A generic construction for voting correctness at minimum cost - Application to Helios

Véronique Cortier[1], David Galindo[1], Stéphane Glondu[2] and Malika Izabachène[1]

[1] LORIA - CNRS
[2] INRIA Nancy Grand Est

Abstract. Most voting schemes aim at providing verifiability: voters should be able to check that their ballots did contribute to the outcome (individual verifiability) and that the tallying authorities did their job properly (universal verifiability). Surprisingly, verifiability still does not answer a very simple and natural question: how can I be sure that the published result corresponds to the (sum of) *intended votes* of the voters? This property is called *correctness* by Juels, Catalano, and Jakobsson. Actually, even a prominent voting system like Helios does not achieve correctness in the case of a dishonest bulletin board, since it may add ballots.

We generalize the aforementioned definition of correctness to account for a malicious bulletin board (*full correctness*) and we provide a generic construction that transforms a correct voting scheme into a fully correct voting scheme. This construction simply requires to send credentials to the voters, with no additional infrastructure. We further provide a simple and natural criteria that implies voting correctness, which can then be turned into full correctness due to our construction. As an application, we build a variant of Helios that is both fully correct, verifiable and private.

Real-world elections often require threshold cryptosystems so that any t out of ℓ trustees can proceed to tallying. We describe a fully distributed (with no dealer) threshold cryptosystem suitable for Helios (in particular, suitable to partial decryption). In doing so we happen to revisit the seminal multi-authority election system from Cramer, Gennaro and Schoenmakers. Altogether, we provide the first proof of privacy, verifiability and correctness for a fully distributed Helios voting scheme (and its enhanced version with credentials), together with its detailed description. This also implies, to our knowledge, the first formal proofs of privacy, verifiability and correctness for the scheme by Cramer *et al.* Last but not least, we provide an open source implementation of our variant of Helios.

Keywords: voting protocols, Helios, correctness, full correctness, verifiability, ballot privacy, fully distributed threshold 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* [16] 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 [14, 7, 9]. The remaining question is whether the result of an election run through Helios does correspond to the votes cast by the voters. Juels *et al* [30] have introduced an important distinction between verifiability and correctness. Verifiability ensures that there is a public algorithm that can check the outcome of the tally, that is, that can check that the authorities behaved as expected. However, this says nothing about the fact that the outcome of the election does correspond to the votes intended by the honest voters (plus possibly some votes from the dishonest voters). This second property, called *correctness*, appears only in [30], the extended version of [29]. It ensures in particular that even malicious ballots correspond to valid votes (for example a malicious voter shall not be able to vote 100 if the two allowed values are 0 and 1).

Is Helios correct? According to JCJ's definition [30], Helios is correct, although this has never been proved. However, JCJ's definition assumes the bulletin board to be honest: an attacker may cast dishonest ballots on the behalf of dishonest voters but no extra ballots may be added nor deleted. This means for example that the result of the election of the 2012 board of the IACR can be trusted only under the assumption that the election server was neither dishonest nor attacked, during the whole duration of the election. This is a rather unsatisfactory assumption, since adding a few extra ballots may easily change the outcome of an election. In the case of Helios, this is mitigated by the fact that voters' identities are public. If the bulletin board adds ballots, it has to tell which voters are supposed to have cast these ballots. Thus hopefully, these voters should notice that the server wrongly cast ballots on their names and would complain. Such complaints are however not guaranteed since absentees typically do not care much about the election. Things may be even worse. In some countries (like France), whether someone voted or not is a private information. It is therefore forbidden to publicly reveal the identities of the voters who cast a vote. Moreover, publishing voters identities compromises privacy in the future: once the keys will be broken (say in 20 years), everyone will learn the vote of each voter. A simple alternative consists in removing the disclosure of voters' identities. This variant of Helios remains perfectly practical and of course still preserves ballot privacy. But it then becomes completely straightforward for a corrupted bulletin board to add as many ballots as needed.

Correctness against a malicious box. We first provide a generalization of the definition of correctness of Juels *et al* [30] that accounts for a malicious bulletin board. Intuitively, a voting scheme is correct if the result corresponds to the votes of

- all honest voters that have checked that their vote was cast correctly (e.g. in Helios, this simply amounts into checking that the encrypted vote appears on the bulletin board);
- at most n valid votes where n is the number of corrupted voters (i.e. the attacker may only use the corrupted voters to cast valid votes);
- a subset of the votes cast by honest voters that did not check their vote was cast correctly (in practice, many voters do not perform any check).

As in [30], this definition requires the tally function to admit *partial tallying* (that is, it is possible to compute the tally by blocks and then retrieve the final result).

Our first main contribution is a generic construction that transforms any correct voting scheme that assumes both the registration authority and the bulletin board honest, into a correct voting scheme under the weaker trust assumption that the registration authority and the bulletin board are not *simultaneously* dishonest. We refer to this as transforming correctness w.r.t. a honest bulletin board to correctness w.r.t. a malicious bulletin board. Following the same terminology, we stress that our transformation also turns ballot privacy and verifiability w.r.t. a honest bulletin board into ballot privacy and verifiability w.r.t. a malicious bulletin board.

Correctness against a malicious bulletin board cannot come from free. In Helios-like protocols, the bulletin board is the only authority that controls the right to vote. It may therefore easily stuff itself, that is, it may easily add ballots. To control the bulletin board, it is necessary to consider an additional authority. Our solution consists in providing each voter with a private credential (actually a signing key) that has a public part (the verification key). The set of all public credentials is public and, in particular, known to the bulletin board. Then each voter simply signs his ballot with his private credential. Note the association between a public credential and the corresponding voter's identity does not need to be known and actually, should not be disclosed to satisfy e.g. the French requirements regarding voting systems.

The advantage of our solution is to be simple: the additional authority is only responsible for the generation of credentials and their distribution to the voters. Once it is done, it can erase all its memory. There is no need for an additional server. It is also possible to have the registration authority to generate the credentials off-line and to distribute them using a non-digital channel, e.g. snail mail (see details on our implementation in Section 6). This minimizes the risk of Internet-based attacks against the registration authority.

A criteria for correctness. Since proving correctness, even with an honest bulletin board, may not be easy, we provide a simple and natural criteria for correctness. We show that any *consistent* and *accurate* voting protocol is correct (w.r.t. an honest bulletin board). Consistency accounts for the natural property that the tally of just honestly casted ballots should always yield the expected result (typically the sum of the votes). Accuracy ensures that any ballot (possibly dishonest) that passes the verification check (e.g. valid proof, well-formedness of the ballots) corresponds to a valid vote, and that for any ballot box the output of the tallying algorithm always satisfies the verification test. Our criteria is satisfied in particular by Helios and we expect it to be satisfied by many existing voting protocols.

Application to Helios. Applying our generic construction to Helios we obtain a voting scheme, that we name as Helios with Credentials (Helios-C), which is correct against a malicious bulletin board, verifiable and private. We go further by providing the first detailed proof of security for Helios without trusted dealer and arbitrary threshold, for both correctness and privacy. 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.

There is an abundant literature regarding dlog-based threshold cryptosystems (see e.g. [19, 36, 38, 11, 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 [24, 40, 3, 26]. 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 [37], which was shown in [24] to output biased public keys in the presence of active adversaries, those stronger DKG protocols are more involved and expensive. In order to build Helios upon the lightest and simplest fully distributed semantically secure threshold cryptosystem, we are able to show in Section 4 that Pedersen's DKG applied to ElGamal can be proven semantically secure under the Decision Diffie-Hellman assumption. In [25, 3, 26] a related result was shown for Schnorr signatures in the Random Oracle Model. It turns out that the same techniques can be safely used in more demanding scenarios, where the adversary solves a decisional problem (in contrast to a search problem, as in the case of digital signatures) and in the standard model (in contrast to the random oracle model).

Going back to Helios, we note that the currently implemented version of Helios is subject to an attack against privacy [14]: 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. The situation is worse in scenarios where the authorities aim at reusing the election public key for further elections. Then, to ensure ballot privacy, [10] shows that ciphertext weeding must also be done with respect to bulletin boards produced in older elections! 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.

Interestingly, our variant of Helios avoids ciphertext weeding. Due to the existence of credentials, we can provably avoid the submission of duplicated ballots by simply adding the credential as an additional input to the random oracle in the non-interactive zero-knowledge (NIZK) proofs for ciphertext well-formedness used in Helios (wich in turn are obtained via the Fiat-Shamir transformation [18]). Checking for duplicates is therefore alleviated to checking that the same credential does not appear twice in the board. Thus the complexity of the test depends only on the size of the number of voters, and it had to be implemented in previous versions of Helios anyway. This solution allows Helios-C to scale for larger elections while attaining correctness. Our detailed description of Helios (with credentials) uses a fully-distributed threshold cryptosystem with arbitrary threshold parameters. For privacy, we built upon the privacy proof of [9, 10]. In fact we show that that the generic constructions for NM-CPA public key encryption in [9, 10] from IND-CPA encryption and Fiat-Shamir non-interactive proofs easily extend to fully-distributed threshold cryptosystems. Additionally we need to prove that adding the credential as an additional input to the random oracle in the Fiat-Shamir transformation maintains its security properties.

We have implemented our variant Helios-C. Our implementation includes the suppression of ciphertext weeding and the fully distributed threshold cryptosystem. It demonstrates that the overhead of our additional signature is very limited on the client side (from 5 to 20% depending on the number of candidates and the overhead decreases with the number of candidates). The implementation is openly accessible [2]. To our knowledge this is the first public implementation of a variant of Helios using a fully distributed threshold cryptosystem.

Related work. Helios [4] is in fact an implementation and simplification of a seminal work on multi-authority voting systems by Cramer, Gennaro and Schoenmakers [16]. The fully-distributed IND-CPA scheme that we propose and analyze in Section 4 was already proposed in [23, 16], but no formal proof of semantic security was given. Our fully-distributed version of Helios resembles almost exactly the voting scheme given in [16]. Thus our work can in particular be seen as a thorough examination of the scheme by Cramer *et al.* 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. [16] avoids ballots duplication by adding the voter's identity id to the hash function in the NIZK proofs. This technique is sound, as proven in [22, 28] and indirectly here, but as we discussed earlier, it is not an option when the voters' identities can not be published together with their ballots.

