g_1, g_2, y_1, y_2) \mid \log_{g_1} y_1 = \log_{g_2} y_2\}$. The Chaum-Pedersen Σ-protocol for proving equality of discrete logarithm works as follows: both prover and verifier have as input $(\mathbb{G}, q, (g_1, y_1), (g_2, y_2))$; prover has a witness $x = \log_{g_1} y_1 = \log_{g_2} y_2$ to the statement as additional input. The prover chooses $r \xleftarrow{R} \mathbb{Z}_q$ and sends $com_1 = g_1^r$ and $com_2 = g_2^r$ to the verifier. The latter sends a random challenge $ch \xleftarrow{R} \mathbb{Z}_q$ to the prover who then responds with $res = r + x \cdot ch$. The verifier accepts iff $g_1^{res} = com_1 \cdot y_1^{ch}$ and $g_2^{res} = com_2 \cdot y_2^{ch}$. For this Σ-protocol, $\mathsf{Simulate}_\Sigma(g_1, g_2, y_1, y_2, ch, res)$ returns $com_1 \leftarrow g_1^{res}/y_1^{ch}$ and $com_2 \leftarrow g_2^{res}/y_2^{ch}$. The Equality of Discrete-Logarithms proof system $\mathsf{EqDl}(g_1, g_2, y_1, y_2) = (\mathsf{PrEq}, \mathsf{VerifyEq})$ for $g_1, g_2, y_1, y_2 \in \mathbb{G}$ is the non-interactive proof system associated to the language $\mathcal{L}_{\mathsf{EqDl}}$ when applying the Fiat-Shamir to the above Σ-protocol. That is, the prover sets $ch \leftarrow H(g_1, g_2, y_1, y_2, com_1, com_2)$, where $com_1, com_2 \in \mathbb{G}$ are as above. Prover's output is $(ch, res = r + x \cdot ch)$. The verifier computes $com_1 \leftarrow g_1^{res}/y_1^{ch}$ and $com_2 \leftarrow g_2^{res}/y_2^{ch}$ and returns the output of the test $ch \stackrel{?}{=} H(g_1, g_2, y_1, y_2, com_1, com_2)$.

On the other hand, $\mathsf{DisjProof}(g, \mathsf{pk}, R, S) = (\mathsf{DisjProve}, \mathsf{DisjVerify})$ is a NIZK proof that an ElGamal ciphertext $C = (R = g^r, S = \mathsf{pk}^r g^m)$ encrypts either $m = 0$ or $m = 1$. This is built using [16] and the proof system for $\mathcal{L}_{\mathsf{EqDl}}$ to show that either $(g, \mathsf{pk}, R, S) \in \mathcal{L}_{\mathsf{EqDl}}$ or $(g, \mathsf{pk}, R, S \cdot g^{-1}) \in \mathcal{L}_{\mathsf{EqDl}}$. It works as follows. Assume wlog that $(g, \mathsf{pk}, R, S) \notin \mathcal{L}_{\mathsf{EqDl}}$. First, the prover fakes a proof $(g, \mathsf{pk}, R, S) \in \mathcal{L}_{\mathsf{EqDl}}$ by choosing $(ch_0, res_0) \xleftarrow{R} \mathbb{Z}_q \times \mathbb{Z}_q$ and setting $U_0 = g^{res_0}/R^{ch_0}$

and $V_0 = \mathsf{pk}^{res_0}/S^{ch_0}$. It then sets $U_1 = g^{u_1}$ and $V_1 = \mathsf{pk}^{u_1}$ for $res_1 \xleftarrow{R} \mathbb{Z}_q$ and $c = H(g, \mathsf{pk}, R, S, U_0, V_0, U_1, V_1)$. It defines $ch_1 = ch - ch_0$ and $res_1 = u_1 + ch_1 r$. On the one hand, $\mathsf{DisjProve}(G, Y, R, S, r)$ is set to output $\pi \leftarrow (ch_0, ch_1, res_0, res_1)$. Finally, $\mathsf{DisjVerify}(g, \mathsf{pk}, R, S, \pi)$ checks if $ch_0 + ch_1 = H(g, \mathsf{pk}, R, S, \frac{G^{res_0}}{R^{ch_0}}, \frac{\mathsf{pk}^{res_0}}{S^{ch_0}}, \frac{G^{res_1}}{R^{ch_1}}, \frac{\mathsf{pk}^{res_1}}{(S \cdot g^{-1})^{ch_1}})$.

