Adapting Helios for provable ballot privacy

David Bernhard[1], Véronique Cortier[2], Olivier Pereira[3],
Ben Smyth[2], Bogdan Warinschi[1]

[1] University of Bristol, England
[2] LORIA - CNRS, France
[3] Université Catholique de Louvain, Belgium

Abstract. Recent results show that the current implementation of Helios, a practical e-voting protocol, does not ensure independence of the cast votes, and demonstrate the impact of this lack of independence on vote privacy. Some simple fixes seem to be available and security of the revised scheme has been studied with respect to symbolic models.
In this paper we study the security of Helios using computational models. Our first contribution is a model for the property known as ballot privacy that generalizes and extends several existing ones.
Using this model, we investigate an abstract voting scheme (of which the revised Helios is an instantiation) built from an arbitrary encryption scheme with certain functional properties. We prove, generically, that whenever this encryption scheme falls in the class of *voting-friendly* schemes that we define, the resulting voting scheme provably satisfies ballot privacy.
We explain how our general result yields cryptographic security guarantees for the revised version of Helios (albeit from non-standard assumptions).
Furthermore, we show (by giving two distinct constructions) that it is possible to construct voting-friendly encryption, and therefore voting schemes, using only standard cryptographic tools. We detail an instantiation based on ElGamal encryption and Fiat-Shamir non-interactive zero-knowledge proofs that closely resembles Helios and which provably satisfies ballot privacy.

1 Introduction

Electronic voting protocols have the potential to offer efficient and sound tallying with the added convenience of remote voting. It is therefore not surprising that their use has started to gain ground in practice: USA, Norway and Estonia are examples of countries where e-voting protocols have been, at the very least, trialled in elections on a national scale.

Due to the sensitive nature of elections, security of e-voting protocols is crucial and has been investigated extensively. Among the security properties that have been identified for e-voting, perhaps the most desirable one is that users' votes should remain confidential. Three levels of confidentiality have been identified. These are (in increasing strength) the following.

– Ballot privacy: A voter's vote is not revealed to anyone.
– Receipt–freeness: A voter cannot obtain information which can prove to a coercer how she voted.
– Coercion resistance: Even a voter who collaborates with a coercer cannot obtain information that proves how she voted.

Other important properties that are desirable include ballot independence [12] (the ballots cast do not depend on each other) and end-to-end verifiability [23, 28, 38] (it is possible to verify that the election process has been followed honestly).

This paper is motivated by recent developments regarding the security of the Helios voting scheme [45]. Starting from version 2.0 [35], Helios has been using a variant of a classical protocol by Cramer et al. [14] incorporating tweaks proposed by Benaloh [29], and has been used in real-world elections, for example by the International Association for Cryptographic Research (IACR) to elect its 2010 board [36], by Princeton University to elect the undergraduate student government [46] and to elect the president of the Université Catholique de Louvain [35]. Helios aims to achieve only ballot privacy and explicitly discards the stronger confidentiality notions (which it does not satisfy) in favor of efficiency. It turns out that the current implementation of Helios does *not* enforce ballot independence (contrary to the original protocol of Cramer et al. [14]) and, as a result, Cortier and Smyth [37, 42] have exhibited several attacks against the ballot privacy property of Helios. (The property is called "ballot secrecy" in Cortier and Smyth's papers.) The attacks range from simple ballot copying to subtle reuse of parts of existing ballots, however they can all be detected (and prevented) by public algorithms. A revised scheme has been proved secure in a symbolic model but its security in the stronger, computational sense has not been assessed.

Contributions. We start by providing a *computational* security model for ballot privacy (Section 2). In a sense, our model generalizes and strengthens the model of [24, 26] where an attacker tries to distinguish when two ballots are swapped. Here, we ask that the adversary cannot detect whether the ballots cast are ballots for votes that the adversary has chosen or not. In doing so, the adversary is allowed to control arbitrarily many players and see the result of the election. Our model uses cryptographic games and thus avoids imposing the more onerous constraints that other definitional styles (in particular simulability) require from protocols.

Next we turn our attention to the revised version of Helios. Our analysis follows a somewhat indirect route: instead of directly analysing the scheme as it has been implemented, we analyze an abstract version of Helios that follows the same basic architecture, but where the concrete primitives are replaced with more abstract versions. Of course, the version we analyze implements the suggested weeding of ballots. We present this abstract scheme as a generic construction of a voting scheme starting from encryption scheme with specific functional and security properties (Section 3).

Focusing on this more abstract version brings important benefits. Firstly, we pin-down more clearly the requirements that the underlying primitives should

satisfy. Specifically, we identify a class of *voting-friendly* encryption schemes which when plugged in our construction yield voting schemes with provable ballot privacy. Roughly speaking, such encryption schemes are IND-CCA2 secure and have what we call a homomorphic embedding (parts of the ciphertexts can be seen as ciphertexts of a homomorphic encryption scheme). Secondly, our analysis applies to all voting schemes obtained as instantiations of our generic construction. Although we analyze and propose constructions which for efficiency reasons resort to random oracles, our generic approach also invites other (non-random oracle based) instantiations.

Next, we show how to construct voting-friendly encryption schemes using standard cryptographic tools (Section 4). We discuss two distinct designs. The first construction starts from an arbitrary (IND-CPA) homomorphic encryption scheme and attaches to its ciphertexts a zero-knowledge proof of knowledge of the plaintext. We refer to this construction as the Enc+PoK construction. Despite its intuitive appeal, we currently do not know how to prove that the above design leads to an IND-CCA2 secure encryption scheme (a proprety demanded by voting-friendliness). We therefore cannot conclude the security of our generic scheme when implemented with an arbitrary Enc+PoK scheme. Nevertheless, an investigation into this construction is important since the instantiation where Enc is the ElGamal scheme and PoK is obtained using the Fiat-Shamir paradigm applied to a Schnorr-like protocol corresponds precisely to the encryption scheme currently used in Helios. The security of this specific construction has been analyzed in prior work. Tsiounis and Yung [17] and Schnorr and Jakobsson [19] demonstrate that the scheme is IND-CCA2 secure, but their proofs rely on highly non-standard assumptions. Nevertheless, in conjunction with the security of our main construction, one can conclude that the current implementation of Helios satisfies ballot privacy based on either the assumption in [17] or those of [19].

We then take a closer look at the Enc+PoK construction and revisit a technical reason that prevents an IND-CCA2 security proof, first studied by Shoup and Gennaro [16]. Very roughly, the problem is that the knowledge extractor associated to the proof of knowledge may fail if used multiple times since its associated security guarantees are only for constant (or logarithmically many) uses. With this in mind, we note that a security proof is possible if the proof of knowledge has a so called *straight line* extractor [22]. This type of extractor can be reused polynomially many times. In this case, the Enc+PoK construction leads to a voting-friendly encryption scheme, whenever Enc is an arbitrary IND-CPA homomorphic encryption scheme.

The second design uses the well-known Naor-Yung transformation [7]. We show that if the starting scheme is an arbitrary (IND-CPA) homomorphic encryption scheme then the result of applying the NY transform is a voting-friendly encryption scheme. Applied generically, the transform may lead to non-efficient schemes (one of its components is a simulation-sound zero-knowledge proof of membership [18]). We present a related construction (where the proof of membership is replaced by a proof of knowledge) which can be efficiently instantiated in the random oracle model. In the final section of the paper (Section 5) we pro-

pose adopting an instantiation of Helios where the encryption-friendly scheme is implemented as above. The computational overhead for this scheme is reasonable (and can be further improved through specific optimization) and the scheme comes with the formal guarantees offered by the results of this paper.

Related work. Chevallier-Mames *et al.* [27] present an unconditional definition of ballot privacy but Helios cannot be expected to satisfy this definition due to its reliance on computational assumptions. Chevallier-Mames additionally show that their definition of unconditional ballot privacy is incompatible with universal verifiability; however, ballot privacy and universal verifiability have been shown to coexist under weaker assumptions, for example as witnessed by Juels, Catalano & Jakobsson [23]. Computational definitions of ballot privacy have been considered by Benaloh *et al.* [2, 4, 5]. These definitions however do not come with a general characterization of the properties that an encryption scheme should satisfy in order to ensure that they are satisfied (the corresponding security notions did not exist at that time either). Wikström [34] considered the general problem of secure submission of inputs with applications to mixnet-based voting protocols. His definitions and constructions are the most closely related to ours, and will be discussed below. Other definitions for voting systems have been proposed in terms of UC realization of ideal voting functionalities, starting with Groth [21], which capture privacy as part of the functionality behavior.

In addition, receipt-freeness has been considered by Benaloh & Tuinstra [11] and Moran & Naor [25] and coercion resistance has been studied by Juels, Catalano & Jakobsson [23], Küsters, Truderung & Vogt [40] and Unruh & Müller-Quade [39]. These definitions can be used to show ballot privacy because it is believed to be a weaker condition [11, 26]; however, they are too strong for protocols which only provide ballot privacy and in particular, they cannot be used to analyse ballot privacy in Helios. Ballot privacy has also been formalized in the symbolic model (for example, [26, 33]) but the symbolic model suffers a serious weakness: In general, a correct security proof does not imply the security of the protocol. Cortier & Smyth [37, 42] present an attack against ballot privacy in Helios and propose a variant of Helios which aims to prevent the attack by weeding ballots. Their solution has been shown to satisfy ballot privacy in the symbolic model but Cortier & Smyth acknowledge that a thorough cryptographic analysis of the solution is necessary.