In addition to Helios, several private and verifiable voting schemes have been proposed, including e.g. Civitas [13] and FOO [20]. Helios is currently the most usable (and used) remote voting scheme in practice. The notion of privacy has been extensively studied. Several privacy definitions for voting schemes have been proposed, from ballot privacy [28, 33, 7–9] to coercion-resistance [29, 21, 32] and applied to voting schemes: Civitas has been shown to be coercion-resistant [13], while Helios has been shown to ensure ballot and vote privacy [14, 7, 9, 8]. While many academic voting schemes aim at being private, correctness and verifiability have deserved less attention. Küsters *et al.* have put forward general definitions of verifiability and accountability in [31, 33, 34] that we believe capture correctness as a particular case. In particular, the standard version of Helios is proved to be verifiable for honest bulletin boards [34]. A definition of vote correctness can be found in [30]. Regarding voting systems with arbitrary threshold parameters, [17] describes an implementation of Civitas with a fully-distributed threshold cryptosystem using the distributed key generation protocol by Gennaro *et al.* [24]. Compared to ours, the latter key generation protocol is twice as complex.

2 Syntax, correctness, verifiability 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*[3]: 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 thorough the paper. A voting protocol $\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{VerifyVote}, \mathsf{Validate}, \mathsf{BallotBox}, \mathsf{Tally}, \mathsf{Verify})$ consists of eight algorithms whose syntax is as follows:

[3] This entity was not consider in [29, 30], as the bulletin board was honest by default.

$\mathsf{Setup}(1^\lambda, \ell)$ 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 $\mathbf{sk}$. 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 $\mathbf{pk}$. Hence every algorithm in the voting protocol has access to L. Of course, if no credentials are needed, L is empty and $\mathsf{Register}(1^\lambda, id)$ is void.

$\mathsf{Vote}(id, \mathsf{upk}, \mathsf{usk}, v)$ 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. The identifier id can be seen as an optional input.

$\mathsf{VerifyVote}(\mathsf{BB}, id, \mathsf{upk}, \mathsf{usk}, b)$ 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.

$\mathsf{BallotBox}(\mathsf{BB}, b)$ 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 the bulletin board $\mathsf{BB} = \{b_1, \ldots, b_\tau\}$ 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, $\mathsf{result} = \mathtt{invalid}$, 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`.

We focus on voting protocols that admit *partial tallying*. This is not a limitation of our results but we need it to be able to define a security property called *correctness* (as discussed in the Introduction). The partial tallying property is specified by two natural requirements usually satisfied in practice. Firstly, the result function $\rho : \mathbb{V}^\tau \to \mathbb{R}$ for $\mathcal{V}$ must admit *partial counting*, namely $\rho(S_1 \cup S_2) = \rho(S_1) \star_{\mathbb{R}} \rho(S_2)$ for any two lists S_1, S_2 containing sequences of elements $v \in \mathbb{V}$ and where $\star_{\mathbb{R}} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a commutative operation. For example, the standard result function that counts the number of votes per candidate admits partial counting.

Secondly, the algorithm Tally must admit *partial tallying*. That is, let $(\mathsf{result}_1, \Pi_1) \leftarrow \mathsf{Tally}(\mathsf{BB}_1, \mathbf{sk})$ and $(\mathsf{result}_2, \Pi_2) \leftarrow \mathsf{Tally}(\mathsf{BB}_2, \mathbf{sk})$. Then, $(\mathsf{result}, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_1 \cup \mathsf{BB}_2, \mathbf{sk})$ is such that $\mathsf{result} = \mathsf{result}_1 \star_{\mathbb{R}} \mathsf{result}_2$, whenever both result_1 and result_2 are different from invalid.

Next we define the minimal procedural requirement, called *consistency*, that every such electronic 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 *consistent* if for $i \in \{1, \ldots, \tau\}$ it holds:
(1) $\mathsf{BallotBox}(\mathsf{BB}, b_i) = \mathsf{BB} \cup \{b_i\}$ if id_i did not previously cast any ballot, where we let $b_i \leftarrow \mathsf{Vote}(id_i, \mathsf{usk}_i, \mathsf{upk}_i, v_i)$ for some $v_i \in \mathbb{V}$; (2) $\mathsf{Validate}(b_i) = \mathtt{accept}$ and $\mathsf{VerifyVote}(\mathsf{BallotBox}(\mathsf{BB}, b_i), \mathsf{usk}_i, \mathsf{upk}_i, b_i) = \mathtt{accept}$; (3) $\mathsf{Tally}(\{b_1, \ldots, b_\tau\}, \mathbf{sk})$ outputs $(\rho(v_1, \ldots, v_\tau), \Pi)$; (4) $\mathsf{Verify}(\{b_1, \ldots, b_\tau\}, \rho(v_1, \ldots, v_\tau), \Pi) = \mathtt{accept}$. 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 consistency can only be guaranteed if at least $t + 1 > \lceil \ell/2 \rceil$ trustees cooperate to compute Tally.

2.1 Correctness against a malicious board

Next we give a definition for a security property for voting schemes called *correctness*. This security property has already been addressed, for instance in in [29, 30]. Our novelty is that, in contrast to previous work, we assume the bulletin board to be possibly dishonest. Note that in this case, as shown in the introduction, and as we will recall later, the bulletin board may try to add ballots to the bulletin board in the name of voters who did never cast a vote. Of course, a correct protocol should forbid or at least detect such a malicious behavior.

```
Experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{corr-malicious}}(\lambda)$
(1) $\mathsf{Init}(\lambda) \rightarrow (\mathbf{pk}, \mathbf{sk})$
(2) $\mathcal{A}^{\mathcal{O}\mathsf{corruptU}(), \mathcal{O}\mathsf{vote}()} \rightarrow (\mathsf{BB}, \mathsf{HVote}, \mathsf{Event})$
(3) $(\mathsf{result}, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathbf{sk})$
(4) if $\mathsf{result} = \mathtt{invalid}$ $\mathtt{return}$ $0$
    let $\{(id_1^E, v_1^E), \ldots, (id_{n_E}^E, v_{n_E}^E)\} = \mathsf{Event}$
    if
(5) $\mathsf{Verify}(\mathsf{BB}, \mathbf{pk}, \mathsf{result}, \Pi) = \mathtt{accept}$ and
(6) $\exists\, (id_1^A, v_1^A), \ldots, (id_{n_A}^A, v_{n_A}^A) \in \mathsf{HVote}$ and
(7) $\exists\, v_1^B, \ldots, v_{n_B}^B \in \mathbb{V}$ s.t. $0 \leq n_B \leq |\mathcal{CU}|$ and
(8) $\mathsf{result} = \rho\left(\{v_i^E\}_{i=1}^{n_E}\right) \star_{\mathbb{R}} \rho\left(\{v_i^A\}_{i=1}^{n_A}\right) \star_{\mathbb{R}} \rho\left(\{v_i^B\}_{i=1}^{n_B}\right)$
    $\mathtt{return}$ $0$
    else $\mathtt{return}$ $1$
```

Fig. 1. Correctness against malicious bulletin board

Attacker model. The adversary against correctness is allowed to corrupt trustees, users and bulletin board. More precisely, for the bulletin board, we let the adversary replace or delete any ballot. The adversary only looses control on the bulletin board once the voting phase ends and before the tallying starts. Indeed, at this point it is assumed that everyone has the same view of the public BB.

Let L be the set of all public credentials and $\mathcal{U}$ the set of all pairs of public and secret credentials. $\mathcal{CT}$ and $\mathcal{CU}$ denote the set of corrupted trustees and corrupted users respectively. The adversary can query oracles $\mathcal{O}\mathsf{register}$, $\mathcal{O}\mathsf{vote}$ and $\mathcal{O}\mathsf{corruptU}$ 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 the 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 $\mathcal{CU} = \mathcal{CU} \cup \{(id, \mathsf{upk}_{id})\}$.
$\mathcal{O}\mathsf{vote}(id, v)$: if $(id, *, *) \notin \mathcal{U}$ or $(id, *) \in \mathcal{CU}$ or $v \notin \mathbb{V}$, it aborts; else it returns $b = \mathsf{Vote}(\mathsf{pk}, \mathsf{upk}_{id}, \mathsf{usk}_{id}, v)$ and updates $\mathsf{HVote} = \mathsf{HVote} \cup \{(id, v)\}$;

In the correctness game we maintain two lists HVote and Event. The first one contains all pairs (id, v) which have been queried to $\mathcal{O}\mathsf{vote}$; the second one consists of all pairs (id, v) such that (1) $(id, v) \in \mathsf{HVote}$; and (2) the voter's verification algorithm was successfully run, *i.e.* $\mathsf{VerifyVote}(\mathsf{BB}, \mathsf{upk}_{id}, \mathsf{usk}_{id}, b) = \mathtt{accept}$.

We define next a procedure Init which helps us describing the view of the correctness adversary:

- $\mathsf{Init}(\lambda, \ell)$: The objects $L, \mathsf{HVote}, \mathsf{Event}, \mathsf{BB}$ 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 $L, \mathcal{U}$ are defined. In a second phase, the setup algorithm is run and $(\mathbf{pk}, \mathbf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, t, \ell)$. 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.