4.2 Fully Distributed Helios

For readability, we describe Helios for a single choice election (voters may simply vote 0 or 1). It can be easily generalized to elections with several candidates. We assume an authenticated channel between each voter and the bulletin board manager. In Helios, this is typically realized through password-based authentication. Formally, Fully Distributed Helios consists of eight algorithms $\mathcal{V}^{\mathsf{heliosd}} = (\mathsf{Register}, \mathsf{Setup}, \mathsf{Vote}, \mathsf{Validate}, \mathsf{VerifyVote}, \mathsf{Box}, \mathsf{Tally}, \mathsf{Verify})$ defined below:

$\mathsf{Register}(1^\lambda, id, L)$ sets $(\mathsf{upk}_{id}, \mathsf{usk}_{id}) \leftarrow (id, \emptyset)$. It adds id to L and outputs id.

$\mathsf{Setup}(1^\lambda, t, \ell)$ runs $\mathsf{DistKG}(1^\lambda, t, \ell)$ from Section 3, such that at the end, each trustee T_j knows a secret key $x_j \in \mathbb{Z}_q$. A public key for encrypting votes $\mathsf{pk} \in \mathbb{G}$ is created. A hash function $H : \{0,1\}^\star \to \mathbb{Z}_q$ is chosen. It outputs $\mathbf{pk} \leftarrow (\mathbb{G}, q, \mathsf{pk}, \mathsf{vk}_1 \leftarrow g^{x_1}, \ldots, \mathsf{vk}_\ell \leftarrow g^{x_\ell}, L, H, \mathbb{V} = \{0,1\})$, the public key of the election.

$\mathsf{Vote}(id, \mathsf{upk}, \mathsf{usk}, v)$ encrypts choice $v \in \{0,1\}$ as $\mathsf{Enc}(\mathsf{pk}, g^v) = C = (R, S)$. It computes a proof $\pi = \mathsf{DisjProve}^{id}(g, \mathsf{pk}, R, S)$ guaranteeing that the encrypted vote is 0 or 1. The ballot is defined as $b = (id, C, \pi)$.

$\mathsf{Validate}(b)$ parses the ballot b as a tuple (id, C, π). It then checks whether: (1) $id \in L$; (2) $\mathsf{DisjVerify}^{id}(g, \mathsf{pk}, C, \pi) = 1$. If any step fails, it returns `reject`; else it returns `accept`.

$\mathsf{VerifyVote}(id, \mathsf{upk}, \mathsf{usk}, b)$ returns the value of the test $b \in \mathsf{BB}$.

$\mathsf{Box}(\mathsf{BB}, b)$ if $\mathsf{Validate}(b) =$ `reject`, it halts. Else, (2) it parses $b = (id, C, \pi)$ and checks whether identity id appears in a previous entry in BB. If so, it erases that entry. (3) It adds b to BB.

$\mathsf{Tally}(\mathsf{BB}, \mathbf{sk})$ consists of two phases, a first one performed by each trustee in isolation and a second one performed interactively by a subset of trustees which outputs the outcome of the election. In the first phase, every trustee $T_j, 1 \le j \le \ell$:

(1) Runs $\mathsf{Validate}(b)$ for every $b \in \mathsf{BB}$. It outputs `invalid`, meaning invalid election, if any b is rejected.

(2) Parses each ballot $b \in \mathsf{BB}$ as (id_b, C_b, π_b).

(3) Checks whether id_b appears in a previous entry in BB. If so, it outputs `invalid`; else,

(4) Computes the atomic result ciphertext $C_\Sigma = (R_\Sigma, S_\Sigma) = (\prod_{b \in \mathsf{BB}} R_b, \prod_{b \in \mathsf{BB}} S_b)$, where $C_b = (R_b, S_b)$. It outputs its decryption shares (j, c_j, π_j) on C_Σ where $c_j \leftarrow \mathsf{ShareDec}'(\mathsf{sk}_j, \mathsf{vk}_j, (R_\Sigma, S_\Sigma))$ and π_j is a proof of knowledge of x_j s.t. $c_j = (R_\Sigma)^{x_j}$ and $\mathsf{vk}_j = g^{x_j}$ obtained via $\mathsf{PrEq}(g, \mathsf{vk}_j, R_\Sigma, c_j)$.