2 Ballot privacy

Notation Throughout this paper, we use the following notation. Assignment and input/output of algorithms are both denoted by a left-facing arrow $\leftarrow$. Picking a value x uniformly at random from a set S is denoted by $x \xleftarrow{R} S$. The expression $C \xleftarrow{+} c$ appends c to the list C, () on its own is an empty list. We use "C" style returns in algorithms, i.e. "Return $a = b$" to mean return 1 if $a = b$, otherwise 0. A function f is called *negligible* if for any polynomial P, there exists η_0 such that $\forall \eta \geq \eta_0$, $f(\eta) \leq \frac{1}{P(\eta)}$.

2.1 Voting Schemes

In this section we fix a general syntax for the class of voting schemes that we treat in this paper. In particular, our syntax encompasses several variations of the Helios protocol.

We consider schemes for votes in a non-empty set $\mathbb{V}$, and we assume $\perp$ to be a special symbol not in $\mathbb{V}$ that indicates that the voter has abstained. The result of an election is then an arbitrary function ρ that takes a list of votes as input and returns the election result. Elections are stateful, so the algorithms that we define next use such a state. Since often, and in particular in the case of Helios, this state is a bulletin board, in the definition below we write BB for this state (and even refer to it as a bulletin board).

Definition 1 (Voting scheme). *Algorithms* (Setup, Vote, ProcessBallot, Tally) *define a voting scheme as follows.*

Setup *The setup algorithm takes a security parameter* 1^λ *as input and returns secret information* x*, public information* y*, and initializes the state* BB*. We write* $(x, y, BB) \leftarrow \mathsf{Setup}(1^\lambda)$ *for this process. We assume the public information is available to all subsequent algorithms.*

Vote *The voting algorithm takes a vote* $v \in \mathbb{V}$ *as input and produces as output a ballot* b *(that encodes the vote). We write* $b \leftarrow \mathsf{Vote}(v)$ *for this process.*

ProcessBallot *The ballot processing algorithm takes a candidate ballot* b *and a bulletin board* BB*, checks the ballot for correctness (e.g. that it is well formed, it is not a duplicate, etc.) and returns a result (accept/reject) and the new state of the bulletin board. We write* $(a, BB) \leftarrow \mathsf{ProcessBallot}(BB, b)$ *for this process. Here* a *is either* accept *or* reject.

Tally *The tallying algorithm takes the secret information* x *and the bulletin board* BB *and produces the election result.*

For correctness of the scheme, we demand two conditions: 1) ballot tallying corresponds to evaluating the function ρ on the underlying votes; and 2) correctly constructed votes will be accepted by the ballot processing algorithm. Both conditions should hold with overwhelming probability and can be captured by the experiment described in Figure 1. In this experiment, an adversary repeatedly submits votes $v_1, v_2, \ldots \in \mathbb{V}$ and each vote is used to construct a ballot which is then processed. The game outputs 1 (the adversary wins) if the ProcessBallot algorithm rejects some ballot or the result of the election does not correspond to the votes cast. The voting scheme is correct if the algorithm outputs 1 with at most negligible probability.

2.2 Security Model

Informally, ballot privacy is satisfied if an adversary in control of arbitrarily many voters cannot learn anything about the votes of the remaining, honest voters beyond what can be inferred from the election result. The adversary can read the (public) bulletin board and the communication channels between the

$\mathbf{Exp}^{corr}_{\Pi}(A)$
$(x, y, BB) \leftarrow \mathsf{Setup}$
$V = ()$
repeat
$(a, v) \leftarrow A$
$b \leftarrow \mathsf{Vote}(v)$
$(r, BB) \leftarrow \mathsf{ProcessBallot}(BB, b)$
$V \xleftarrow{+} v$
until $a = \mathsf{stop}$ or $r = \mathsf{reject}$
if $r =$"reject" or $\mathsf{Tally}(x, BB) \neq \rho(V)$ then return 1 else return 0

Fig. 1. Experiment for defining the correctness of a voting scheme.

honest parties and the bulletin board (in other words, we assume them to be authentic but not secret). Ballot privacy requires that the adversary is unable to distinguish between real ballots and fake ballots, where ballots are replaced by ballots for some fixed vote ε chosen by the adversary.

Formally, we consider an adversary that can issue two types of queries, vote and ballot, to an oracle $\mathcal{O}$. The oracle maintains two bulletin boards initialized via the setup algorithm: BB is visible to the adversary and BB' always contains ballots for the real votes. A vote query causes a ballot for the given vote to be placed on the hidden BB'. In the real world, the same ballot is placed on BB; in the fake one a ballot for ε is placed on BB instead. A ballot query always causes the submitted ballot to be processed on both boards. This process is defined formally in Figure 2. The experiment on the right of Figure 2 is used to define ballot privacy. The selection of β corresponds to the real world ($\beta = 0$) or the fake world ($\beta = 1$). Throughout the experiment the adversary has access to BB, but tallying is done using BB'.

Definition 2 (Ballot Privacy). *We define the advantage of adversary A in defeating ballot privacy for voting scheme Π by:*

$$\mathbf{Adv}^{BS}_{\Pi}(A) = \Pr[\mathbf{Exp}^{BS}_{\Pi}(A) = 1] - \frac{1}{2}$$

and say that Π ensures ballot privacy if for any efficient adversary its advantage is negligible.

We make a few remarks regarding the security model that we propose. Firstly, we use cryptographic games rather than a simulation based definition. The former offer well-accepted levels of security, are more flexible, and allow for more efficient implementations. Second, we model directly the more relaxed notion of vote privacy and not stronger notions like receipt-freeness or coercion resistance [26]. While stronger notions are certainly desirable, they are more difficult to achieve leading to rather inefficient protocols. Indeed, Helios deliberately trades these stronger notions for efficiency. Finally, we emphasize that our computational definition does not mirror existing security definitions in more

$\mathsf{vote}(v)$
$b' \leftarrow \mathsf{Vote}(v)$
if $\beta = 0$ then $b \leftarrow b'$
else $b \leftarrow \mathsf{Vote}(\varepsilon)$
$(r, BB) \leftarrow \mathsf{ProcessBallot}(b, BB)$
$(r', BB') \leftarrow \mathsf{ProcessBallot}(b', BB')$
return (r, BB, b)

$\mathsf{ballot}(b)$
$(r, BB) \leftarrow \mathsf{ProcessBallot}(b, BB)$
if $r = \mathsf{accept}$ then
$(r', BB') \leftarrow \mathsf{ProcessBallot}(b, BB')$
return (r, BB)

$\mathbf{Exp}_{\Pi}^{BS}(A)$
$(x, y, BB) \leftarrow \mathsf{Setup}(1^{\lambda})$
$BB' \leftarrow BB$
$(\varepsilon, st) \leftarrow A(y)$
$\beta \leftarrow \{0, 1\}$
$st \leftarrow A^{\mathcal{O}}(st)$
$\mathsf{result} \leftarrow \mathsf{Tally}(x, BB')$
$\hat{\beta} \leftarrow A(st, \mathsf{result})$
return $\beta = \hat{\beta}$

Fig. 2. The algorithms on the left explain how the oracle processes adversary's queries. The experiment on the right is used to define ballot privacy.

abstract models, e.g. [24]. It turns out that the direct extension of that definition to computational models seems strictly weaker than the definition that we provide. We comment more on this point later in the paper.

3 A generic construction of voting schemes with ballot privacy

In this section we present a generic construction of a voting scheme starting from any encryption scheme with certain properties. We first fix this class of encryption schemes (which we call voting-friendly), then give our construction and prove its security.

3.1 Voting-Friendly Encryption

In a nutshell, a voting-friendly encryption scheme is a "(threshold) checkable provable IND-CCA2 secure public key encryption scheme with key derivation and a homomorphic embedding". These rather convoluted looking requirements are in fact not too onerous. We explain informally each of the requirements in turn and give formal definitions. For simplicity, the presentation in this section is for the non-threshold case, that is decryption is carried out using a single key by a single party, as opposed to implementing decryption via an interactive process where several parties share the keys.

Non-Interactive Zero Knowledge Proof Systems. Here we recall some basic notions regarding non-interactive zero-knowledge proof systems [6]. Given language L_R defined by NP relation R we write $(w, x) \in R$ if w is the witness that $x \in L_R$. A proof system for L_R is given by a pair of algorithms $(\mathsf{Prover}, \mathsf{Verifier})$ called

prover and verifier, respectively. We distinguish between proof systems in the common reference string model (in this situation, an additional algorithm Setup produces a common reference string accessible to both the prover and the verifier) and the random oracle model (where the setup is not required, but all algorithms in the system have access to a random oracle). In a standard execution of the proof system, the prover and the verifier both have an element $x \in L_R$ as input and in addition, the prover has as input a witness w that $x \in L_R$ (i.e. $R(w, x) = 1$). The prover sends a single message π to the verifier who outputs the decision to accept/reject. We call π a proof for the statement $x \in L_R$. Typical requirements for such proof systems are that they should be sound (if the input x is not in L_R then the verifier rejects π with overwhelming probability) and complete (if x is in the language then the verifier accepts π with probability 1). We write $\pi \leftarrow \mathsf{Prover}$ for the process of producing proof π when the statement x and the witness w are clear from the context. A non-interactive proof system is zero-knowledge if there exists a simulator Sim that is able to produce transcripts indistinguishable from those of a normal execution of the protocol. The simulator may use a trapdoor in the common reference string model, or can program the random oracle in the random oracle model. We occasionally write $(\mathsf{Prover}, \mathsf{Verifier}) : R$ to indicate that the proof system is for the language L_R.