Correctness Any voting scheme should guarantee that the result output by $\mathsf{Tally}()$ counts the actual votes cast by honest voters. In particular an adversary controlling a subset of eligible voters and a subset of trustees, should not be able to alter the output of the tally so that honest votes are not counted in result. In our case, this shall hold even if the bulletin board is malicious. More precisely, correctness shall guarantee that result as output by the algorithm Tally actually counts 1) votes cast by honest voters who *checked* that their ballot appeared in the bulletin board once the voting phase is over; 2) a subset of the votes cast by honest voters who *did not check* this. Indeed it can not be ensured that result counted their votes but it might still be the case that some of their ballots were not deleted by the adversary. 3) For voters controlled by the adversary, correctness only guarantees that the adversary cannot cast more ballots than users were corrupted, and that ballots produced by corrupted voters contribute to result only with admissible votes $v \in \mathbb{V}$. The correctness security game is formally given by experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{corr-malicious}}$ in Figure 1. We say that a voting protocol $\mathcal{V}$ is *correct against a malicious board* or *fully correct* if there exists a negligible function $\nu(\lambda)$ such that, for any PPT adversary $\mathcal{A}$,

$$\mathsf{Succ}^{\mathsf{corr-malicious}}(\mathcal{A}) = \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{corr-malicious}}(\lambda) = 1\right] < \nu(\lambda)$$

The adversary modelling a honest bulletin board is obtained from the previous adversary by: 1) not allowing it to delete ballots from BB; 2) letting it cast only ballots b that pass $\mathsf{Validate}()$. The experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{corr-honest}}$ defining correctness against a honest bulletin board is obtained from $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{corr-malicious}}$ by: letting $\mathsf{Event} := \mathsf{HVote}$, since ballots output by $\mathcal{O}\mathsf{vote}()$ can not be erased from BB; asking that every ballot b in BB passes the validity test. We say that a voting protocol $\mathcal{V}$ is *correct against a honest board* or, simply, *correct* if there exists a negligible function $\nu(\lambda)$ such that, for any PPT adversary $\mathcal{A}$,

$$\mathsf{Succ}^{\mathsf{corr-honest}}(\mathcal{A}) = \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{corr-honest}}(\lambda) = 1\right] < \nu(\lambda)$$

2.2 Verifiability

Verifiability of a voting protocol ensures that the tally of an election is unique. In other words, two different tallies $\mathsf{result} \neq \mathsf{result}'$ can not be accepted by the verification algorithm, even if all the players in the system, except for the registration authority, are malicious. The goal of the adversary against verifiability is to output a public key $\mathbf{pk}$, a list of legitimate public credentials, a bulletin board BB, and two tallies $\mathsf{result} \neq \mathsf{result}'$, and corresponding proofs of valid tabulation Π and Π', such that both pass verification.

```
Experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ver}}(\lambda)$
(1) $(\mathbf{pk}, \mathsf{BB}, \mathsf{result}, \Pi, \mathsf{result}', \Pi') \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{register}(), \mathcal{O}\mathsf{corruptU}()}$
(2) if $\mathsf{result}' \neq \mathsf{result}$ and
(3) $\mathsf{Verify}(\mathsf{BB}, \mathsf{result}, \Pi) = \mathsf{Verify}(\mathsf{BB}, \mathsf{result}', \Pi') =$ accept
    return 1 else return 0
```

Fig. 2. Verifiability

We define verifiability by the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ver}}$ in Figure 2, where oracles $\mathcal{O}\mathsf{register}(), \mathcal{O}\mathsf{corruptU}()$ are defined as in Section 2.1. A voting protocol $\mathcal{V}$ is *verifiable* if there is a negligible function $\nu(\lambda)$ s.t. for any PPT adversary $\mathcal{A}$,

$$\mathsf{Succ}^{\mathsf{ver}}(\mathcal{A}) = \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ver}}(\lambda) = 1\right] < \nu(\lambda)$$

This definition is an adaptation of the definition of [30] to better cope with the cases where $\mathsf{Tally}()$ might be ill-defined.

2.3 Ballot privacy

Ballot privacy requires that any coalition of at most t trustees cannot infer information from ballots cast by honest voters to the bulletin board. We extend the simulation-based game definition from [9] to voting protocols with credentials and corrupted bulletin board. Formally, we consider two experiments: one in which tally is left to the 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 $\beta = 1$. The board $\mathsf{BB}_{1-\beta}$ is invisible to the adversary. The latter is given access to oracles $\mathcal{O}\mathsf{register}, \mathcal{O}\mathsf{corruptU}$ as defined in Section 2.1. We define two procedures Init and Main that help defining the view of the adversary in the ballot privacy game. Additionally the privacy adversary has access to a left-or-right oracle $\mathsf{OvoteLR}(id, v_0, v_1)$ defined as follows:

- $\mathsf{OvoteLR}(id, v_0, v_1)$: if id was previously queried, or upk_{id} does not appear in $\mathcal{U}$, 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 $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \cup \{\mathsf{Vote}(id, v_1)\}$.
- $\mathsf{Init}(\lambda, \ell)$: 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 $L, \mathsf{HVote}, \mathsf{Event}$ 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 $L, \mathcal{U}$ are defined. 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 $\Pi \leftarrow \mathsf{SimTally}(\mathsf{BB}_0, \mathsf{BB}_1, \mathbf{pk}, \mathtt{info})$, where $\mathtt{info}$ contains any information known to the challenger. The output is $(\mathsf{result}, \Pi, \beta')$, 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 $\beta = \beta'$
```

Fig. 3. Ballot privacy

As discussed in [8], if $\beta = 1$ then the announced result will not match the ballots on the board. Ballot privacy asks that an adversary cannot distinguish ballots containing the true votes of the honest users from ballots containing a fixed, constant vote.

s The main difference between our privacy definition and previous ones [5, 14, 9] is that in our case the adversary has full control on the bulletin board. In particular, it can delete any votes added by the challenger via the left-right voting oracle. Formally, we say that a voting protocol $\mathcal{V}$ has *ballot privacy against a malicious board* if there exists an efficient simulator $\mathsf{SimTally}$ such that any PPT algorithm cannot 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 3.

Ballot privacy against an honest board is defined similarly, except that the adversary is asked to produce a ballot box BB such that $\mathsf{Validate}(b) = 1$ for every $b \in \mathsf{BB}$, and it can not delete ballots output by the left-right voting oracle.

2.4 Accuracy

We introduce a new property for voting protocols that we call *accuracy*. We say that a voting protocol $\mathcal{V}$ has *accuracy* (equivalently it is accurate) if the following two properties hold with overwhelming probability:

1. for any ballot b it holds that $[\mathsf{Validate}(b) = \mathtt{accept} \Leftrightarrow \mathsf{Tally}(\{b\}, \mathbf{sk}) = \rho(v)$ for some $v \in \mathbb{V}]$;
2. for any bulletin board BB it holds $\mathsf{Verify}(\mathsf{BB}, \mathsf{Tally}(\mathsf{BB}, \mathbf{sk})) = \mathtt{accept}$.

Condition 1 reflects the natural requirement that even a dishonest ballot should correspond to an admissible vote. In Helios-like protocols, this is typically ensured by requiring the voter to produce a proof that the encrypted vote belongs to $\mathbb{V}$. Condition 2 guarantees that the proof produced by the tally procedure passes the verification test. In practice, this property is usually held.

3 Sufficient conditions for voting correctness

In this section we identify sufficient conditions for electronic voting correctness, both for honest and corrupted bulletin boards.

3.1 A sufficient condition for voting correctness. Case I: honest bulletin board

Consistency and accuracy suffice to ensure correctness. Since the two properties of consistency and accuracy are simple and easy to check, we believe that this result may often ease the proof of correctness.

Theorem 1. *Let $\mathcal{V}$ be a consistent and accurate voting protocol that admits partial tallying. Then $\mathcal{V}$ satisfies correctness.*

The proof is given in Appendix **??**.

3.2 A sufficient condition for voting correctness. Case II: corrupted bulletin board

As discussed in the introduction, a correct protocol like Helios may still admit ballot stuffing, e.g. from the bulletin board itself. We provide a generic construction that protects any correct voting scheme against a malicious board. Let $\mathcal{V} = (\mathsf{Register}', \mathsf{Setup}', \mathsf{Vote}', \mathsf{VerifyVote}', \mathsf{Validate}', \mathsf{BallotBox}', \mathsf{Tally}', \mathsf{Verify}')$ be a voting protocol (possibly without credentials). Let $\mathcal{S} = (\mathsf{SKey}, \mathsf{Sign}, \mathsf{SVerify})$ be a signature scheme (for the syntax and security properties of a digital signature scheme we refer to [27]). Let us consider the following voting protocol with credentials $\mathcal{V}^{\mathsf{cred}} = (\mathsf{Register}, \mathsf{Setup}, \mathsf{Vote}, \mathsf{VerifyVote}, \mathsf{Validate}, \mathsf{BallotBox}, \mathsf{Tally}, \mathsf{Verify})$ obtained from $\mathcal{V}$ and $\mathcal{S}$:

$\mathsf{Register}(1^\lambda, id)$ first runs $(\mathsf{upk}', \mathsf{usk}') \leftarrow \mathsf{Register}'(1^\lambda, id)$. It then runs $(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{SKey}(1^\lambda)$, and adds $((\mathsf{upk}', \mathsf{upk}), (\mathsf{usk}', \mathsf{usk}))$ to the credential key pair for id. It updates the list of credentials' public keys as $L \leftarrow L \cup \{(\mathsf{upk}', \mathsf{upk})\}$. Let us denote $\mathbf{upk} \leftarrow (\mathsf{upk}', \mathsf{upk})$ and $\mathbf{usk} \leftarrow (\mathsf{usk}', \mathsf{usk})$.

$\mathsf{Setup}(1^\lambda, t, \ell)$ runs $(\mathbf{pk}', \mathbf{sk}') \leftarrow \mathsf{Setup}'(1^\lambda, t, \ell)$ and sets $\mathbf{pk} \leftarrow (\mathbf{pk}', L)$. It outputs $(\mathbf{pk}, \mathbf{sk}')$. Recall that $\mathbf{pk}$ is an implicit input to the remaining algorithms.

$\mathsf{Vote}(id, \mathbf{upk}, \mathbf{usk}, v)$ starts by running $\alpha \leftarrow \mathsf{Vote}'(id, \mathsf{upk}', \mathsf{usk}', v)$. It computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{usk}, \alpha)$ and returns a ballot $b \leftarrow (\mathsf{upk}, \alpha, \sigma)$.

$\mathsf{VerifyVote}(\mathsf{BB}, id, \mathbf{upk}, \mathbf{usk}, b)$ verifies that the ballot b appears in BB once the bulletin board is closed. Let $\mathsf{BB} = \{(\mathsf{upk}_1, \alpha_1, \sigma_1), \ldots, (\mathsf{upk}_\tau, \alpha_\tau, \sigma_\tau)\}, \mathsf{BB}' = \{\alpha_1, \ldots, \alpha_\tau\}$. If there is an entry of the form $(\mathsf{upk}, \alpha, \sigma)$ in BB, then it runs $\mathsf{VerifyVote}(\mathsf{BB}, id, \mathsf{upk}', \mathsf{usk}', \alpha)$. Otherwise, it outputs `reject`.

$\mathsf{Validate}(b)$ it parses $b = (\mathbf{upk}, \alpha, \sigma)$. If $\mathsf{SVerify}(\mathsf{upk}, \alpha, \sigma) = \texttt{reject}$ it outputs `reject`. Else, it outputs $\mathsf{Validate}'(\alpha)$.

$\mathsf{BallotBox}(\mathsf{BB}, b)$ it runs $\mathsf{Validate}(b)$ and lets BB unchanged if b was rejected. Else, it parses $b = (\mathsf{upk}, \alpha, \sigma)$, and lets BB unchanged if there is no entry of the form $(*, \mathsf{upk})$ in L. Else, it removes any previous entry in BB containing $(*, \mathsf{upk})$, and updates $\mathsf{BB} \leftarrow \mathsf{BB} \cup \{b\}$.

$\mathsf{Tally}(\mathsf{BB}, \mathbf{sk})$ starts by every trustee checking whether BB is well-formed. We say BB is well-formed if: every upk appearing in BB appears only once; every upk showing up in BB belongs to L; $\mathsf{Validate}(b) = \texttt{accept}$ for every $b \in \mathsf{BB}$. If any of these checks fails, trustees halt and output `invalid`. Else trustees run $\mathsf{Tally}'(\mathsf{BB}', \mathbf{sk})$, where $\mathsf{BB}' = \{\alpha_1, \ldots, \alpha_\tau\}$ if $\mathsf{BB} = \{(\mathsf{upk}_1, \alpha_1, \sigma_1), \ldots, (\mathsf{upk}_\tau, \alpha_\tau, \sigma_\tau)\}$.

$\mathsf{Verify}(\mathsf{BB}, \mathsf{result}, \varPi)$ starts by checking whether BB is well-formed. If not, it outputs `accept` if $\mathsf{result} = \texttt{invalid}$; else it outputs `reject`. Else, it runs $\mathsf{Verify}'(\mathsf{BB}', \mathsf{result}, \varPi)$, where $\mathsf{BB} = \{(\mathsf{upk}_1, \alpha_1, \sigma_1), \ldots, (\mathsf{upk}_\tau, \alpha_\tau, \sigma_\tau)\}$ and $\mathsf{BB}' = \{\alpha_1, \ldots, \alpha_\tau\}$.

Our construction protects against a malicious bulletin board, in the sense that it updates correctness (against a honest board to correctness) to full correctness (against a malicious board).

Theorem 2. *Let $\mathcal{V}$ be a voting protocol that satisfies correctness and admits partial tallying. Let $\mathcal{S}$ be an existentially unforgeable signature scheme. Then $\mathcal{V}^{\mathsf{cred}}$ satisfies full correctness.*

Additionally our transformation preserves verifiability and ballot privacy. It shall be noted that any voting scheme inside our framework is verifiable, alternatively ballot private, for a malicious bulletin board iff it satisfies the corresponding property against a honest board.

Theorem 3. *If $\mathcal{V}$ satisfies verifiability, then $\mathcal{V}^{\mathsf{cred}}$ preserves verifiability.*

Theorem 4. *If $\mathcal{V}$ satisfies privacy then $\mathcal{V}^{\mathsf{cred}}$ satisfies privacy.*

Using the fact that $\mathcal{V}$ is consistent, it is easy to see that $\mathcal{V}^{\mathsf{cred}}$ is consistent. The proofs of this fact and of the above theorems are to be found in Appendix A.

4 A Fully Distributed NM-CPA Threshold Cryptosystem from Decision Diffie-Hellman

Our aim is to build voting systems 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*. 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. Eventually we 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 $1\le t\le\ell$, 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 $\mathsf{Rec}(\mathsf{pk},C,\mathcal{C})\neq\texttt{reject}\neq\mathsf{Rec}(\mathsf{pk},C,\mathcal{C}')$ it holds that $\mathsf{Rec}(\mathsf{pk},C,\mathcal{C})=\mathsf{Rec}(\mathsf{pk},C,\mathcal{C}')$.

IND-CPA security: (informal) Any adversary with knowledge of at most t private keys amongst the set $\mathbf{sk}=\{\mathsf{sk}_1,\ldots,\mathsf{sk}_\ell\}$ can not distinguish between encryptions $\mathsf{Enc}(\mathsf{pk},m_0)$ and $\mathsf{Enc}(\mathsf{pk},m_1)$ for any two admissible messages $m_0\neq m_1$ such that $|m_0|=|m_1|$.