In the second phase, each trustee T_j:

(5) Checks whether $\mathsf{VerifyEq}(g, \mathsf{vk}_k, R_\Sigma, c_k, \pi_k) = 1$ for $k = 1, \ldots, \ell$. If not, it outputs $\mathsf{result} \leftarrow \emptyset$. Else,

(6) Computes $g^{\mathsf{result}} \leftarrow \mathsf{Rec}(\mathsf{pk}, \mathsf{vk}, (R_\Sigma, S_\Sigma), \mathcal{C})$. The tally result is obtained from g^{result} in time $\sqrt{\tau}$ for result lying in the interval $[0, \tau]$ and τ equals the number of legitimate voters.

(7) Finally $\Pi = \{(c_j, \pi_j)\}_{j=1,\ldots,\ell}$.

$\mathsf{Verify}(\mathsf{BB}, \mathsf{result}, \Pi)$

(1) Performs the checks (1-3) done in the algorithm Tally. If any of the checks fail, then it returns `reject` unless the result is itself set to `invalid`. Else,

(2) Computes the result ciphertext

$$(R_\Sigma, S_\Sigma) = \left(\prod_{b \in \mathsf{BB}} R_b, \prod_{b \in \mathsf{BB}} S_b \right)$$

It verifies the decryption shares (j, c_j, π_j), for $1 \le j \le \ell$. If any check fails, it returns `reject` unless the result is itself set to `invalid`.

(3) Checks whether $\mathsf{Rec}(\mathsf{pk}, \mathsf{vk}, (R_\Sigma, S_\Sigma, \mathcal{C})) = g^{\mathsf{result}}$, where $\mathcal{C} \subseteq \Pi$ is any $(t+1)$-subset. If all the checks pass, the algorithm returns `accept` and it returns `reject` otherwise.

Theorem 4.1 *Fully Distributed Helios has* ballot privacy *under the* DDH *assumption in the Random Oracle Model.*

PROOF. The main ideas in this proof are those used in [9]. Our only novelty is that we see that [9] easily generalises to a fully distributed setting with arbitrary threshold parameters. To do this, the first step is to show that the pair (C, π) in Fully Distributed Helios can be seen as the ciphertext of a fully distributed NM-CPA cryptosystem.

Indeed, let $\mathcal{D}^\dagger$ be the cryptosystem from Section 3.1 and let $\mathcal{D} = (\mathsf{DistKG}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{Rec})$ be the threshold cryptosystem obtained by letting $\mathsf{DistKG} = \mathsf{DistKG}^\dagger$ and

$\mathsf{Enc}(\mathsf{pk}, m)$: for $m \in \{0,1\}$ chooses $r \xleftarrow{R} \mathbb{Z}_q$ and set $(R, S) = (g^r, \mathsf{pk}^r \cdot g^m)$. Now it runs $\pi \leftarrow \mathsf{DisjProve}^{id}(g, \mathsf{pk}, R, S, r)$ and returns $((R, S), \pi)$.

$\mathsf{ShareDec}(\mathsf{sk}_j, \mathsf{vk}_j, ((R, S), \pi))$: returns `reject` if

$$\mathsf{DisjVerify}^{id}(g, \mathsf{pk}, R, S, \pi) = 0.$$

Else, it runs $(j, c_j) \leftarrow \mathsf{ShareDec}^\dagger(\mathsf{sk}_j, \mathsf{vk}_j, (R, S))$. It runs $\pi_j \leftarrow \mathsf{PrEq}(g, R, \mathsf{vk}_j, c_j, \mathsf{sk}_j)$. It outputs (j, c_j, π_j).

$\mathsf{Rec}(\mathsf{pk}, \mathsf{vk}, ((R, S), \pi), \mathcal{C})$: returns `reject` if

$$\mathsf{DisjVerify}^{id}(g, \mathsf{pk}, R, S, \pi) = 0.$$

Else, it parses every element in $\mathcal{C}$ as (j, c_j, π_j). If for any i it happens $\mathsf{VerifyEq}(g, R, \mathsf{vk}_j, c_j, \pi_j) = 0$, it outputs `reject`. Else, it runs $\mathsf{Rec}^\dagger(\mathsf{pk}, \mathsf{vk}, (R, S), \{c_{i1}, \ldots, c_{i(t+1)}\})$.

where id is a fixed voter's identity.