We assume the reader is familiar with public key encryption and its associated security notions. We write $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for the key generation, encryption, and decryption algorithms of a public key encryption scheme.

Homomorphic encryption. We also briefly recall the notion of *homomorphic* encryption. An encryption scheme is homomorphic if the plaintext space is a group and there exists an algorithm Add that takes two ciphertexts for messages m_0 and m_1 and produces a ciphertext for $m_0 \circ m_1$ (where $\circ$ is the group operation on plaintexts).

Embeddable Encryption. A crucial property for the encryption schemes that are the focus of this section is that they have a *homomorphic embedding*. Informally, this property means that it is possible to identify part(s) of the ciphertexts as forming a ciphertext for some other encryption scheme, and this second encryption scheme is homomorphic. The ElGamal+PoK construction sketched in the previous section is an example of an encryption scheme with an homomorphic embedding. Indeed the e component of a ciphertext (e, π) is a ciphertext for an homomorphic encryption scheme (ElGamal). The next definition makes this discussion more precise.

Definition 3 (Homomorphic Embedding). *We say that the homomorphic encryption scheme* $\Pi = (\mathsf{EGen}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EAdd})$ *is embedded in encryption scheme* $\Pi' = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$*, or alternatively that encryption scheme* Π' *has* Π *as a* homomorphic embedding *if there are algorithms* $\mathsf{ExtractKey}, \mathsf{Extract}$ *such that for all* m, pk, sk, c

$$\mathsf{EGen}() = \mathsf{ExtractKey}(\mathsf{Gen}())$$

$$\mathsf{EEnc}(m, \mathsf{ExtractKey}(pk)) = \mathsf{Extract}(\mathsf{Enc}(m, pk))$$

$$\mathsf{Dec}(c, sk) = \mathsf{EDec}(\mathsf{Extract}(c), sk)$$

Essentially, the ExtractKey algorithm maps keys (or key pairs) for the "larger" scheme to keys for the embedded one, and the Extract algorithm extracts the ciphertext for the embedded scheme out of ciphertext for the larger one, while performing validity verifications at the same time.

The Extract algorithm must, by definition, produce a ciphertext that decrypts to the same value as the input that it is given; in particular it must produce a "ciphertext" that decrypts to $\perp$ if and only if its input does. However, the Extract algorithm does not take any secret keys as input. This implies that anyone can check whether a ciphertext is valid (in the sense that it decrypts to something other than $\perp$) without knowing the secret key. This property forms the basis for combining homomorphic and IND-CCA2 secure encryption in our construction.

We note that an IND-CCA2 secure cryptosystem with homomorphic embedding is actually very close to a submission secure augmented (SSA) cryptosystem as defined by Wikström [34]. Some important differences appear, though. The most important one is that SSA cryptosystems do not require public verifiability of the ciphertexts: it might be necessary to publish a private key augmentation to be able to perform ciphertext validity checks. While this feature enables efficient solutions that are secure in the standard model, it is however often not desirable in practice: it is quite useful to be able to dismiss invalid votes as soon as they are submitted (and to resolve potential conflicts at that time) rather than needing to wait for some partial key to be revealed. Besides, in order to mitigate this inconvenience, SSA cryptosystems allow multiple independent augmentations, which enables updating an augmentation and revealing the previous one in order to be able to check the validity of previously submitted ciphertexts. Our requirement of immediate public verifiability property makes this feature unnecessary for our purpose.

We also note that in concurrent work, Persiano [44] and Smart [41] define similar embedding concepts.

S2P Key Derivation. This property simply requires that if a key pair is produced by the key generation algorithm of an encryption scheme then it is possible to compute the public key from the secret key. This property will allow us to use proofs of knowledge of the secret key corresponding to the public key.

Definition 4 (S2P Key Derivation). *An encryption scheme has the* S2P key derivation *property if there is an algorithm* DeriveKey *such that* $(x, y) \leftarrow \mathsf{Gen}$ *implies* $y = \mathsf{DeriveKey}(x)$.

Provable Encryption. In our generic construction voters need to certify that various encryptions in the ballots that they produce satisfy some desirable properties (e.g. that a ciphertext encrypts 0 or 1, and not something else), and such certification can be done via zero-knowledge proofs of knowledge. Since all of the

statements that we are interested in are NP statements, the existence of appropriate proof systems follows from general results [9]. Here, we make more precise the statements for which we demand the existence of such proof systems and introduce some useful notation for the proof systems associated to the various languages that we define.

In particular, it should be possible to prove knowledge of the secret key corresponding to the public key, knowledge of the plaintext underlying a ciphertext, as well as proving that a certain plaintext has been obtained by decrypting with the key associated to the public key. To avoid complex nomenclature, we call a scheme for which this is possible a scheme with provable encryption.

Definition 5 (Provable Encryption). *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* provable *if it has the S2P key derivation property and the following non-interactive zero-knowledge proof systems exist:*

1. $(\mathsf{ProveGen}, \mathsf{VerifyGen})$: $R_1(x, y) := y \stackrel{?}{=} \mathsf{DeriveKey}(x)$
2. $(\mathsf{ProveEnc}, \mathsf{VerifyEnc})$: $R_2((m, r), c) := c \stackrel{?}{=} \mathsf{Enc}(m; r)$
3. $(\mathsf{ProveDec}, \mathsf{VerifyDec})$: $R_3(x, (c, y, d)) := y \stackrel{?}{=} \mathsf{DeriveKey}(x) \wedge d \stackrel{?}{=} \mathsf{Dec}(x, c)$

The above definition is for standard encryption schemes. For the case when the encryption scheme that we need is embedded, we demand in addition the existence of proof systems for the following two properties. The first requires that one can prove a statement that involves plaintexts underlying several ciphertexts, and secondly, one should be able to prove that the keys for the embedded schemes in use have been correctly obtained from the keys of the embedding one. This latter condition is a simple adaptation of provability as defined above.

Definition 6 (Provable Embedding). *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *for message space* M *with embedded scheme* $(\mathsf{EGen}, \mathsf{EEnc}, \mathsf{EDec})$ *has* embedded provability *for* $M' \subseteq M^N$ *(for some* $N \in \mathbb{N}$*) if the following zero-knowledge proof-systems exist:*

1. $(\mathsf{ProveGen}, \mathsf{VerifyGen})$: $R_4(x, y) := y \stackrel{?}{=} \mathsf{DeriveKey}(x)$
2. $(\mathsf{ProveEnc}, \mathsf{VerifyEnc})$: $R_5((m_1, m_2, \ldots, m_N, r_1, r_2, \ldots, r_N), (c_1, c_2, \ldots, c_N)) :=$

$$\bigwedge_{i=1}^{N} c_i \stackrel{?}{=} \mathsf{Enc}(m_i; r_i) \wedge (m_1, \ldots, m_N) \in M'$$

3. $(\mathsf{ProveEDec}, \mathsf{VerifyEDec})$: $R_6(x, (y, d, c)) :=$

$$y \stackrel{?}{=} \mathsf{DeriveKey}(x) \wedge (x', y') \leftarrow \mathsf{ExtractKey}(x, y) \wedge d \stackrel{?}{=} \mathsf{EDec}(x', c)$$

In the last relation, the second conjunct is not a boolean condition, but simply indicates that the keypair (x', y') is derived from (x, y) using the $\mathsf{ExtractKey}$ algorithm.

The following definition states all the properties that we require from an encryption scheme in order to be able to implement our generic voting scheme.

Definition 7 (Voting-Friendly Encryption). *A voting-friendly encryption scheme for vote space $\mathbb{V}$ is a public-key scheme for message space M with $\mathbb{V} \subseteq M^N$ such that it is IND-CCA2 secure and has S2P key derivation, an embedded homomorphic scheme and embedded provability for $\mathbb{V}$.*

Note that voting-friendly encryption requires security guarantees of both the encryption scheme and the contained proof systems.

3.2 Our Generic Construction

In this section we describe a voting scheme based on an arbitrary voting-friendly encryption scheme. The design idea is similar to that of Helios.

The scheme handles elections with multiple candidates. In an election with three candidates a vote is a triple (a, b, c) such that $a, b, c \in \{0, 1\}$ and $a+b+c = 1$. A ballot is then simply formed by individually encrypting each component of the list with an IND-CCA scheme that has an homomorphic embedding, and proving in zero-knowledge that the individual plaintexts in a ballot satisfy the desired relation. To prevent an adversary from casting a vote somehow related to that of an honest voter, we ensure that each ballot cast does not contain any ciphertexts that are duplicates of ones in the ballots already on the bulletin board. This condition is checked while processing ballots.

More formally, denote the set of ciphertexts contained in a ballot b by $\mathsf{Cipher}(b)$ and the set of all ciphertexts on the bulletin board BB by $\mathsf{Cipher}(BB)$, that is $\mathsf{Cipher}(BB) = \bigcup_{b' \in BB} \mathsf{Cipher}(b')$. When submitting a ballot b, we check that $\mathsf{Cipher}(b) \cap \mathsf{Cipher}(BB) = \emptyset$.

Definition 8 (Abstract Voting Scheme). *Let Π be a voting-friendly encryption scheme. The abstract voting scheme $V(\Pi)$ is the construction consisting of algorithms 1–4.*

In our construction, $\mathcal{V}$ is the set of voters, $\mathcal{Z}$ is a party representing "the public" (elements sent to $\mathcal{Z}$ are published) which also functions as a trusted party for generating the initial setup parameters and $\mathcal{T}$ is the trustee of the election (that receives the decryption keys).