IND-CPA Fully Distributed (t, n)-Threshold Cryptosystem from DDH in the Standard Model There are few electronic voting proposals based on threshold cryptosystems that make explicit 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 exception is the work by Juels, Catalano and Jakobsson [30], whose idea has been detailed and implemented by Davis, Chmelev and Clarkson [17]. They propose to use a computationally secure distributed simulation of the process of generating a public key $Y = g^x, x \overset{R}{\leftarrow} \mathbb{Z}_q$. In particular, they propose the distributed key generation scheme by Gennaro, Jarecki, Krawczyk and Rabin in [26].

Here we give a full description and proof of a fully distributed cryptosystem compatible with Helios, which is in fact obtained by coupling Pedersen's distributed key generation protocol (DKG) [37] with ElGamal. This cryptosystem has been previously described in [23, 16]. However we shall notice that still a full proof of semantic security is needed and is lacking. The latter is somehow unsatisfactory, since as it is common wisdom in cryptographic protocols, the devil is in the details. In fact Pedersen's protocol is shown in [26] to not always output uniformly random public keys, so in principle it does not seem to be a good choice to distribute ElGamal while retaing the Decision Diffie-Hellman (DDH) assumption (we refer to [39] 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 [25, 26]. 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 [26] 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 [23, 16, 17]:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ proceeds as follows:

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 $i = 1, \ldots, \ell$:
$$g^{s_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k} \tag{1}$$
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 $r \overset{R}{\leftarrow} \mathbb{Z}_q$.
$\mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{vk}_i, C)$ 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 5. *The above scheme is* IND-CPA *secure under the* DDH *assumption.*

The proof is given in Appendix B. We know from previous work that ballot private Helios-like voting protocols are tightly related to NM-CPA cryptosystems [9, 10]. In Appendix G we show how to improve the previous IND-CPA ElGamal-based threshold cryptosystem into a robust and NM-CPA threshold cryptosystem that encrypts bit-by-bit. To do this, we use the NIZK proof system *à la* Fiat-Shamir $\mathsf{DisjProof}^{\mathsf{upk}}(g, \mathsf{pk}, R, S) = (\mathsf{DisjProve}^{\mathsf{upk}}$, $\mathsf{DisjVerify}^{\mathsf{upk}})$ from Appendix E to prove in zero-knowledge that (R, S) encrypts g^0 or g^1.

5 Helios-C : Fully Distributed Helios with Credentials

In this section we propose a modification of Helios [4] which is fully distributed, allows to choose an arbitrary threshold value and adds credentials to prevent ballot stuffing. We name it Helios-C, as a shortening for Helios with Credentials. It takes into account the specification of Helios variants such as [14, 9]. We emphasize that we present a completely detailed fully distributed version of Helios with arbitrary threshold parameters, for the first time with formal proofs. We use the fully distributed IND-CPA cryptosystem $\mathcal{D} = (\mathsf{DistKG}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{Rec})$ from Section 4; an existentially unforgeable signature scheme $\mathcal{S} = (\mathsf{SKeyGen}, \mathsf{Sign}, \mathsf{SVerify})$; the NIZK proof system $\mathsf{DisjProof}^{\mathsf{upk}}(g, \mathsf{pk}, R, S)$ from Appendix E to prove in zero-knowledge that (R, S) encrypts g^0 or g^1; and the NIZK proof system $\mathsf{EqDl}(g, R, \mathsf{vk}, c)$ to prove in zero-knowledge that $\log_g \mathsf{vk} = \log_R c$ for $g, R, \mathsf{vk}, c \in \mathbb{G}$. 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, Helios-C consists of eight algorithms $\mathcal{V}^{\mathsf{heliosc}} = (\mathsf{Register}, \mathsf{Setup}, \mathsf{Vote}, \mathsf{Validate}, \mathsf{VerifyVote}, \mathsf{BallotBox}, \mathsf{Tally}, \mathsf{Verify})$ defined below:

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

$\mathsf{Setup}(1^\lambda, t, \ell)$ runs $\mathsf{DistKG}(1^\lambda, t, \ell)$ from Section 4, 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.

$\mathsf{Vote}\,(id, \mathsf{upk}, \mathsf{usk}, v)$ it is used by a voter with credentials $(\mathsf{upk}, \mathsf{usk})$ to create a ballot b as follows:

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

(2) It computes a signature on the ciphertext and proof as $\sigma \leftarrow \mathsf{Sign}(\mathsf{usk}, (C, \pi))$.

The ballot is defined as $b = (\mathsf{upk}, (C, \pi), \sigma)$.

$\mathsf{Validate}(b)$ parses the ballot b as a tuple $(\mathsf{upk},\ (C, \pi), \sigma)$. It then checks whether: (1) $\mathsf{upk} \in L$; (2) $\mathsf{DisjVerify}^{\mathsf{upk}}(g, \mathsf{pk}, C, \pi) = 1$; (4) $\mathsf{SVerify}(\mathsf{upk}, \sigma, (C, \pi))$ accepts. 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 $b \in \mathsf{BB}$.

$\mathsf{BallotBox}(\mathsf{BB}, b)$ if $\mathsf{Validate}(b) = \texttt{reject}$, it halts. Else, (2) it parses $b = (\mathsf{upk}, (C, \pi), \sigma)$ and checks whether the credential upk 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 $T_j, 1 \leq j \leq \ell$:

(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 $(\mathsf{upk}_b, (C_b, \pi_b), \sigma_b)$.

(3) Checks whether upk_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 $\mathsf{ProveEq}(g, \mathsf{vk}_j, R_\Sigma, c_j)$.

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. 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 $1 \leq j \leq \ell$. 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 6. *Helios-C is correct, verifiable and ballot private against a malicious bulletin board.*

The proofs are to be found in Appendices I, J and K.

6 Implementation of Helios-C

We have implemented a proof of concept of our variant of Helios, called Helios-C, openly accessible at [2].

In case credentials are generated by a third-party provider and sent to the voters by snail mail, it would be cumbersome for voters to copy their signature key by typing it. We use a trick which consists in sending only the random seed used for generating the key, which can be encoded in about 12-15 alphanumeric characters depending on the desired entropy. It is expected that this seed is used by the provider to add the generated public key to L, then sent (as a password) to its rightful recipient and immediately destroyed.

Our variant of Helios requires voters to additionally sign their ballots. Table 4 shows the overhead induced by the signature, for various numbers of candidates (from 2 to 50). The two first lines are timings on the client side: the first one indicates the time needed by the voter's browser to form the ballot (without signature) while the second line indicates the computation time for signing. The third and fourth lines indicate the computation time on the server side for performing the verification tests (well-formedness of the ballot, validity of the proofs of knowledge and validity of the signature). In practice, we use the Schnorr signature scheme, which is very similar to the proofs of knowledge already in place for Helios, and a 256-bit multiplicative subgroup of a 2048-bit prime field for ElGamal and Schnorr operations. The figures have been obtained on a computer with an Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz, running Firefox 18. Unsurprisingly, the overhead of the signature is small compared to the computation time of the whole ballot.

For the fully distributed threshold cryptosystem to be more practical for the parties involved, we use a similar 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.

number of candidates	2	5	10	20	30	40	50
encryption+proofs time (ms)	600	1197	2138	4059	6061	7789	9617
signature time (ms)	196	215	248	301	358	423	484
signature verification time	$<$ 10 ms	$<$ 10 ms	$<$ 10 ms	$<$ 10 ms	$<$ 10 ms	$<$ 10 ms	$<$ 10 ms
ballot verification time (ms)	110	210	390	720	1070	1380	1730

Fig. 4. Overhead induced by adding signatures

Acknowledgements. We are thankful to Bogdan Warsinchi for his insights on ballot privacy and to Mark Ryan for his comments.

References

1. International association for cryptologic research. Elections page at http://www. iacr.org/elections/.
2. Anonymized for blind submission.
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. Josh Benaloh. Verifiable secret-ballot elections. PhD thesis, Yale University, 1987.
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. 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.
12. 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.
13. 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.
14. Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. In *CSF*, pages 297–311. IEEE Computer Society, 2011.
15. 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.
16. 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.
17. Adam M. Davis, Dmitri Chmelev, and Michael R. Clarkson. Civitas: Implementation of a threshold cryptosystem. Computing and information science technical report, Cornell University, 2008.
18. 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.
19. Pierre-Alain Fouque and David Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In Boyd [12], pages 351–368.
20. 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.
21. 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.
22. Rosario Gennaro. Achieving independence efficiently and securely. In James H. Anderson, editor, *PODC*, pages 130–136. ACM, 1995.
23. 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`.

24. 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.
25. 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.
26. 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.
27. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
28. 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.