Lemma 4.2 *$\mathcal{D}$ is* NM-CPA *and* robust *in the Random Oracle Model under the* DDH *assumption.*

PROOF. NM-CPA security is obtained from Theorem 3.2. This is because $\mathcal{D}^{\dagger}$ is IND-CPA and because the proof system $\mathsf{DisjProof}^{\mathsf{upk}}$ is the result of applying the augmented Fiat-Shamir transform from Appendix C to Chaum-Perdersen Σ-protocol (see Appendix B). This way the requirements of Theorem 3.2 are satisified and the result holds.

Let us now briefly address robustness. The soundness of the proof system $\mathsf{EqDl}(g, R, \mathsf{vk}_j, c_j)$ implies that $\log_g \mathsf{vk}_j = \log_R c_j$ with overwhelming probability if

$$\mathsf{VerifyEq}(g, R, \mathsf{vk}_j, c_j, \pi_j) = 1$$

for any $1 \leq j \leq l$. Therefore, for any $(R, S) \in \mathbb{G}^2, \mathcal{C} = \{c_{i_1}, \ldots, c_{i_{t+1}}\}, \mathcal{C}^{\star} = \{c^{\star}_{k_1}, \ldots, c^{\star}_{k_{t+1}}\}$ such that $\mathsf{Rec}(\mathsf{pk}, (R, S), \mathcal{C}) \neq \mathtt{reject} \neq \mathsf{Rec}(\mathsf{pk}, (R, S), \mathcal{C}^{\star})$, we have that

$$\prod_{j=1,\ldots,t+1} c_{i_j}^{\lambda^{\mathcal{I}}_j} = \prod_{j=1,\ldots,t+1} (c^{\star})_{k_j}^{\lambda^{\mathcal{K}}_j} = R^x$$

and thus

$$S \cdot \left(\prod_{j=1,\ldots,t+1} c_{i_j}^{\lambda^{\mathcal{I}}_j} \right)^{-1} = S \cdot \left(\prod_{j=1,\ldots,t+1} (c^{\star})_{k_j}^{\lambda^{\mathcal{K}}_j} \right)^{-1} = m \quad (2)$$

with $\mathcal{I} = \{i_1, \ldots, i_{t+1}\}, \mathcal{K} = \{k_1, \ldots, k_{t+1}\}, \mathsf{pk} = g^x$. Finally, equation (2) is equivalent to equation

$$\mathsf{Rec}(\mathsf{pk}, \mathbf{vk}, C, \mathcal{C}) = \mathsf{Rec}(\mathsf{pk}, \mathbf{vk}, C, \mathcal{C}^{\star}) \quad \square$$

Secondly, we need to build a simulator such that if we let $(R_{L\Sigma}, S_{L\Sigma}), (R_{R\Sigma}, S_{R\Sigma})$ be the result ciphertexts in the left and right boards respectively, then when the ballot private adversary is supposed to have access to the right board, the simulator needs to make the adversary thinking that $\mathsf{Rec}(\mathsf{pk}, \mathbf{vk}, (R_{R\Sigma}, S_{R\Sigma}), \mathcal{C}) = S_{L\Sigma} \cdot (R_{L\Sigma})^{-x}$, that is the simulator needs to give out the result from tallying BB_L to $\mathcal{A}$, while $\mathcal{A}$ is tallying BB_R, without $\mathcal{A}$ noticing. Since the simulator can program the random oracle, this can be done by using the simulate algorithm of the equality of discrete logarithms sigma-protocol from Appendix B. This makes it possible for the simulator to cheat the adversary by convincing $\mathcal{A}$ to accept the result from $\mathsf{Tally}(\mathsf{BB}_L, \mathbf{sk})$, while $\mathcal{A}$ has access to the right board. $\square$