If M is the message space of the voting-friendly encryption scheme we consider the space of votes to be $\mathbb{V} \subseteq M^N$ for some $N \in \mathbb{N}$.

We consider result functions of the form $\rho : \mathbb{V}^* \to M^*$ where $\mathbb{V}^* := \cup_{i \in \mathbb{N}_0} \mathbb{V}^i$ (this allows us to tally an arbitrary number of votes) and each component of the range of ρ can be described by a sum of the form $\rho_k = \sum_{i \in \mathbb{N}} a_{i,k} \cdot v_i$ for constants $a_{i,k} \in \mathbb{N}$. This covers the class of result functions that can be computed homomorphically, including normal and weighted sums of votes but also the special case of revealing all the votes and allows us to exploit the homomorphism in the tallying operation: The same operation can be performed on homomorphic ciphertexts using the EAdd algorithm, for which we write $\oplus$ i.e. $a \oplus b := \mathsf{EAdd}(a, b)$. Furthermore, we can define scalar multiplication $\otimes$ on the ciphertexts i.e. $2 \otimes a := \mathsf{EAdd}(a, a)$.

Algorithm 1 $\mathsf{Setup}(1^\lambda)$

$\mathcal{Z}$:

$params \leftarrow \mathsf{Setup}(1^\lambda)$. These parameters are implicitly available to all further algorithms.

$BB \leftarrow ()$

$\mathcal{T}$:

$(x, y) \leftarrow \mathsf{Gen}(1^\lambda)$

$\pi^{Gen} \leftarrow \mathsf{ProveGen}(x, y)$

$\mathcal{Z} \leftarrow (y, \pi^{Gen})$

$\mathcal{Z}$:

$\mathsf{VerifyGen}(y, \pi^{Gen}) \stackrel{?}{=} 1$ or abort with failure.

Algorithm 2 $\mathsf{Vote}((v_1, v_2, \ldots, v_N))$

$\forall j \in \{1, 2, \ldots, N\}$

$c_j \leftarrow \mathsf{Enc}(y, v_j)$

$\pi_j^b \leftarrow \mathsf{ProveEnc}(y, v_j, c_j)$

$b_j \leftarrow (c_j, \pi_j^b)$

output b

Algorithm 3 $\mathsf{ProcessBallot}(b, BB)$

if $\mathsf{VerifyEnc}(b) = 0$ **then return** ("reject", BB) **end if**

for all $c \in \mathsf{Cipher}(b)$ **do**

if $\mathsf{Extract}(c) = \perp$ **then return** ("reject", BB) **end if**

if $\mathsf{Cipher}(b) \cap \mathsf{Cipher}(BB) \neq \emptyset$ **then return** ("reject", BB) **end if**

end for

$BB \stackrel{+}{\leftarrow} b$

return ("accept", BB)

Algorithm 4 $\mathsf{Tally}(BB)$

for all $c_j \in BB$ $(j \in \mathcal{V})$ **do** $e'_j \leftarrow \mathsf{Extract}(c_j)$ **end for**

for all k **do**

$e''_k \leftarrow \bigoplus_{j \in \mathcal{V}} (a_{j,k} \otimes e'_j)$ {I.e. use EAdd to compute ciphertexts for the results.}

$r_k \leftarrow \mathsf{EDec}(x, e''_k)$

$\pi_k^{Dec} \leftarrow \mathsf{ProveEDec}(x, e''_k, r_k)$

end for

$\mathcal{Z} \leftarrow (r_k, \pi_k^{Dec})_k$

Algorithm 5 Verification

$\mathcal{Z}$ performs the following, aborting if any of the checks (denoted by $\stackrel{?}{=}$) fail. The ordering on $\mathcal{V}$ is a slight abuse of notation; it represents the order the ballots were received in. If successful, the result of the election is r.

$\mathsf{VerifyGen}(y, \pi^{Gen}) \stackrel{?}{=} 1$
for all $j \in \mathcal{V}$ **do**
  $(c_j, \pi_j^b) \leftarrow b_j$
  $\mathsf{VerifyEnc}(b_j) \stackrel{?}{=} 1$
  $(c_j \notin (c_{j'})_{j' \in \mathcal{V}, j' < j}) \stackrel{?}{=} 1$
  $e'_j \leftarrow \mathsf{Extract}(c_j)$
  $e'_j \stackrel{?}{\neq} \perp$
end for
$e' \leftarrow \mathsf{EAdd}(\rho, (e'_j)_{j \in \mathcal{V}})$
$\mathsf{VerifyEDec}(r, \pi^{Dec}, e') \stackrel{?}{=} 1$

We also provide a public verification algorithm as Algorithm 5 although we do not define this property formally.

We only prove ballot privacy of our construction formally; correctness follows from the correctness of the voting-friendly encryption scheme. The following theorem states that ballot privacy relies entirely on the security of the underlying voting-friendly scheme.

Theorem 1. *Let Π be a voting-friendly encryption scheme. Then $V(\Pi)$ has ballot privacy.*

To prove the theorem we proceed in two steps. First, we strip the voting scheme of the unnecessary details that concern verifiability, resulting in a scheme that we call "mini-voting". We prove that ballot privacy for this latter scheme only relies on the IND-CCA2 security of the encryption scheme employed (which highlights IND-CCA2 security as the crucial property needed from the underlying building block). We then explain how to adapt the proof to show the security of $V(\Pi)$.

The full proof can be found in the full version of this paper.

4 Constructions for voting–friendly schemes

In the previous section we gave a generic construction of a voting scheme with ballot privacy starting from an arbitrary voting-friendly encryption scheme. In this section we show that such schemes can be easily constructed using standard cryptographic tools in both the standard and the random oracle models. We discuss three different possibilities.

Encrypt + PoK. This construction does not lead immediately to a voting-friendly scheme but its security is highly relevant to that of Helios, and the design idea forms the basis of a construction that we discuss later.

Under this paradigm, one attempts to construct an IND-CCA2 scheme starting from an IND-CPA scheme and adding to the ciphertext a non-interactive proof of knowledge of the underlying plaintext. Intuitively, this ensures that an adversary cannot make use of a decryption oracle (since he must know the underlying plaintext of any ciphertext) hence the security of the scheme only relies on IND-CPA security. Unfortunately, this intuition fails to lend itself to a rigorous proof, and currently the question whether Enc+PoK yields an IND-CCA2 scheme is widely open. A detailed treatment of the problem first appeared in [16].

Yet, the question is important for the security of Helios: the current implementation is essentially an instantiation of our generic construction with an Enc+PoK encryption scheme. More precisely the encryption scheme Enc is ElGamal, and the proof of knowledge is obtained by applying the Fiat-Shamir transform to a Schnorr proof. Per the above discussion, no general results imply that the resulting ElGamal+PoK scheme is IND-CCA2 secure (a requirement for voting-friendliness) and our generic result does not apply. However, if one is prepared to accept less standard assumptions, two existing results come in handy. The security of the particular construction that employs ElGamal encryption and Fiat-Shamir zero-knowledge proofs of knowledge has been investigated by Tsiounis & Yung [17] and Schnorr & Jakobsson [19]. Both works support the conjecture that the construction is IND-CCA2 but neither result is fully satisfactory. Tsiounis & Yung make a knowledge assumption that essentially sidesteps a crucial part in the security proof, whereas the proof of Schnorr & Jakobsson assumes both generic groups [13] and random oracles [10]. Nevertheless, since using either assumption we can show that ElGamal+PoK construction is a voting-friendly scheme, we conclude that Helios satisfies ballot privacy under the same assumptions. Unfortunately, the security of the construction under standard assumptions is a long-standing open question. This observation motivates the search for alternative constructions of voting-friendly schemes.

Straight-line Extractors. To motivate the construction that we discuss now, it is instructive to explain why a proof that Enc+PoK is IND-CCA2 fails. In such a proof, when reducing the security of the scheme to that of the underlying primitive, a challenger would need to answer the decryption queries of the adversary. Since the underlying encryption scheme is only IND-CPA secure, the only possibility is to use the proof of knowledge to extract the plaintext underlying the queried ciphertexts. Unfortunately here the proof gets stuck. Current definitions and constructions for proofs of knowledge only consider single statements and the knowledge extractor works for polylogarithmically many proofs but it may break down (run in exponential time [19]) for polynomially many. Since the IND-CCA2 adversary is polynomially bounded answering all of its decryption queries may thus not be feasible.

A construction that gets around this problem employs a zero-knowledge proof of knowledge with a *straight-line* extractor. Such extractors do not need to rewind the prover and in this case the Enc+PoK construction yields an IND-CCA2 encryption scheme. This notion of extraction and a variation of the Fiat-

Shamir transform that turns a sigma-protocol into a non-interactive proof of knowledge with a straight-line extractor in the random oracle model has recently been proposed by Fischlin [22]. As above, starting with a homomorphic encryption scheme would yield a voting friendly encryption scheme. Unfortunately the construction in that paper is not suffficiently efficient to yield a practical encryption scheme.

The Naor-Yung Transformation. This transformation starts from any IND-CPA secure encryption scheme. An encryption of message m is simply two distinct encryptions c_1 and c_2 of m under the original scheme, together with a simulation-sound zero-knowledge proof π that c_1 and c_2 encrypt the same message with an extra property that we call unique applicability. Formally, we have the following definition.