29. 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.
30. Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections*, pages 37–63, 2010.
31. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 526–535. ACM, 2010.
32. 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.
33. 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.
34. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash attacks on the verifiability of e-voting systems. In *IEEE Symposium on Security and Privacy*, pages 395–409. IEEE Computer Society, 2012.
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 [12], pages 331–350.
37. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.
38. Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptology*, 15(2):75–96, 2002.
39. 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.
40. 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.

A Proofs from Section 3

A.1 Generic construction satisfies consistency

(1) $\mathsf{BallotBox}(\mathsf{BB}, b_j) = \mathsf{BB} \cup \{b_j\}$ if id_j did not previously cast any ballot, where we let $b_j \leftarrow \mathsf{Vote}(id_j, \mathsf{Register}(1^\lambda, id_j), v_j)$, $v_j \in \mathbb{V}$. This is in fact the case, since $\mathsf{Validate}(b_j) = \mathtt{accept}$ due to the correctness of $\mathcal{S}$, the accuracy of $\mathcal{V}$, and the fact that upk_j does not previously appear in BB.

(2) $\mathsf{Validate}\big(\mathsf{Vote}(id_j, \mathsf{Register}(1^\lambda, id), v_j)\big) = \mathtt{accept}$ for $i \in \{1, \ldots, \tau\}, v_j \in \mathbb{V}$ was argued in the previous point.

(3) $\mathsf{Tally}(\{b_1, \ldots, b_\tau\}, \mathbf{sk})$ outputs $(\rho(v_1, \ldots, v_\tau), \Pi)$; this holds due to the accuracy of $\mathcal{V}$;

(4) $\mathsf{Verify}(\{b_1, \ldots, b_\tau\}, \rho(v_1, \ldots, v_\tau), \Pi) = \mathtt{accept}$, is implied the fact that BB is well-formed and by the equality $\mathsf{Verify}(\{\alpha_1, \ldots, \alpha_\tau\}, \rho(v_1, \ldots, v_\tau), \Pi) = \mathtt{accept}$.

A.2 Proof of Theorem 1

We start by noting that, since the bulletin board is honest, we know that $\mathsf{Validate}(b) = \mathtt{accept}\ \forall b \in \mathsf{BB}$ by definition. The fact that $\mathcal{V}$ admits partial tallying and is accurate (Condition 1 in Definition 2.4), implies that $\mathsf{result} \neq \mathtt{invalid}$. Additionally, $\mathsf{Event} := \mathsf{HVote}$, so it contains all the pairs (id_i^E, v_i^E) that were queried to the oracle $\mathcal{O}\mathsf{vote}()$.

We proceed to show that each one of the conditions (5), (7-8) from Figure 1 is satisfied with overwhelming probability (Condition (6) being void since $\mathsf{Event} := \mathsf{HVote}$). This would imply that $\mathcal{V}$ is correct for honest bulletin boards.

We start by discarding $\mathsf{Verify}(\mathsf{BB}, \mathsf{Tally}(\mathsf{BB}, \mathbf{sk})) \neq \mathtt{accept}$. This is indeed infeasible, $\mathcal{V}$ is accurate (Condition 2 in Definition 2.4). Thus Condition (5) is satisfied with overwhelming probability.

Now we show that the requirement $\mathsf{result} = \rho\left(\{v_i^E\}_{i=1}^{n_E}\right) \star_{\mathbb{R}} \rho\left(\{v_i^B\}_{i=1}^{n_B}\right)$ where $v_1^B, \ldots, v_{n_B}^B \in \mathbb{V}$ and $0 \leq n_B \leq |\mathcal{CU}|$, must hold with overwhelming probability. Before proceeding, let us easily see that $n_B \leq |\mathcal{CU}|$. Indeed, if BB is honest, and given that each voter can cast only one ballot to BB, the adversary needs to corrupt an extra voter for each extra ballot it wants to cast without calling $\mathcal{O}\mathsf{vote}()$.

(i) Let $b_1^B, \ldots, b_{n_B}^B$ be the ballots appearing in BB which have been directly cast by the adversary. Since $\mathcal{V}$ admits partial tallying, we have that $\mathsf{Tally}(\{b_i^B\}_{i=1}^{n_B}, \mathbf{sk}) = \mathsf{result}_1^B \star_{\mathbb{R}} \ldots \star_{\mathbb{R}} \mathsf{result}_{n_B}^B$ where $(\mathsf{result}_i^B, \mathit{\Pi}_B^i) \leftarrow \mathsf{Tally}(\{b_i^B\}, \mathbf{sk})$, whenever $\mathsf{result}_i^B \neq \mathtt{invalid}$ for $i = 1, \ldots, n_B$. But since $\mathcal{V}$ is accurate, Condition 2 in Definition 2.4 establishes that $\mathsf{result}_i = \rho(v_i^B)$ for a certain (maybe unknown) $v_i^B \in \mathbb{V}$ with overwhelming probability.

(ii) Let us show that every vote in Event is numbered in result. Let us assume that, on the contrary, there exists at least one vote v_j^E which does not contribute to result. Since BB is honest, we can write $\mathsf{BB} = \{b_i^E\}_{i=1}^{n_E} \cup \{b_i^B\}_{i=1}^{n_B}$. Since $\mathcal{V}$ is consistent and admits partial tallying, we know that $\mathsf{Tally}(\mathsf{BB}, \mathbf{sk}) = \mathsf{result}_E \star_{\mathbb{R}} \mathsf{result}_B$, where $(\mathsf{result}_E, \mathit{\Pi}_E) \leftarrow \mathsf{Tally}(\{b_i^E\}_{i=1}^{n_E}, \mathbf{sk})$ and $(\mathsf{result}_B, \mathit{\Pi}_B) \leftarrow \mathsf{Tally}(\{b_i^B\}_{i=1}^{n_B}, \mathbf{sk})$ (we know from **(i)** that $\mathsf{result}_B \neq \mathtt{invalid}$). Furthermore, the partial tallying property and the consistency of $\mathcal{V}$ imply that $\mathsf{Tally}(\{b_i^E\}_{i=1}^{n_E}, \mathbf{sk}) = \rho(v_1^E) \star_{\mathbb{R}} \ldots \rho(v_{n_E}^E)$. Thus, if there exists $(id_j^E, v_j^E) \in \mathsf{Event}$ that does not contribute to result, the adversary must have deleted the corresponding ballot from BB. But by definition the adversary can not delete ballots from BB. We conclude then that every vote v_j^E belonging to Event is counted in the tally.

Putting together **(i)** and **(ii)** implies that Conditions (7-8) are satisfied. This ends the proof. □

A.3 Proof of Theorem 2

We proceed to show that each one of the conditions (5-8) from Figure 1 is satisfied with overwhelming probability, and thus $\mathcal{V}^{\mathsf{c}}$ has correctness against a malicious bulletin board.

We start by discarding $\mathsf{Verify}(\mathsf{BB}, \mathsf{Tally}(\mathsf{BB}, \mathbf{sk})) \neq \mathtt{accept}$. First, we notice that since $\mathsf{result} \neq \mathtt{invalid}$, it follows that BB is well-formed (as defined in the transformation in Section 3). Therefore $\mathsf{Verify}(\mathsf{BB}, \mathsf{Tally}(\mathsf{BB}, \mathbf{sk})) := \mathsf{Verify}(\mathsf{BB}', \mathsf{Tally}(\mathsf{BB}', \mathbf{sk}))$, where BB' is obtained from BB as specified in the transformation. In particular, since BB is well-formed, we have $\mathsf{Validate}(\alpha) = \mathtt{accept}$ for every $\alpha \in \mathsf{BB}'$. Finally, $\mathsf{Verify}(\mathsf{BB}', \mathsf{Tally}(\mathsf{BB}', \mathbf{sk})) = \mathtt{accept}$ using that $\mathcal{V}$ is correct against an honest board. Thus Condition (5) is satisfied with overwhelming probability.

Now we show that the requirements $\mathsf{result} = \rho\left(\{v_i^E\}_{i=1}^{n_E}\right) \star_{\mathbb{R}} \rho\left(\{v_i^A\}_{i=1}^{n_A}\right) \star_{\mathbb{R}} \rho\left(\{v_i^B\}_{i=1}^{n_B}\right)$, such that $\{(id_1^E, v_1^E), \ldots, (id_{n_E}^E, v_{n_E}^E)\} = \mathsf{Event}$; $(id_1^A, v_1^A), \ldots, (id_{n_A}^A, v_{n_A}^A) \in \mathsf{HVote}$; $v_1^B, \ldots, v_{n_B}^B \in \mathbb{V}$ and $0 \leq n_B \leq |\mathcal{CU}|$, must hold with overwhelming probability. Before proceeding, let us write $\mathsf{BB} = \{b_i^E\}_{i=1}^{n_E} \cup \{b_i^A\}_{i=1}^{n_A} \cup \{b_i^B\}_{i=1}^{n_B}$ and $\mathsf{BB}' = \{\alpha_i^E\}_{i=1}^{n_E} \cup \{\alpha_i^A\}_{i=1}^{n_A} \cup \{\alpha_i^B\}_{i=1}^{n_B}$, where the α's are obtained from the b's as specified in the transformation in Section 3. Since BB is well-formed, we know that $\mathsf{Tally}(\mathsf{BB}, \mathbf{sk}) = \mathsf{Tally}'(\mathsf{BB}', \mathbf{sk})$. Furthermore, since $\mathcal{V}$ admits partial tallying, we have $\mathsf{Tally}'(\mathsf{BB}', \mathbf{sk}) = \mathsf{result}_E \star_{\mathbb{R}} \mathsf{result}_A \star_{\mathbb{R}} \mathsf{result}_B$, where $(\mathsf{result}_X, \mathit{\Pi}_X) \leftarrow \mathsf{Tally}'(\{\alpha_i^X\}_{i=1}^{n_X}, \mathbf{sk})$ as long as $\mathsf{result}_X \neq \mathtt{invalid}$ for $X = E, A, B$, which is the case. Furthermore, $\mathsf{Tally}'(\{\alpha_i^X\}_{i=1}^{n_X}, \mathbf{sk}) = \rho(v_1^X) \star_{\mathbb{R}} \ldots \rho(v_{n_X}^X)$ for $X = E, A, B$.

(i) The fact that every vote in Event is numbered in result is guaranteed by the correctness against an honest board of $\mathcal{V}$.

(ii) Let us now assume that the adversary has added a ballot $b = (\mathsf{upk}_{id}, \alpha, \sigma)$ to BB, such that $\mathsf{Validate}(b) = \mathtt{accept}$, but b was never submitted by an honest voter id with $(id, *) \in \mathsf{HVote}$. This could be potentially done by the adversary without being caught, since voters appearing in HVote do not check whether their ballot appears in BB after the voting phase is over. But, since b passes the validity test, the signature σ on α is valid, while σ was never produced by the challenger. Thus, $\mathcal{A}$ would succeed in this case in forging a signature for public signing key upk_{id}, which is only possible with negligible probability since the signature scheme $\mathcal{S}$ is existentially unforgeable.

(iii) let us assume this time that $n_B > |\mathcal{CU}|$, where n_B is the number of ballots directly cast by the adversary. This means that:

(iii-1) there is more than one ballot associated to a corrupted user which contributes to the election tally result. In this case, the bulletin board contains two entries $b_i = (\mathsf{upk}_{id}, \alpha_i, \sigma_i)$ and $b_j = (\mathsf{upk}_{id}, \alpha_j, \sigma_j)$. But this is discarded, since BB is well-formed. Indeed, if $\mathsf{Tally}(\mathsf{BB}, \mathbf{sk}) \neq \mathtt{invalid}$, then BB is well-formed and it cannot contain two entries with duplicated credentials.