Ciphertext weeding avoided. Recall that weeding ballots is essential for proving ballot privacy. It turns out that there is no need in Fully Distributed Helios for weeding ciphertexts (as previously done in [15, 9]). Let us recall that the latter property is essential for proving ballot privacy. In effect, let $\pi^{id} = (c_0, c_1, f_0, f_1)$ and $\pi^{id'} = (c'_0, c'_1, f'_0, f'_1)$ be disjunctive Chaum-Pedersen NIZKs asserting that two given ciphertexts belonging to different voters with public credentials $id \neq id'$ are encryptions of 0 or 1 in Fully Distributed Helios. Ballot weeding consists on rejecting to add ballots $b_{id'}$ to the bulletin board such that $\pi^{id'} = \pi^{id}$ if the atomic proof π^{id} is contained in a previous ballot b_{id}. We aim at simplifying this procedure. First, notice that if the proofs $\pi^{id}, \pi^{id'}$ verify with respect to the corresponding ciphertexts $(R, S), (R', S')$ then it holds

candidates	2	5	10	20	30	50
enc+proofs	600	1197	2138	4059	6061	9617
ballot verif	110	210	390	720	1070	1730

Figure 2: Timing in miliseconds for Fully Distributed Helios

$$c_0 + c_1 = H\left(id, R, S, \frac{g^{f_0}}{R^{c_0}}, \frac{Y^{f_0}}{S^{c_0}}, \frac{g^{f_1}}{R^{c_1}}, \frac{Y^{f_1}}{(S/g)^{c_1}}\right)$$

$$c'_0 + c'_1 = H\left(id', R', S', \frac{g^{f'_0}}{(R')^{c'_0}}, \frac{Y^{f'_0}}{(S')^{c'_0}}, \frac{g^{f'_1}}{(R')^{c'_1}}, \frac{Y^{f'_1}}{(S'/g)^{c'_1}}\right)$$

The presence of the signing verification key of each voter to the hash function makes ballots weeding trivial. In fact for any pair of valid atomic disjunctive proofs $\pi^{id}, \pi^{id'}$ we have that $\Pr[\pi^{id} = \pi^{id'} \mid id \neq id'] = \Pr[H(id, \pi^{id}) = H(id', \pi^{id'}) \mid id \neq id']$ for any $\pi^{id}, \pi^{id'}$ satisfying Equation 3. In particular assume that H is a collision-resistant hash function. Then $\mathsf{Prob}[\pi^{id} = \pi^{id'} \mid id \neq id']$ is less than the probability of finding a collision on the hash function.

Replacing identities with nonces. We have seen that our fully distributed variant of Helios avoids ciphertext weeding by adding voter's identifier id to the ballot and as input to the hash function used in the NIZK proofs of ballot well-formedness. This solution might be unsatisfactory in some scenarios: for instance this practice is forbidden in presidential elections in some countries (e.g. France) and it harms any form of everlasting privacy [37, 5, 18]. We propose an alternative solution that equally avoids ciphertext weeding while preserving ballot privacy: replace id by a random nonce n_r produced anew at every invocation of the Vote algorithm. Now ciphertext weeding is avoided by implementing nonce weeding. Let τ, n_C be respectively the number of voters and candidates. The new proposed weeding has the merit that it again takes complexity proportional to τ, in contrast to complexity proportional to τn_C^2 for ciphertext weeding.

We have proposed this solution in response to a public request to avoid replay attacks done in GitHub by the main developer of Helios, and it might be incorporated in upcoming versions [2].

5. OPEN SOURCE IMPLEMENTATION OF FULLY DISTRIBUTED HELIOS

We have implemented a proof of concept of our variant of Helios, called Fully Distributed Helios, openly accessible at [29].

Table 2 shows timings, for various numbers of candidates (from 2 to 50). The first line are timings on the time needed by the voter's browser to form the ballot. The second line indicates the computation time on the server side for performing the verification tests (well-formedness of the ballot, validity of the proofs of knowledge. In practice, we use a 256-bit multiplicative subgroup of a 2048-bit prime field for ElGamal and Fiat-Shamir operations. The figures have been obtained on a computer with an Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz, running Firefox 18.

For the fully distributed threshold cryptosystem to be more practical for the parties involved, we use the following trick: each trustee P_i derives from a random seed r_i so-called "setup" keys along with the polynomial f_i of the DistKG algorithm. Setup keys consist of a signature keypair and an encryption keypair whose public parts are published with the A_{ik} at the beginning of the key distribution algorithm. Then, DistKG can be run as described earlier, using setup keys to establish the needed secure communication channels through a single server. That server can also be used to securely store all messages so that from the point of view of trustee P_i, only r_i needs to be remembered.