Definition 9 (Naor-Yung Transformation). *Let $E = (\mathsf{EGen}, \mathsf{EEnc}, \mathsf{EDec})$ be a public-key encryption system. Let $P = (\mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ be a non-interactive zero-knowledge proof scheme for proving (in Camenisch's notation [15])*

$$\mathbf{PoK}\{(m, r_1, r_2) : \ c_1 = \mathsf{Enc}(y_1, m; r_1) \wedge c_2 = \mathsf{Enc}(y_2, m; r_2)\}$$

with uniquely applicable proofs. Assume the input to Prove *is given in the form $(m, y_1, y_2, r_1, r_2, c_1, c_2)$.*

The Naor-Yung transformation [7] $NY(E, P)$ of the encryption system is the public-key cryptosystem defined in Algorithm 6.

Algorithm 6 Naor-Yung Transformation

Gen
  $(x_1, y_1) \leftarrow \mathsf{EGen}$
  $(x_2, y_2) \leftarrow \mathsf{EGen}$
  return $((x_1, x_2), (y_1, y_2))$
$\mathsf{Enc}((y_1, y_2), m; (r_1, r_2))$
  $c_1 \leftarrow \mathsf{Enc}(y_1, m; r_1)$
  $c_2 \leftarrow \mathsf{Enc}(y_2, m; r_2)$
  $\pi \leftarrow \mathsf{Prove}(m, y_1, y_2, r_1, r_2, c_1, c_2)$
  return (c_1, c_2, π)
$\mathsf{Dec}(c_1, c_2, \pi)$
  if $\mathsf{Verify}(c_1, c_2, \pi) = 1$ **then return** $\mathsf{EDec}(x_1, c_2)$ **else return** $\perp$ **end if**

Sahai [18] showed that the above transformation yields an IND-CCA2 encryption scheme if the starting scheme is IND-CPA and the proof system is simulation-sound and has uniquely applicable proofs (essentially each proof can only be used to prove one statement).

Theorem 2 (Sahai[18]). *If the zero-knowledge proof system P has uniquely applicable proofs then the Naor-Yung transformation $NY(E, P)$ of an IND-CPA secure scheme E gives IND-CCA2 security.*

It turns out that if the starting encryption scheme is homomorphic, then the resulting construction is a voting-friendly encryption scheme. Indeed, the resulting scheme has a homomorphic embeding (given either the first or the second component of the ciphertext) and it is checkable (the checking algorithm only needs to verify the validity of π). As explained earlier, the required proof-systems for provability of the embeding exist, from general results. One can therefore obtain voting schemes with provable ballot privacy in the standard model starting from any homomorphic encryption scheme that is IND-CPA secure in the standard model.

In general, the above construction may not be very efficient (the simulation-sound zero-knowledge proof and associated required proof-systems may be rather heavy). In the random oracle model one can implement the above idea efficiently by replacing the simulation-sound zero-knowledge proof (of membership) with a zero-knowledge proof of knowledge of the message that underlies the two ciphertexts. Interestingly, one may regard the NY transform as providing the underlying encryption scheme with a straight-line extractor (so our previous results already apply).

The following theorem is a variation of the basic Naor-Yung transform applied to our setting.

Theorem 3. *If E is an IND-CPA secure homomorphic encryption scheme with S2P key derivation and P is a zero-knowledge proof of knowledge system with uniquely applicable proofs, then $NY(E, P)$ is a voting friendly encryption scheme.*

5 Application to the Helios protocol

We propose an enhanced version of Helios 3.0 which is an instantiation of our generic voting scheme with a voting-friendly encryption scheme obtained from ElGamal encryption [1] via the NY transform [7]. The required proof of knowledge is obtained via the Fiat-Shamir transform [3] applied to generalized Schnorr proofs. In this scheme duplicate ballots would be rejected as defined in the ProcessBallot procedure (Algorithm 3). We can further improve the efficiency by reusing some components as described by [20].

Thanks to Theorems 1 and 3, we deduce that the enhanced version of Helios 3.0 (provably) preserves ballot privacy. The modification of Helios we propose does not change the architecture nor the trust assumption of Helios and can be easily implemented. The computational overhead is reasonable (both the length of the messages and the time of computation would at most double and some optimizations can be foreseen). In exchange, we get the formal guarantee that Helios does preserve ballot privacy, a very crucial property in the context of electronic voting. For concreteness, we prove the details of the construction, as well as a proof of security in the full version of this paper.

We emphasize that our results go beyond proving ballot privacy of a particular e-voting protocol. We have identified IND-CCA2 as a sufficient condition for constructing voting schemes satisfying our notion of ballot privacy and have

given an abstract construction of a Helios-type voting scheme from IND-CPA secure homomorphic threshold encryption and non-interactive zero-knowledge proofs of knowledge. Our construction is independent of any hardness assumptions or security models (in particular, the random oracle model). We have formalized the concept of embeddable encryption and showed how to construct IND-CCA2 secure encryption with homomorphic embedding, despite the known impossibility of homomorphic IND-CCA2 secure encryption.

As further work, we plan to extend the definitions and proofs for threshold encryption scheme in order to have a fully complete proof for Helios. We are confident that our proof techniques will apply in a straightforward way. We also wish to investigate the possibility of defining ballot privacy in a more general way, e.g. allowing the current voting algorithm to be replaced by a protocol. Indeed, it could the case that casting a vote or tallying the vote require more than one step.

Acknowledgements We are very grateful to Ben Adida for helpful discussions on how to enhance ballot privacy in Helios.

This work was partially supported by the European Commission through the ICT Programme under Contract ICT- 2007-216676 ECRYPT II, by the Interuniversity Attraction Pole P6/26 BCRYPT, and by the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement number 258865 (ProSecure project). Olivier Pereira is a Research Associate of the Belgian Funds for Scientific Research (F.R.S.-FNRS).

References

1. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In: IEEE transactions on information theory, pages 469–472, Volume 31, 1985.
2. J. (Benaloh) Cohen and M. Fischer. A Robust and Verifiable Cryptographically Secure Election Scheme. In: Proceedings of the 26th Symposium on Foundations of Computer Science, pages 372–382, 1985.
3. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In: Proceedings on advances in cryptology (CRYPTO '86), pages 186–194, 1986.
4. J. Benaloh and M. Yung. Distributing the Power of a Government to Enhance the Privacy of Voters In: Proceedings of the 5th Symposium on Principles of Distributed Computing, pages 52–62, 1986.
5. J. Benaloh. Verifiable Secret-Ballot Elections. Yale University Department of Computer Science Technical Report number 561, 1987.
6. M.Blum, P. Feldman and S. Micali. Non-interactive zero-knowledge and its applications. In: 20th STOC, pages 103–112, 1988.
7. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In: Proceedings of the twenty-second annual ACM symposium on theory of computing (STOC '90), pages 42–437, 1990.
8. C. Schnorr. Efficient signature generation for smart cards. In: Journal of cryptology, Volume 4, pages 161–174, 1991.

9. I. Damgård. Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with preprocessing. In Eurocrypt, volume 658 of LNCS, pages 341–355, 1992.
10. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: Proceedings of the 1st ACM conference on Computer and communications security (CCS '93), pages 62–73, 1993.
11. J. Benaloh and D. Tuinstra. Receipt-Free Secret-Ballot Elections. In: Proceedings of the 26th ACM Symposium on Theory of Computing, pages 544–553, 1994.
12. R. Gennaro. Achieving independence efficiently and securely. In: Proceedings of the 14th Principles of Distributed Computing Symposium (PODC'95), pages 130–136, 1995.
13. V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In: Advances in Cryptology (EUROCRYPT '97), pages 256–266, 1997.
14. R. Cramer, R. Gennaro and B. Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In: Advances in Cryptology (EUROCRYPT '97), pages 103–118, 1997.
15. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In: Proceedings of the 17th annual international cryptology conference on advances in cryptology (CRYPTO '97), pages 410–424, 1997.
16. V. Shoup and R. Gennaro. Securing Threshold Cryptosystems Agains Chosen-Ciphertext Attack. In: Advances in Cryptology (Eurocrypt '98), LNCS 1403, pages 1–16, 1998.
17. Y. Tsiounis and M. Yung. On the security of ElGamal-based encryption. In: International Workshop on Practice and Theory in Public Key Cryptography (PKC '98), pages 117–134, 1998.
18. A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: Proceedings of th 40th annual symposium on foundations of computer science (FOCS '99), pages 543–553, 1999.
19. C.P. Schnorr and M. Jakobsson. Security of Signed ElGamal Encryption. In: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '00), pages 73–89, 2000.
20. M. Bellare, A. Boldyreva and J. Staddon. Multi-recipient encryption schemes: Security notions and randomness re-use. Full version, `http://cseweb.ucsd.edu/~mihir/papers/bbs.html`. Preliminary version in: Public key cryptography (PKC 2003), Lecture notes in computer science, Vol. 2567, 2003.
21. J. Groth. Evaluating Security of Voting Schemes in the Universal Composability Framework. Applied Cryptography and Network Security, ACNS 2004, pages 46–60, 2004.
22. M. Fischlin. Communication-Efficient Non-Interactive Proofs of Knowledge with Online Extractors. In: Proceedings of the 25th annual international cryptology conference on advances in cryptology (CRYPTO '05), pages 152–168, 2005.
23. A. Juels, D. Catalano and M. Jakobsson. Coercion-Resistant Electronic Elections. In: Proceedings of the 4th Workshop on Privacy in the Electronic Society (WPES'05), pages 61–70, 2005.
24. S. Kremer and M. D. Ryan. Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. In: 14th European Symposium on Programming (ESOP '05), pages 186–200, 2005.
25. T. Moran and M. Naor. Receipt-Free Universally-Verifiable Voting with Everlasting Privacy. In: Proceedings of the 26th International Cryptology Conference (CRYPTO'06), pages 373–392, 2006.