(iii-2) the adversary has added a valid ballot $b = (\mathsf{upk}_{id}, *, *)$ without invoking $\mathcal{O}\mathsf{corrupt}$ for voter id. In this case, the adversary has produced a ballot $b = (\mathsf{upk}_{id}, \alpha, \sigma)$ for a registered user id such that σ is a valid signature on α without knowledge of the corresponding usk. This means that the adversary succeeds in forging a valid signature for a message α, which violates the existential unforgeability of $\mathcal{S}$.

(iv) Let $b_1^B, \ldots, b_{n_B}^B$ be the ballots appearing in BB which have been directly cast by the adversary. We know that $\mathsf{result}_B = \mathsf{result}_1^B \star_{\mathbb{R}} \ \ldots \ \star_{\mathbb{R}} \mathsf{result}_{n_B}^B$ where $(\mathsf{result}_i^B, \Pi_B^i) \leftarrow \mathsf{Tally}'(\{\alpha_i^B\}, \mathbf{sk})$. The correctness of $\mathcal{V}$ against an honest bulletin board implies that $\mathsf{result}_i = \rho(v_i^B)$ for a certain (maybe unknown) $v_i^B \in \mathbb{V}$ with overwhelming probability.

Putting together **(i-iv)** implies that Conditions (6-8) are satisfied. This ends the proof. □

A.4 Proof of Theorem 3

Since the triples $(\mathsf{BB}, \mathsf{result}, \Pi)$ and $(\mathsf{BB}, \mathsf{result}', \Pi')$ with $\mathsf{result} \neq \mathsf{result}'$ output by $\mathcal{A}$ passes verification for $\mathcal{V}^{\mathsf{c}}$, it must be the case that the triples $(\mathsf{BB}', \mathsf{result}, \Pi)$ and $(\mathsf{BB}', \mathsf{result}', \Pi')$, where BB' is obtained from BB as specified in the transformation, also pass verification for $\mathcal{V}$. But this can only happen with negligible probability, since $\mathcal{V}$ is verifiable. □

A.5 Proof of Theorem 4

Assume that there exists an adversary $\mathcal{A}^{\mathsf{cred}}$ against the ballot privacy of $\mathcal{V}^{\mathsf{c}} = (\mathsf{Register}, \mathsf{Setup}, \mathsf{Vote}, \mathsf{VerifyVote}, \mathsf{Validate}, \mathsf{BallotBox}, \mathsf{Tally}, \mathsf{Verify})$. We will build from it an adversary $\mathcal{A}$ against the ballot privacy of the underlying voting scheme $\mathcal{V} = (\mathsf{Setup}', \mathsf{Vote}', \mathsf{VerifyVote}', \mathsf{Validate}', \mathsf{BallotBox}', \mathsf{Tally}', \mathsf{Verify}')$.

To do this we will make $\mathcal{A}$ simulate the ballot privacy game to $\mathcal{A}^{\mathsf{cred}}$ using the ballot privacy challenger for $\mathcal{V}$. Thus, when $\mathcal{A}^{\mathsf{cred}}$ makes a query $\mathcal{O}\mathsf{register}(id)$, the adversary $\mathcal{A}$ runs $(\mathsf{upk}', \mathsf{usk}') \leftarrow \mathsf{Register}'(id)$ and $(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{SKeyGen}(1^\lambda)$, and maintain the lists $L, \mathcal{U}, \mathcal{CU}$ consistently to what $\mathcal{A}^{\mathsf{cred}}$ expects to see. We denote $\mathbf{upk} \leftarrow (\mathsf{upk}', \mathsf{upk})$ and $\mathbf{usk} \leftarrow (\mathsf{usk}', \mathsf{usk})$. Once $\mathcal{A}^{\mathsf{cred}}$ asks to see the public key of the election, $\mathcal{A}$ uses the public key $\mathbf{pk}'$ he receives from $\mathcal{V}$'s challenger to build $\mathbf{pk} \leftarrow (\mathbf{pk}', L)$ and sends $\mathbf{pk}$ to $\mathcal{A}^{\mathsf{cred}}$. Eventually parts of $\mathbf{sk}$ are revealed both to $\mathcal{A}$ and $\mathcal{A}^{\mathsf{cred}}$ if the latter chose to corrupt a subset of trustees. The first phase of the ballot privacy game is over.

Next, $\mathcal{A}$ will, in interaction with $\mathcal{A}^{\mathsf{cred}}$, build its two local boards $\mathsf{BB}'_0, \mathsf{BB}'_1$. At the same time, $\mathcal{A}$ will build $\mathcal{A}^{\mathsf{cred}}$'s local boards $\mathsf{BB}_0, \mathsf{BB}_1$ using the contents of $\mathsf{BB}'_0, \mathsf{BB}'_1$. Adversary $\mathcal{A}^{\mathsf{cred}}$ can ask $\mathcal{O}\mathsf{corruptU}$ queries on input id. Then $\mathcal{A}$ relays $(\mathbf{upk}, \mathbf{usk})$ to $\mathcal{A}^{\mathsf{cred}}$, whenever they are defined. When $\mathcal{A}^{\mathsf{cred}}$ invokes the OvoteLR oracle on input (id, v_0, v_1), adversary $\mathcal{A}$ first checks whether user id is registered. If not, $\mathcal{A}$ replies nothing; otherwise, $\mathcal{A}$ starts by computing $\alpha_0 \leftarrow \mathsf{Vote}'(id, \mathsf{upk}', \mathsf{usk}', v_0)$ and $\alpha_1 \leftarrow \mathsf{Vote}'(id, \mathsf{upk}', \mathsf{usk}', v_1)$. Next it sets $b_0 \leftarrow (\mathsf{upk}, \alpha_0, \sigma_0)$ and $b_1 \leftarrow (\mathsf{upk}, \alpha_1, \sigma_1)$, where $\sigma_0 = \mathsf{Sign}(\mathsf{usk}, \alpha_0)$ and $\sigma_1 = \mathsf{Sign}(\mathsf{usk}, \alpha_1)$. Finally $\mathcal{A}$ runs $\mathsf{BallotBox}(\mathsf{BB}_0, b_0)$ and $\mathsf{BallotBox}(\mathsf{BB}_1, b_1)$.

At some point $\mathcal{A}^{\mathsf{cred}}$ outputs a modified bulletin board BB_β (for $\beta \in \{0, 1\}$ unknown both to $\mathcal{A}, \mathcal{A}^{\mathsf{cred}}$) which consists, on the one hand, on ballots $b = (\mathsf{upk}, \alpha_\beta, \sigma_\beta)$ as created by $\mathcal{A}$ using the specification of the algorithm $\mathsf{Vote}(\cdot)$, and on the other hand, on ballots $b = (\mathsf{upk}, \alpha, \sigma)$ created arbitrarily by $\mathcal{A}^{\mathsf{cred}}$ itself.

Internally, $\mathcal{A}$ needs to define a simulator $\mathsf{SimTally}$ such that $\mathcal{A}^{\mathsf{cred}}$ can not distinguish between the output of $\mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$ and the output of $\mathsf{SimTally}(\mathsf{BB}_0, \mathsf{BB}_1, \mathbf{pk}, \mathtt{info})$. Note that while one of the boards, actually BB_β, is known to $\mathcal{A}$, there exists a second board, actually $\mathsf{BB}_{1-\beta}$, which is unknown to $\mathcal{A}$. To build the output of $\mathsf{SimTally}$ the adversary $\mathcal{A}$ thus makes use of $\mathsf{SimTally}'$, which is guaranteed to exist since $\mathcal{V}$ is private. $\mathcal{A}$ defines the output of $\mathsf{SimTally}(\mathsf{BB}_0, \mathsf{BB}_1, \mathbf{pk}, \mathtt{info})$ to be the output of $\mathsf{SimTally}'(\mathsf{BB}'_0, \mathsf{BB}'_1, \mathbf{pk}', \mathtt{info})$, where $\mathsf{BB}'_0, \mathsf{BB}'_1$ are obtained from $\mathsf{BB}_0, \mathsf{BB}_1$ by simply keeping the α-component of each of their ballots.

Next, when $\mathcal{A}^{\mathsf{cred}}$ asks $\mathcal{A}$ to reveal either $\mathsf{Tally}(\mathsf{BB}, \mathbf{sk})$ or $\mathsf{SimTally}(\mathsf{BB}_0, \mathsf{BB}_1, \mathbf{pk}, \mathtt{info})$, the adversary $\mathcal{A}$ does the following. It checks before deciding whether to give a meaningful response to $\mathcal{A}^{\mathsf{cred}}$ that BB is well-formed. Namely if: every upk appearing in BB appears only once; every upk showing up in BB belongs to L; $\mathsf{Validate}(b) = \mathtt{accept}$ for every $b \in \mathsf{BB}$. If any of these checks fails, $\mathcal{A}$ relays $\mathtt{invalid}$ to $\mathcal{A}^{\mathsf{cred}}$ and outputs $\hat{\beta} \overset{R}{\leftarrow} \{0,1\}$ as it guess to $\mathcal{V}$'s challenger. Else, $\mathcal{A}$ relies to $\mathcal{A}^{\mathsf{cred}}$ whatever $\mathcal{V}$'s challenger output at the end of procedure $\mathsf{End}(\mathsf{BB}'_\beta, \mathbf{pk}', \mathbf{sk})$, where BB'_β is obtained from BB_β by simply keeping the α-component of each of the ballots in BB_β.

By construction, it holds that BB'_β is a legitimate board in the view of $\mathcal{V}$'s challenger, as it contains only valid ballots from different voters. Furthermore, the contents of BB_β are correlated with BB'_β. Let $\hat{\beta}$ be the guess of $\mathcal{A}^{\mathsf{cred}}$. Finally, $\mathcal{A}$ outputs $\hat{\beta}$ as its guess to $\mathcal{V}$'s ballot privacy challenger. It is easy to see that, the adversary $\mathcal{A}$ we have described is always a legitimate ballot privacy adversary against $\mathcal{V}$. Hence a ballot privacy $\mathcal{A}^{\mathsf{cred}}$ against $\mathcal{V}^{\mathsf{c}}$ implies a ballot privacy adversary $\mathcal{A}$ against $\mathcal{V}$. The proof of the theorem is concluded. □

B Proof of Theorem 5

The reduction we show next is based on the the ideas used by Gennaro *et. al* [26] 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 $h \overset{R}{\leftarrow} \mathbb{G}$. To this end, we will use the IND-CPA adversary against the cryptosystem from Section 4. 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 $\beta \overset{R}{\leftarrow} \{0,1\}$. 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 $s_{lj} \overset{R}{\leftarrow} \mathbb{Z}_q$ 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 $s_{lj} \overset{R}{\leftarrow} \mathbb{Z}_q$ for $j \in B$ as indicated in [26]. 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. [26] 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 [26] 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. □

C 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 $\overline{\mathsf{Verify}}_\Sigma(Y, com, ch, res) = 1$.

D Fiat-Shamir Transformation and NIZKs

Definition 1 (Fiat-Shamir Transformation [18,9]). *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 overwehlming 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. Here, we recall the dlog-based NIZK systems for proving equaltity of discrete-logarithm and disjunctive Chaum-Pedersen statements.

Equality of Discrete-Logarithms 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 $com_2 \leftarrow g_2^{res}/y_2^{ch}$.

We write $\mathsf{EqDl}(g_1, g_2, y_1, y_2) = (\mathsf{ProveEq}, \mathsf{VerifyEq})$ for $g_1, g_2, y_1, y_2 \in \mathbb{G}$ be 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, let the prover set $ch \leftarrow H(g_1, g_2, y_1, y_2, com_1, com_2)$, 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 $ch \overset{?}{=} H(g_1, g_2, y_1, y_2, com_1, com_2)$.

Disjunctive Chaum-Pedersen Let $\mathsf{DisjProof}(g, \mathsf{pk}, R, S) = (\mathsf{DisjProve}, \mathsf{DisjVerify})$ be 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 [15] 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 $\pi \leftarrow (ch_0, ch_1, res_0, res_1)$. On the other hand, $\mathsf{DisjVerify}(g, \mathsf{pk}, R, S, \pi)$ checks whether $ch_0 + ch_1 = H(g, \mathsf{pk}, R, S, \frac{G^{res_0}}{R^{ch_0}}, \frac{\mathsf{pk}^{res_0}}{S^{ch_0}}, \frac{G^{res_1}}{R^{ch_1}}, \frac{\mathsf{pk}^{res_1}}{(S \cdot g^{-1})^{ch_1}})$.