6. REFERENCES

[1] International association for cryptologic research. Elections page at http://www. iacr.org/elections/.

[2] `https://github.com/benadida/helios-server/issues/35`.

[3] Masayuki Abe and Serge Fehr. Adaptively secure Feldman VSS and applications to universally-composable threshold cryptography. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 317–334. Springer, 2004.

[4] Ben Adida, Olivier de Marneffe, Oliver Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections*, 2009.

[5] Myrto Arapinis, Véronique Cortier, Steve Kremer, and Mark Ryan. Practical everlasting privacy. In David A. Basin and John C. Mitchell, editors, *POST*, volume 7796 of *Lecture Notes in Computer Science*, pages 21–40. Springer, 2013.

[6] Josh Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *Proceedings of the Second Usenix/ACCURATE Electronic Voting Technology Workshop*, 2007.

[7] David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. Adapting Helios for provable ballot secrecy. In Springer, editor, *Proceedings of the 16th European Symposium on Research in Computer Security (ESORICS'11)*, volume 6879 of *Lecture Notes in Computer Science*, 2011.

[8] David Bernhard, Véronique Cortier, Olivier Pereira, and Bogdan Warinschi. Measuring vote privacy, revisited. In *19th ACM Conference on Computer and Communications Security (CCS'12)*, Raleigh, USA, October 2012. ACM.

[9] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to helios. In X. Wang and K. Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012.

[10] David Bernhard, Olivier Pereira, and Bogdan Warinschi. On necessary and sufficient conditions for private ballot submission. Cryptology ePrint Archive, Report 2012/236, 2012. `http://eprint.iacr.org/`.

[11] David Bernhard and Ben Smyth. Ballot privacy and ballot independence coincide. In Springer, editor, *Proceedings of the 18th European Symposium on Research in Computer Security (ESORICS'13)*, Lecture Notes in Computer Science, 2013.

[12] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 226–243. Springer, 2006.

[13] Colin Boyd, editor. *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*. Springer, 2001.

[14] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *Proc. IEEE Symposium on Security and Privacy*, pages 354–368, 2008.

[15] Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. In *CSF*, pages 297–311. IEEE Computer Society, 2011.

[16] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.

[17] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 1997.

[18] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. Election verifiability or ballot privacy: Do we need to choose? In Springer, editor, *Proceedings of the 18th European Symposium on Research in Computer Security (ESORICS'13)*, Lecture Notes in Computer Science, 2013.

[19] Adam M. Davis, Dmitri Chmelev, and Michael R. Clarkson. Civitas: Implementation of a threshold cryptosystem. Computing and information science technical report, Cornell University, 2008.

[20] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1987.

[21] Pierre-Alain Fouque and David Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In Boyd [13], pages 351–368.

[22] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In Jennifer Seberry and Yuliang Zheng, editors, *AUSCRYPT*, volume 718 of *Lecture Notes in Computer Science*, pages 244–251. Springer, 1992.

[23] Ryan W. Gardner, Sujata Garera, and Aviel D. Rubin. Coercion resistant end-to-end voting. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 344–361. Springer, 2009.

[24] Rosario Gennaro. Achieving independence efficiently

and securely. In James H. Anderson, editor, *PODC*, pages 130–136. ACM, 1995.

[25] Rosario Gennaro. An optimally efficient multi-authority election scheme. SCN 1996 - Security in Communication Networks Workshop, 1996. `http://www.di.unisa.it/SCN96/papers/Gennaro.ps`.

[26] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310. Springer, 1999.

[27] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure applications of Pedersen's distributed key generation protocol. In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2003.

[28] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology*, 20(1):51–83, 2007.

[29] Stéphane Glondu. Open source helios with fully distributed key generation. `https://github.com/glondu/helios-server`.

[30] Jens Groth. Evaluating security of voting schemes in the universal composability framework. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *ACNS*, volume 3089 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2004.

[31] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine, editors, *WPES*, pages 61–70. ACM, 2005.

[32] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections*, pages 37–63, 2010.

[33] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. A game-based definition of coercion-resistance and its applications. In *CSF*, pages 122–136. IEEE Computer Society, 2010.

[34] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, privacy, and coercion-resistance: New insights from a case study. In *IEEE Symposium on Security and Privacy*, pages 538–553. IEEE Computer Society, 2011.