26. S. Delaune, S. Kremer and M. D. Ryan. Coercion-Resistance and Receipt-Freeness in Electronic Voting. 19th Computer Security Foundations Workshop (CSFW '06), pages 28–42, 2006.
27. B. Chevallier-Mames, P. Fouque, D. Pointcheval, J. Stern and J. Traoré. On Some Incompatible Properties of Voting Schemes. In: Proceedings of the Workshop on Trustworthy Elections (WOTE'06), 2006.
28. Participants of the Dagstuhl Conference on Frontiers of E-Voting. Dagstuhl Accord. `http://www.dagstuhlaccord.org/`, 2007.
29. J. Benaloh. Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. In: Proceedings of the Second Usenix/ACCURATE Electronic Voting Technology Workshop, 2007.
30. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '98), pages 13–25, 2008.
31. M. R. Clarkson, S. Chong and A. C. Myers, Civitas: Toward a Secure Voting System. In: Proceedings of the 29th Security and Privacy Symposium (S&P'08), pages 354–368, 2008.
32. B. Adida. Helios: Web-based open-audit voting. In: 17th USENIX security symposium, pages 335–348, 2008. `http://www.usenix.org/events/sec08/tech/full_papers/adida/adida.pdf`
33. M. Backes, C. Hriţcu and M. Maffei. Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-calculus. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF'08), pages 195–209, 2008.
34. D. Wikström. Simplified Submission of Inputs to Protocols. In: Security and Cryptography for Networks, 6th International Conference, SCN 2008, pages 293–308, 2008.
35. B. Adida, O. de Marneffe, O. Pereira and J.-J. 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.
36. International association for cryptologic research. Election page at `http://www.iacr.org/elections/2010`
37. V. Cortier and B. Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. Website with description and video at `http://www.bensmyth.com/publications/10-attacking-helios/` Cryptology ePrint Archive, Report 2010/625.
38. S. Kremer, M. D. Ryan and B. Smyth. Election verifiability in electronic voting protocols. In: Proceedings of the 15th European Symposium on Research in Computer Security (ESORICS'10), pages 389–404, 2010.
39. D. Unruh and J. Müller-Quade. Universally Composable Incoercibility. In: Proceedings of the 30th International Cryptology Conference (CRYPTO'10), pages 411–428, 2010.
40. R. Küsters, T. Truderung and A. Vogt. A Game-Based Definition of Coercion-Resistance and its Applications. In: Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF'10), pages 122–136, 2010.
41. J. Loftus, A. May, N.P. Smart and F. Vercauteren. On CCA-Secure Fully Homomorphic Encryption `http://eprint.iacr.org/2010/560`
42. V. Cortier and B. Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. To appear in: Proceedings of the 24th Computer Security Foundations Symposium (CSF '11), 2011.

43. R. Küsters, T. Truderung and A. Vogt. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. Preprint, to appear at the 32nd Security and Privacy Symposium (S&P'11).
44. G. Persiano. About the Existence of Trapdoors in Cryptosystems. "Work in Progress", Available at http://libeccio.dia.unisa.it/Papers/Trapdoor/.
45. Helios voting. Website: http://heliosvoting.org
46. Helios Headquarters, Princeton University Undergraduate Student Government. http://usg.princeton.edu/officers/elections-center/helios-headquarters.html

A Details of our construction

We give a detailed description of our proposed construction of a provably secure extension of Helios.

A.1 Choice of groups

Let λ be a security parameter. Pick primes p, q such that $p - 1 = kq$ for some $k \in \mathbb{Z}$ and $q \approx \lambda$. Pick an element $g \in (\mathbb{Z}_p^*, \times)$ of order q. This defines a cyclic group of prime order q which we will call G. The group operation is integer multiplication with reduction mod p.

Occasionally we need the group $(\mathbb{Z}_q, +)$ directly. In this group the operation is integer addition with reduction mod q. Although these groups are isomorphic, for security we rely on the fact that we have a homomorphism $\mathbb{Z}_q \to G$ and the conjecture that it is hard to find one in the opposite direction.

The choice of these groups and parameters can be done as in existing Helios implementations. We summarize these groups in the following table.

Group	Order	Op.	Modulus	Neutral	Generator
G	q	$\times$	p	1	g
$\mathbb{Z}_q$	q	$+$	q	0	1

A.2 Key generation

An election is created by a number of trustees who each hold a share of the decryption/tallying key. To create an election for s trustees, each trustee $\mathcal{T}_i$ picks two secrets $x_{(i,0)}, x_{(i,1)} \xleftarrow{R} \mathbb{Z}_q$ where $i \leftarrow 1 \ldots s$. Each trustee then computes two values $y_{(i,b)} = g^{x_{(i,b)}} \mod p$ for $b \in \{0, 1\}$ and publishes these values. The election public key is the pair

$$\left(y_0 = \prod_{i=1}^{s} y_{(i,0)},\ y_1 = \prod_{i=1}^{s} y_{(i,1)} \right)$$

A trustee must also prove knowledge of his secrets with a proof of knowledge of a discrete logarithm. Although the proof of correct decryption of the election result will also include a proof of knowledge of the trustees' secret keys, voters must be able to verify that the election public keys are well-formed before they use them to encrypt their votes. The proof can be done as follows by trustee $\mathcal{T}_i$ and repeated for both $j = 0$ and $j = 1$. It is a straightforward Schnorr proof:

1. $\alpha_j \xleftarrow{R} \mathbb{Z}_q$.
2. $\beta_j \leftarrow g^{\alpha_j} \mod p$.
3. $\gamma_j \leftarrow H(\beta_j)$.
4. $\delta_j \leftarrow \alpha_j + \gamma_j \cdot x_{(i,j)} \mod q$.

The proof is the tuple $(\beta_0, \delta_0, \beta_1, \delta_1)$ and can be checked by ensuring $\beta_j \in G, \delta_j \in \mathbb{Z}_q$ and $g^{\delta_j} = \beta_j \cdot y_{(i,j)}^{\gamma_j} \mod p$ for $j \in \{0, 1\}$.

A.3 Creating a ballot

A ballot encodes a number of votes. The election parameters describe

- The number n_V of votes each voter must cast.
- For each vote i in $1 \dots n_V$, a range $(\min_i, \max_i)$ that the vote must lie in.
- A range $(\min_V, \max_V)$ that the sum of all votes must lie in.

To create a ballot for votes $v_1 \dots v_{n_V}$ satisfying these conditions, a voter performs the following algorithm. In the interest of readability, we split the algorithm into several distinct procedures as is good practice in programming and give comments and pre/postconditions for each.

create-ballot This is the main algorithm that takes a list of votes and creates a ballot.
Input: Number of votes n_V, votes $v_1 \dots v_{n_V}$, ranges $(\min_i, \max_i)_{i=1}^{n_V}$, range $(\min_V, \max_V)$, public key (y_0, y_1).
Preconditions: $n_V \geq 1$, for all $i \in 1 \dots n_V$ we have $\min_i \leq \max_i$, $\min_V \leq \sum_{i=1}^{n_V} \min_i \leq \sum_{i=1}^{n_V} \max_i \leq \max_V$.
Output: Valid ballot for $v_1 \dots v_{n_V}$.

1. For $i \leftarrow 1 \dots n_V$:
   (a) $r_i \xleftarrow{R} \mathbb{Z}_q$.
   (b) $e_i \leftarrow \mathsf{encrypt\text{-}vote}(y_0, y_1, v_i, r_i)$.
   (c) $\pi_{i,NY} \leftarrow \mathsf{create\text{-}NY\text{-}proof}(y_0, y_1, r_i, e_i)$.
   (d) $\pi_{i,R} \leftarrow \mathsf{create\text{-}range\text{-}proof}(y_0, y_1, r_i, v_i, e_i, \min_i, \max_i)$.
2. $\pi_V \leftarrow \mathsf{create\text{-}sum\text{-}proof}(y_0, y_1, (e_i, r_i, v_i)_{i=1}^{n_V}, \min_V, \max_V)$.
3. Return the ballot
$$((e_i, \pi_{i,NY}, \pi_{i,R})_{i=1}^{n_V}, \pi_V)$$

encrypt-vote This procedure creates a 3-element ElGamal encryption of a vote.
Input: Vote v, randomness r, public keys y_0, y_1.
Preconditions: None.
Output: Encrypted vote $e = (a, b, c)$.
Compute
$$(a \leftarrow g^r,\ b \leftarrow g^v y_0^r,\ c \leftarrow g^v y_1^r) \mod p$$
and return $e \leftarrow (a, b, c)$.

create-NY-proof This procedure takes an encrypted vote e and the underlying randomness r and produces a proof that both encryptions are for the same message under the same randomness.
Input: Public keys y_0, y_1, randomness r, encrypted vote e.
Preconditions: e is a valid encryption of a vote with randomness r under public keys y_0, y_1.
Output: Non-interactive zero-knowledge proof π.