E 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 $\mathsf{Simulate}_\Sigma((id, Y), c, f) := \mathsf{Simulate}'_\Sigma(Y, c, f)$, $\overline{\mathsf{Verify}}_\Sigma((id, Y), A, c, f) = \overline{\mathsf{Verify}}'_\Sigma(Y, A, c, f)$ and let $\mathsf{Extract}_\Sigma$ be identical to $\mathsf{Extract}'_\Sigma$.

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 7. *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 7 implies that if we let $\mathsf{DisjProof}^{\mathsf{upk}}(g, \mathsf{pk}, R, S)$ be the NIZK proof system obtained from $\mathsf{DisjProof}(g, \mathsf{pk}, R, S)$ by adding upk 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 in Helios-C to avoid ciphertext weeding.

F NM-CPA Transformation from Fully Distributed (t, ℓ)-Threshold Cryptosystems

In this section, we apply the result of [10, 9] to the IND-CPA fully distributed (t, ℓ)-threshold cryptosystem construction of Section 4 with non-interactive version of the Chaum-Pedersen proof system to obtain a NM-CPA Fully Distributed (t, ℓ)-Threshold Cryptosystem.

Theorem 8. *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.*

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

$\mathsf{Enc}(\mathsf{pk}, m; r)$*: given* pk *and* m*, first 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)*. 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)*. If* $\overline{\mathsf{Verify}}((\mathsf{pk}, C'), com, ch, res) = 0$*, it returns* `reject`*. 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 ShareDec and Rec have to be adapted. We leave the details to the full version of this paper, as the proof from [10, 9] can be adapted with no additional difficulties.

G NM-CPA and Robust Fully Distributed (t, n)-Threshold Cryptosystem from DDH with Random Oracle

We know from previous work that ballot private Helios-like voting protocols are tightly related to NM-CPA cryptosystems [9, 10]. In this section, we show how to improve the IND-CPA ElGamal-based threshold cryptosystem from Section 4 into a robust and NM-CPA threshold cryptosystem that encrypts bit-by-bit. To do this, we use the NIZK proof system $\mathsf{DisjProof}^{\mathsf{upk}}(g, \mathsf{pk}, R, S) = (\mathsf{DisjProve}^{\mathsf{upk}}, \mathsf{DisjVerify}^{\mathsf{upk}})$ from Appendix E, where upk is any bit string, to prove in zero-knowledge that to prove in zero-knowledge that (R, S) encrypts g^0 or g^1.

Let $\mathcal{D}^\dagger$ be the cryptosystem from the previous section 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 $r \overset{R}{\leftarrow} \mathbb{Z}_q$ and set $(R, S) = (g^r, \mathsf{pk}^r \cdot g^m)$. Now it runs $\pi \leftarrow \mathsf{DisjProve}^{\mathsf{upk}}(g, \mathsf{pk}, R, S, r)$ and returns $C \leftarrow ((R, S), \pi)$.

$\mathsf{ShareDec}(\mathsf{sk}_j, \mathsf{vk}_j, C)$**:** parses C as $((R, S), \pi)$. If $\mathsf{DisjVerify}^{\mathsf{upk}}(g, \mathsf{pk}, R, S, \pi) = 0$, it returns `reject`. Else, it runs $(j, c_j) \leftarrow \mathsf{ShareDec}^\dagger(\mathsf{sk}_j, \mathsf{vk}_j, (R, S))$. It runs $\pi_j \leftarrow \mathsf{ProveEq}(g, R, \mathsf{vk}_j, c_j, \mathsf{sk}_j)$. It outputs (j, c_j, π_j).

$\mathsf{Rec}(\mathsf{pk}, \mathsf{vk}, C, \mathcal{C})$**:** parses C as $((R, S), \pi)$. If $\mathsf{DisjVerify}^{\mathsf{upk}}(g, \mathsf{pk}, R, S, \pi) = 0$, it returns `reject`. 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 $\mathsf{Rec}^\dagger(\mathsf{pk}, \mathsf{vk}, (R, S), \{c_{i1}, \ldots, c_{i(t+1)}\})$.

Theorem 9. $\mathcal{D}$ *is NM-CPA and robust in the Random Oracle Model under the DDH assumption.*

Proof. NM-CPA is obtained from Theorem 8 in Appendix F. This is because $\mathcal{D}^{\dagger}$ is IND-CPA (which has been shown in Theorem 5 and because the proof system $\mathsf{DisjProof}^{\mathsf{upk}}$ is the result of applying the augmented Fiat-Shamir transform from Appendix E to Chaum-Perdersen Σ-protocol (see Appendix D). This way the requirements of Theorem 8 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 $\mathsf{Rec}(\mathsf{pk}, (R, S), \mathcal{C}) \neq \texttt{reject} \neq \mathsf{Rec}(\mathsf{pk}, (R, S), \mathcal{C}^\star)$, we have that $\prod_{j=1,\ldots,t+1} c_{i_j}^{\lambda_j^{\mathcal{I}}} = \prod_{j=1,\ldots,t+1} (c^\star)_{k_j}^{\lambda_j^{\mathcal{K}}} = R^x$ and thus

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

with $\mathcal{I} = \{i_1, \ldots, i_{t+1}\}, \mathcal{K} = \{k_1, \ldots, k_{t+1}\}, \mathsf{pk} = g^x$. Equation (2) is equivalent to equation $\mathsf{Rec}(\mathsf{pk}, \mathbf{vk}, C, \mathcal{C}) = \mathsf{Rec}(\mathsf{pk}, \mathbf{vk}, C, \mathcal{C}^\star)$. □

H Avoiding weeding ballots belonging to different users

Recall that weeding ballots is essential for proving ballot privacy. We will show that there is no need in Helios-C of weeding ballots (as previously done in [14, 9]) to avoid duplication of atomic ballots. Let us recall that the latter property is essential for proving ballot privacy. In effect, let $\pi^{\mathsf{upk}} = (c_0, c_1, f_0, f_1)$ and $\pi^{\mathsf{upk}'} = (c'_0, c'_1, f'_0, f'_1)$ be disjunctive Chaum-Pedersen NIZKs asserting that two given ciphertexts belonging to different voters with public credentials $\mathsf{upk} \neq \mathsf{upk}'$ are encryptions of 0 or 1 in Helios-C. Ballot weeding consists on rejecting to add ballots $b_{\mathsf{upk}'}$ to the bulletin board such that $\pi^{\mathsf{upk}'} = \pi^{\mathsf{upk}}$ if the atomic proof π^{upk} is contained in a previous ballot b_{upk}. We aim at simplifying this procedure. First, notice that if the proofs $\pi^{\mathsf{upk}}, \pi^{\mathsf{upk}'}$ verify with respect to the corresponding ciphertexts $(R, S), (R', S')$, then it holds

$$c_0 + c_1 = H\left(\mathsf{upk}, R, S, \frac{g^{f_0}}{R^{c_0}}, \frac{Y^{f_0}}{S^{c_0}}, \frac{g^{f_1}}{R^{c_1}}, \frac{Y^{f_1}}{(Sg^{-1})^{c_1}}\right)$$

$$c'_0 + c'_1 = H\left(\mathsf{upk}', R', S', \frac{g^{f'_0}}{(R')^{c'_0}}, \frac{Y^{f'_0}}{(S')^{c'_0}}, \frac{g^{f'_1}}{(R')^{c'_1}}, \frac{Y^{f'_1}}{(S'g^{-1})^{c'_1}}\right)$$

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 $\pi^{\mathsf{upk}}, \pi^{\mathsf{upk}'}$ we have that $\Pr[\pi^{\mathsf{upk}} = \pi^{\mathsf{upk}'} \mid \mathsf{upk} \neq \mathsf{upk}'] = \Pr[H(\mathsf{upk}, \pi^{\mathsf{upk}}) = H(\mathsf{upk}', \pi^{\mathsf{upk}'}) \mid \mathsf{upk} \neq \mathsf{upk}' \text{ fixed in advance }]$ for any $\pi^{\mathsf{upk}}, \pi^{\mathsf{upk}'}$ satisfying Equation 3. In particular assume that H is a collision-resistant hash function. Then $\mathsf{Prob}[\pi^{\mathsf{upk}} = \pi^{\mathsf{upk}'} \mid \mathsf{upk} \neq \mathsf{upk}']$ is less than the probability of finding a collision on the hash function.

I Proof of Correctness against a Malicious Bulletin Board for Helios-C

We first see that Helios-C without credentials is correct against a honest bulletin board if at least $t > \lceil \ell/2 \rceil$ are honest and follow the $\mathsf{Setup}(1^\lambda, t, \ell)$ algorithm from $\mathcal{V}^{\mathsf{heliosc}}$. By Theorem 1, it suffices to show that $\mathcal{V}^{\mathsf{heliosc}}$ admits partial tallying, is consistent and accurate if at least $t > \lceil \ell/2 \rceil$ are honest.

Partial tallying is implied by the homomorphic properties of the underlying IND-CPA threshold cryptosytem and by the nature of the result function, which is in fact the summation of votes.

Consistency easily follows from the completeness and robustness properties of the fully-distributed NM-CPA cryptosystem used in Helios-C, to be found in Appendix G.

Condition 1 of accuracy asks that for any ballot b it should hold that $[\mathsf{Validate}(b) = \mathtt{accept} \Leftrightarrow \mathsf{Tally}(\{b\}, \mathbf{sk}) = \rho(v)$ for some $v \in \mathbb{V}]$ with overwhelming probability. This is due to the completeness and soundness properties of the NIZK system $\mathsf{DisjProof}^{\mathsf{upk}}(g, \mathsf{pk}, R, S)$ from Appendix E, and to the consistency of Helios-C without credentials.

Condition 2 of accuracy asks that for any bulletin board BB it should hold $\mathsf{Verify}(\mathsf{BB}, \mathsf{Tally}(\mathsf{BB}, \mathbf{sk})) = \mathtt{accept}$ with overwhelming probability. For Helios-C without credentials this is guaranteed by the the completeness and soundness properties of the NIZK system $\mathsf{EqDl}(g_1, g_2, y_1, y_2)$ from Appendix D, and to the consistency of Helios-C without credentials.

Finally, the previous discussion and Theorem 2 implies that Helios-C is correct against a malicious bulletin board. □

J Proof of Verifiability for Helios-C

Verifiability easily follows from the robustness property of the fully-distributed NM-CPA cryptosystem from Appendix G. □

K Proof of Theorem 6

Sketch of the proof. We content ourselves here to shortly indicate how the proof works. This is because we can hardly claim any novelty here, as the main ideas 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.

Our proof of ballot privacy for Helios-C is divided into two parts. First we prove that removing the credentials from Helios-C results in a fully distributed ballot private voting protocol for honest bulletin boards. Secondly, the previous result and Theorem 4 imply that Helios-C is ballot private for malicious bulletin boards.

Proving that Helios-C without credentials is ballot private against a honest bulletin board boils down to generalizing the ballot privacy proof of [9] to the fully distributed setting. To do this, the first step is to show that every atomic ciphertext in Helios-C is obtained from a fully distributed NM-CPA cryptosystem. This is in fact the case, since Helios-C is built upon the fully distributed NM-CPA DDH-based cryptosystem from Appendix G.

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 D. 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. □