[35] Benoît Libert and Moti Yung. Non-interactive cca-secure threshold cryptosystems with adaptive security: New framework and constructions. In Ronald Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 75–93. Springer, 2012.

[36] Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In Boyd [13], pages 331–350.

[37] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 373–392. Springer, 2006.

[38] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[39] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptology*, 15(2):75–96, 2002.

[40] Yiannis Tsiounis and Moti Yung. On the security of ElGamal based encryption. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, volume 1431 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 1998.

[41] Douglas Wikström. Universally composable DKG with linear number of exponentiations. In Carlo Blundo and Stelvio Cimato, editors, *SCN*, volume 3352 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2004.

APPENDIX

A. SIGMA PROTOCOLS

Let $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be an efficiently computable relation. Let $\mathcal{L}_R = \{Y \in \{0,1\}^* \mid \exists w : R(w,y)\}$ be the language defined by R and let Λ be such that $\mathcal{L}_R \subseteq \Lambda$ and Λ is decidable in polynomial time.

A proof system for the language $\mathcal{L}_R$ is a pair of possibly interactive algorithms $(\mathsf{Prove}, \mathsf{Verify})$ such that with overwhelming probability the interaction $\mathsf{Verify}(Y) \leftrightarrow \mathsf{Prove}(w, Y)$ ends with `accept` for every $(w, Y) \in R$.

A Σ-protocol is an interactive between a prover and a verifier in which the sender starts the interaction by sending a value *com*, the *commitment*. The verifier replies with a *challenge ch* taken uniformly at random from a given challenge set. The prover ends by sending a *response res*. The verifier checks the validity of the claimed proof by calling a deterministic algorithm $\overline{\mathsf{Verify}}_\Sigma(Y, com, ch, res)$. Basic properties for Σ-protocol are:

Special honest-verifier zero-knowledge: there is an algorithm $\mathsf{Simulate}_\Sigma$, called the simulator that takes as input a statement $Y \in \{0,1\}^*$ (that may or may not be valid), a challenge *ch* and a response *res* and outputs a commitment *com* such that $\overline{\mathsf{Verify}}_\Sigma(Y, com, ch, res) = 1$. Furthermore, if com, res are uniformly random, then (com, ch, res) is distributed as a real conversation between the prover on input (w, Y) and an honest verifier.

Zero-knowledge: The Σ-protocol is *zero-knowledge* if the previous simulator can be efficiently built and the verifier can possibly be dishonest.

Special soundness: if there is an algorithm $\mathsf{Extract}_\Sigma$ that given a statement Y and any two triples (com, ch, res) and (com, ch', res') with $ch \neq ch'$ as input, returns a witness w such that $R(w, Y)$ holds.

Unique Responses: A Σ-protocol has unique responses if for any statement Y, any commitment *com* and any challenge *ch*, there is at most one value *res* such that $\overline{\mathsf{Verify}}_\Sigma(Y, com, ch, res) = 1$.

B. FIAT-SHAMIR TRANSFORMATION AND NIZKS

Definition B.1 (Fiat-Shamir Transformation [20, 9]) *Let* $\Sigma = (\mathsf{Prove}_\Sigma, \mathsf{Verify}_\Sigma)$ *and* $H : \{0,1\}^\star \to \mathrm{Ch}$ *a hash function, where* Ch *is the challenge set for* Σ. *The Fiat-Shamir transformation of* Σ *is the non-interactive proof system* $\mathsf{FS}_H(\Sigma) = (\mathsf{Prove}, \mathsf{Verify})$ *defined as follows:*

$\mathsf{Prove}(w, Y)$: *runs* $\mathsf{Prove}_\Sigma(w, Y)$ *to obtain commitment com and computes* $ch \leftarrow H(Y, com)$. *It then completes the run of* Prove_Σ *with ch as input to get the response res and outputs the pair* (ch, res).

$\mathsf{Verify}(Y, ch, res)$: *computes* $com \leftarrow \mathsf{Simulate}_\Sigma(Y, ch, res)$ *and runs* $\overline{\mathsf{Verify}}_\Sigma(Y, com, ch, res)$.