1. $\alpha_1, \alpha_2 \xleftarrow{R} \mathbb{Z}_q$.
2. $(\beta_1 \leftarrow g^{\alpha_2},\ \beta_2 \leftarrow g^{\alpha_1} y_0^{\alpha_2},\ \beta_3 \leftarrow g^{\alpha_1} y_1^{\alpha_2}) \mod p$.
3. $\gamma \leftarrow H(\beta_1 \| \beta_2 \| \beta_3)$.
4. $\delta_1 \leftarrow \alpha_1 + \gamma \cdot m \mod q,\ \delta_2 \leftarrow \alpha_2 + \gamma \cdot r \mod q$.
5. Return $\pi \leftarrow (\beta_1, \beta_2, \beta_3, \delta_1, \delta_2)$.

create-range-proof This procedure produces a zero-knowledge proof that a vote lies in a given range. Note that we only consider the first public key which is sufficient as the zero-knowledge proof of equality guarantees that the second encryption will be in the same range.
Input: Public keys y_0, y_1, encrypted vote e, vote v, randomness r, range of valid votes $(\min, \max)$.
Preconditions: e is the encryption of v under public keys y_0, y_1 and randomness r and $v \in [\min, \max]$.
Output: Zero-knowledge proof π.

1. Parse e as (a, b, c).
2. For $j \leftarrow [\min, \max] \setminus \{v\}$, create a simulated proof:
   (a) $\gamma_j, \delta_j \xleftarrow{R} \mathbb{Z}_q$.
   (b) $\Delta_{j,1} \leftarrow g^{\delta_j} \mod p,\ \Delta_{j,2} \leftarrow y_0^{\delta_j} \mod p$.
   (c) $\beta_{j,1} \leftarrow \Delta_{j,1} - \gamma_i \cdot a \mod p,\ \beta_{j,2} \leftarrow \Delta_{j,2} - \gamma_i \cdot (b/g^j) \mod p$.
3. $\alpha_v \xleftarrow{R} \mathbb{Z}_q$.
4. $\beta_{v,0} \leftarrow g^{\alpha_v} \mod p,\ \beta_{v,1} \leftarrow y_0^{\alpha_v} \mod p$.
5. $\gamma \leftarrow H(\beta_{\min,1} \| \beta_{\min,2} \| \beta_{\min+1,1} \| \beta_{\min+1,2} \| \ldots \| \beta_{\max,1} \| \beta_{\max,2})$.
6. $\gamma_v \leftarrow \gamma - \sum_{j=\min, j \neq v}^{\max} \gamma_j \mod q$.
7. $\delta_v \leftarrow \alpha_v + \gamma_v \cdot r$.
8. Return $(\beta_{j,1}, \beta_{j,2}, \gamma_j, \delta_j)_{j=\min}^{\max}$.

create-sum-proof This procedure produces a zero-knowledge proof that the sum of all votes $(v_1, \ldots, v_{n_V})$ underlying a single ballot lies in the range $[\min_V, \max_V]$. This is done by homomorphically adding all votes and then producing a range proof on the sum. Sum proofs, like range proofs, need only be done on one of the two encryption.
Input: Public keys y_0, y_1, encrypted votes, underlying randomness and plain votes $(e_i, r_i, v_i)_{i=1}^{n_V}$, range $(\min_V, \max_V)$.
Preconditions: Valid encrypted votes for given public keys, plain votes and randomness; sum of all plain votes lies in range $[\min_V, \max_V]$.
Output: Zero-knowledge proof π.

1. Parse each encrypted vote as (a_i, b_i, c_i).
2. $(a, b) \leftarrow (\prod_{i=1}^{n_V} a_i \mod p,\ \prod_{i=1}^{n_V} b_i \mod p)$.
3. $(r, v) \leftarrow (\sum_{i=1}^{n_V} r_i \mod q, \sum_{i=1}^{n_V} v_i \mod q)$.
4. Return create-range-proof$(y_0, y_1, r, v, (a, b, \perp), \min_V, \max_V)$.

A.4 Verifying a ballot

When a ballot is cast, the entity responsible for collecting ballots must check its validity and publish it if found valid. This process must be publicly verifiable, indeed anyone must be able to check ballot validity without access to any secret information.

Ballot authentication is out of scope of this discussion, for simplicity we may assume authentic channels from all voters to the ballot casting centre. We explicitly include checks that all elements are in the required groups.

verify-ballot This algorithm takes a ballot and returns 1 if it is deemed to be valid, otherwise 0.
Input: A claimed ballot of the form $((e_i, \pi_{i,NY}, \pi_{i,R})_{i=1}^{n_V}, \pi_V)$.
Preconditions: None (we are checking if the input is valid, after all).
Output: 0 or 1.

Wherever "check" is written, the algorithm immediately aborts returning 0 if a check fails.

1. Check that the number of elements is correct.
2. For $i \leftarrow 1 \dots n_V$
   (a) Parse e_i as (a, b, c). Check all three elements are in G.
   (b) Check verify-NY$(y_0, y_1, e_i, \pi_{i,NY})$.
   (c) Check verify-range$(y_0, y_1, e_i, \pi_{i,R})$.
3. $(a', b') \leftarrow (\prod_{i=1}^{n_V} a_i, \prod_{i=1}^{n_V} b_i) \mod p$.
4. Check verify-range$(y_0, y_1, (a', b', \perp), \pi_V)$.

verify-NY This procedure verifies a Naor-Yung transformation proof that two encryptions share the same message and randomness.
Input: Public key (y_0, y_1), encrypted vote $e = (a, b, c)$, Naor-Yung proof $\pi = (\beta_1, \beta_2, \beta_3, \delta_1, \delta_2)$.
Preconditions: $a, b, c \in G$.
Output: 0 or 1.

1. Check that $\beta_1 \in G, \beta_2 \in G, \beta_3 \in G, \delta_1 \in \mathbb{Z}_q, \delta_2 \in \mathbb{Z}_q$ or return 0 if this fails.
2. $\Delta_1 \leftarrow g^{\delta_2} \mod p$, $\Delta_2 \leftarrow g^{\delta_1} y_0^{\delta_2} \mod p$, $\Delta_3 \leftarrow g^{\delta_1} y_1^{\delta_2} \mod p$.
3. $\gamma \leftarrow H(\beta_1 \| \beta_2 \| \beta_3)$.
4. Return 1 if the following checks pass, otherwise 0:
   - $\Delta_1 \stackrel{?}{=} \beta_1 \cdot a^{\gamma} \mod p$.
   - $\Delta_2 \stackrel{?}{=} \beta_2 \cdot b^{\gamma} \mod p$.
   - $\Delta_3 \stackrel{?}{=} \beta_3 \cdot c^{\gamma} \mod p$.

verify-range This procedure verifies a zero-knowledge proof that an encrypted vote lies within a given range.

Input: Public key (y_0, y_1), encrypted vote (a, b, c), disjunctive proof $(\beta_{j,1}, \beta_{j,2}, \gamma_j, \delta_j)_{j=\min}^{\max}$.
Preconditions: $a, b, c \in G$, encrypted vote NY-verified.
Output: 0 or 1.

1. For $j \leftarrow \min, \ldots, \max$:
   (a) Check that $\beta_{j,1} \in G, \beta_{j,2} \in G, \gamma_j \in \mathbb{Z}_q, \delta_j \in \mathbb{Z}_q$ or return 0 if this fails.
   (b) $\Delta_{j,1} \leftarrow g^{\delta_j} \mod p$, $\Delta_{j,2} \leftarrow y_0^{\delta_j} \mod p$.
   (c) Return 0 if any of the following two checks fail:
      – $\Delta_{j,1} \stackrel{?}{=} \beta_{j,1} \cdot a^{\gamma_j} \mod p$.
      – $\Delta_{j,2} \stackrel{?}{=} \beta_{j,2} \cdot (b/g^j)^{\gamma_j} \mod p$.
2. $\gamma \leftarrow H(\beta_{\min,1} \| \beta_{\min,2} \| \beta_{\min+1,1} \| \beta_{\min+1,2} \| \ldots \| \beta_{\max,1} \| \beta_{\max,2})$.
3. Check that $\gamma \stackrel{?}{=} \sum_{j=\min}^{\max} \gamma_j$. Return 1 if this succeeds, otherwise 0.

A.5 Computing a Tally

A group of trustees each hold a share of the decryption key. Once voting is over, they must each verify all ballots and reject any that fail the checks. It is also advisable that they check for duplicate proofs among all valid ballots. After this stage, all zero-knowledge proofs and even the third elements of triples forming encrypted votes can be discarded.

Each trustee should independently compute the sum of all votes. Specifically, if the ballot of voter i contains the n_V encryptions $(a_{i,j}, b_{i,j})_{j=1}^{n_V}$ then the trustees should compute

$$\left(a_j \leftarrow \prod_{i=1}^{n} a_{i,j},\ b_j \leftarrow \prod_{i=1}^{n} b_{i,j} \right)_{j=1}^{n_V}$$

Although this must be publicly computable to verify the election, it is important that each trustee does not apply his decryption key to anything until he is sure it is the correct sum.

The trustees can each compute a partial decryption by applying their secret key share: The trustee holding x_k (which we can take to be $x_{(k,0)}$) computes $(d_{j,k} \leftarrow (a_j)^{x_k} \mod p)_{j=1}^{n_V}$ and publishes this value along with a proof of knowledge attesting to the fact that they decrypted correctly. Such a proof (to be repeated for $j \leftarrow 1 \ldots n_V$) takes the form

$$\mathbf{PoK}\{(x_k) : d_{j,k} = a_j^{x_k} \mod p \ \wedge\ y_k = g^{x_k} \mod p\}$$

and is computed as follows.

1. $\alpha \stackrel{R}{\leftarrow} \mathbb{Z}_q$
2. $(\beta_1, \beta_2) \leftarrow (a_j^{\alpha}, g^{\alpha}) \mod p$.
3. $\gamma \leftarrow H(\beta_1 \| \beta_2)$.
4. $\delta \leftarrow \alpha + \gamma \cdot x_k \mod q$.