The resulting non-interactive zero-knowledge (NIZK) proof system for the relation R is **complete**; **sound**, meaning that if $R(w, Y) = 0$ for any w, then with overwhelming probability, it holds that $0 \leftarrow \mathsf{Verify}(Y, ch, res)$ for any (ch, res); **zero-knowledge**, meaning that there exists a simulator that given a valid statement Y it outputs (com, ch, res) indistinguishable from a real proof such that Verify accepts.

C. AD-HOC FIAT-SHAMIR TRANSFORMATION

We prove that if one adds an arbitrary string (which will be the voter's credential) to the hash function, the proof-system obtained by applying the Fiat-Shamir transform remains sound. Let $(\mathsf{Prove}'_\Sigma, \mathsf{Verify}'_\Sigma)$ be a Σ-protocol for a given language $\mathcal{L}'_R \subseteq \Lambda'$. We will modify it into a Σ-protocol for the extended language $\mathcal{L}_R = \{(id, Y) | \exists w : R(w, (id, Y)\}$ which is equal to $\{0,1\}^* \times \mathcal{L}'_R$ since id is any string in $\{0,1\}^*$. And $\mathcal{L}_R \subseteq \Lambda = \{0,1\}^* \times \Lambda'$.

It is easy to see that if Λ' is decidable, so is Λ. The interaction $\mathsf{Verify}_\Sigma(id, Y) \leftrightarrow \mathsf{Prove}_\Sigma(w, (id, Y))$ is defined identically to $\mathsf{Verify}'_\Sigma(Y) \leftrightarrow \mathsf{Prove}'_\Sigma(w, Y)$. It turns out that if $(\mathsf{Prove}'_\Sigma, \mathsf{Verify}'_\Sigma)$ is special honest-verifier zero-knowledge and special sound, so is $(\mathsf{Prove}_\Sigma, \mathsf{Verify}_\Sigma)$. This is easily seen by defining

$$\begin{aligned} \mathsf{Simulate}_\Sigma((id, Y), c, f) &:= \mathsf{Simulate}'_\Sigma(Y, c, f) \\ \overline{\mathsf{Verify}}_\Sigma((id, Y), A, c, f) &:= \overline{\mathsf{Verify}}'_\Sigma(Y, A, c, f) \\ \mathsf{Extract}_\Sigma &:= \mathsf{Extract}'_\Sigma \end{aligned}$$

Applying the Strong Fiat-Shamir transformation to $(\mathsf{Prove}_\Sigma, \mathsf{Verify}_\Sigma)$ provides a non-interactive proof system $(\mathsf{Prove}, \mathsf{Verify})$ as follows: $\mathsf{Prove}(w, (id, Y))$ runs $\mathsf{Prove}_\Sigma(w, (id, Y))$ to obtain commitment A, computes $c \leftarrow H(id, Y, A)$, completes the run of Prove_Σ with c as input to get the response f and finally outputs the pair (c, f). $\mathsf{Verify}((id, Y), c, f)$ computes A from $((id, Y), c, f)$ by using the $\mathsf{Simulate}_\Sigma$ algorithm and then runs $\overline{\mathsf{Verify}}_\Sigma(Y, A, c, f)$.

Theorem C.1 *Let* $(\mathsf{Prove}'_\Sigma, \mathsf{Verify}'_\Sigma)$ *be a* Σ*-protocol that is special honest-verifier zero-knowledge and special sound and let* $(\mathsf{Prove}, \mathsf{Verify})$ *be the non-interactive proof system obtained from our ad-hoc Fiat-Shamir transformation. Then the new proof system is* zero-knowledge *and* simulation-sound extractable.

PROOF. This is a corollary of Theorem 1 in [9], since the ad-hoc Fiat-Shamir transformation can be seen as extending $(\mathsf{Prove}'_\Sigma, \mathsf{Verify}'_\Sigma)$ to a new Σ-protocol $(\mathsf{Prove}_\Sigma, \mathsf{Verify}_\Sigma)$ that takes an arbitrary dummy string id as input and then applies Fiat-Shamir. □

Theorem C.1 implies that if we let $\mathsf{DisjProof}^{id}(g, \mathsf{pk}, R, S)$ be the NIZK proof system obtained from $\mathsf{DisjProof}(g, \mathsf{pk}, R, S)$ by adding id as an input to the hash function, then the new proof system retains all the security properties hold by the original NIZK. This new proof system is essential to avoid ciphertext weeding.