The proof consists of the elements $(\beta_1, \beta_2, \delta)$ and verification involves checking the following three conditions.

1. β_1 and β_2 are in G and δ in $\mathbb{Z}_q$.
2. $(a_j)^\delta = \beta_1 \cdot d_{j,k}^{H(\beta_1 \| \beta_2)} \mod p$.
3. $g^\delta = \beta_2 \cdot y_k^{H(\beta_1 \| \beta_2)} \mod p$.

The decrypted tally is then $(d_j \leftarrow b_j / \prod_{k=1}^{n_K} d_{j,k})_{j=1}^{n_V}$ where n_K is the number of key shares. This will satisfy the equation $(d_j = g^{m_j} \mod p)$ where m_j is the sum of all jth votes (for $j \leftarrow 1 \ldots n_V$). The actual result m_j can be recovered by computing powers of g modulo p until all values d_j are found.

A.6 Simulating a Tally

In the ballot privacy game for $b = 1$, we need to simulate these proofs. The values $d_{j,k}$ must be computed correctly and the proof simulated as follows.

1. $\gamma \xleftarrow{R} \mathbb{Z}_q,\ \delta \xleftarrow{R} \mathbb{Z}_q$
2. $\beta_1 \leftarrow (a_j)^\delta / (d_{j,k})^\gamma \mod p$
3. $\beta_2 \leftarrow (g)^\delta / (y_k)^\gamma \mod p$
4. Patch H at input $(\beta_1 \| \beta_2)$ to equal γ.

A.7 Verifying the Election

Anyone can publicly perform the following verification steps.

- Check each published ballot for validity.
- Check there are no duplicate proofs among published ballots.
- Compute the encryption of the tally from the ballots using the homomorphic property of the ElGamal encryption.
- Check the proofs of correct partial decryption.
- Compute the tally from the partial decryptions.

B Ballot Privacy for the Mini–Voting Scheme

For the proof, we first consider a "mini-voting" scheme (in Theorem 4) that removes all the extra zero-knowledge proofs concerning verifiability and show that ballot privacy follows from IND-CCA2 security of the encryption scheme. Then, we extend this proof to our construction (in Theorem 1).

We stress that we do not require any additional conditions on the zero-knowledge proofs (for verifiability); in particular, they need not be non-malleable.

Theorem 4. *Let* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be an IND-CCA2 secure encryption scheme for message space* $\mathbb{V}$*. Then the mini-voting scheme in Algorithm 7 is a voting scheme with ballot privacy.*

Algorithm 7 Mini-Voting Scheme

Setup
  $(x, y) \leftarrow \mathsf{Gen}$
  $\mathcal{Z} \leftarrow y$
  $BB \leftarrow ()$
Vote(v)
  $b \leftarrow \mathsf{Enc}(v)$
ProcessBallot(c, BB)
  if $b \notin BB$ **then**
    $BB \xleftarrow{+} b$
    return ("accept", BB)
  else
    return ("reject", BB)
  end if
Tally(x, BB)
  for all $b_j \in BB$ **do** $v'_j \leftarrow \mathsf{Dec}(b_j)$ **end for**
  $r \leftarrow \rho((v'_j)_{j=1}^{|BB|})$

Proof (Theorem 4). Suppose $\mathcal{A}$ is an adversary that has non-negligible advantage in winning the ballot privacy game.

Let n be an upper bound on the number of vote calls the adversary makes. Let $(H_k)_{k=0}^{n}$ be the sequence of games where in game H_k, the first k calls to vote are answered as if $\beta = 0$ and all from the $k+1$st onwards are answered as if $\beta = 1$. We note that H_0 is equivalent to the ballot privacy game for $\beta = 0$ and H_n for $\beta = 1$.

Using the triangular inequality, if any adversary can distinguish H_0 from H_n with non-negligible advantage then there is a value k such that the adversary can distinguish the pair of hybrids (H_k, H_{k+1}) with non-negligible advantage.

Given such a k, we construct a reduction to IND-CCA2 of the underlying scheme. We assume w.l.o.g. that in the $k+1$st call to vote, the vote is not ε or else the two hybrids would be identical. Let $\mathcal{C}$ be a challenger for IND-CCA2 of the encryption scheme. We construct an adpater $\mathcal{B}$ which will play against $\mathcal{C}$ using $\mathcal{A}$ as follows.

Setup $\mathcal{B}$ receives a public key y from $\mathcal{C}$, performs the setup of the voting scheme and hands the public key and parameters to $\mathcal{A}$.

vote

queries $1 \ldots k$**:** $\mathcal{B}$ processes these as in the ballot privacy game with $\beta = 0$.

query $k+1$**:** Let v^* be the input to this query. If it is $\perp$, $\mathcal{B}$ does nothing. If not, it stores v^* and forwards the pair (v^*, ε) to $\mathcal{C}$ and gets a ciphertext c^* back. It processes c^* onto BB and creates a new ciphertext $c' = \mathsf{Vote}(v^*)$ which it processes onto BB'. Finally, $\mathcal{B}$ returns the new state of BB to $\mathcal{A}$.

queries $\geq k+2$**:** $\mathcal{B}$ processes these as in the ballot privacy game with $\beta = 1$.

ballot
: For all queries before the $k+1$st vote query (when $\mathcal{B}$ obtains the IND-CCA2 challenge), it passes the ballot to the decryption oracle and stores the plaintext and ballot. $\mathcal{B}$ then handles the query like the ballot privacy experiment.

Tallying As long as c^* does not appear on BB', $\mathcal{B}$ decrypts all ballots on BB' (using the decryption oracle of $\mathcal{C}$) to get the votes, evaluates ρ on these votes and returns the result to $\mathcal{A}$. We deal with the case that c^* is on BB' below.

Guess $\mathcal{B}$ forwards the bit $\widehat{\beta}$ it receives from $\mathcal{A}$ to $\mathcal{C}$.

As long as c^* does not appear on BB', the adversary's view when interacting with $\mathcal{B}$ will be identically distributed to that when he is interacting with $H_{k+\beta}$ where β is the bit chosen by $\mathcal{C}$ and therefore $\mathcal{B}$ wins against $\mathcal{C}$ with the same advantage as $\mathcal{A}$ has in distinguishing the two hybrids.

Suppose c^* appears on BB' and consider the following cases. In each case, we show that $\mathcal{B}$ obtains the plaintext of c^* and can therefore guess β with probability 1 as we assumed $v^* \neq \varepsilon$.

1. c^* was added while processing a ballot command *before* the challenge query. In this case, $\mathcal{B}$ has decrypted c^* and stored the plaintext/ciphertext pair while processing the command.
2. c^* was added while processing a ballot command *after* the challenge query. This is impossible as c^* is definitely on BB after the challenge query and so further calls to ballot(c^*) would reject before trying to process c^* onto BB'.
3. c^* was added while processing a vote query.
   In this case, $\mathcal{B}$ knows the plaintext because it receives it as input.

Therefore, $\mathcal{B}$ has at least as high an advantage against $\mathcal{C}$ as $\mathcal{A}$ has of distinguishing H_k from H_{k+1}. □

Proof (Theorem 1). We will prove the theorem by a sequence of "game hops". Starting with the mini-voting scheme, we modify the voting scheme in each hop and argue how the proof needs to be adapted.

Let *Scheme 2* be the mini-voting scheme where each voter may cast k votes. Duplicate checking is performed as in our full voting scheme, i.e. we extract the list of ciphertexts from each ballot and reject if any of them is a duplicate. For n voters, this gives us a total of $N := n \cdot k$ encrypted votes and we simply run the same hybrid argument (with N hybrids).

Next, we add the proofs of knowledge everywhere to obtain *Scheme 3*. In the security reduction, we perform two hybrid arguments. The first one replaces the zero-knowledge proofs by simulated ones, one step at a time. The second one switches votes to ε as before. If the adversary can detect a difference between any two consecutive hybrids, he can either attack the encryption (as argued earlier) or distinguish a real from a simulated zero-knowledge proof.

Finally, we switch the tallying operation to perform checking, extraction, homomorphic tallying and decryption of the result only to obtain our proposed scheme. (If ρ is not homomorphically computable, we skip this step as our scheme

cannot tally homomorphically either.) We extend our sequence of games by 1, where the first hop is switching from homomorphic decryption back to decrypting all ballots. The adversary will not notice any difference between the two games as he is not involved in the decryption and tallying process, so ballot privacy still holds.

□

C Security proof of the Naor-Yung transformation

C.1 The Naor-Yung Transformation

We prove security of the Naor-Yung transformation [7].

Naor and Yung originally used zero-knowledge proofs of language membership yet their technique and proof can easily be adapted to proofs of knowledge. This adaptation also removes a substantial amount of the complexities they had to deal with in their original paper.

Naor and Yung proved IND-CCA1 of their transformation. Sahai [18] realized that this transformation even gives an IND-CCA2 secure encryption scheme if the zero-knowledge proof system satisfies a few extra conditions. All of these hold trivially for proofs of knowledge except the one he called *uniquely applicable proofs*; we will have to show that this holds for our proposed scheme.

We prove IND-CCA2 security of the Naor-Yung transformation two steps. First, we prove Naor and Yung's theorem [7] giving us IND-CCA1 security. The original theorem concerned zero-knowledge proofs of *language membership* and used specific definitions and propositions concerned with such proofs. Our version omits all these as we deal with the simpler case of proofs of *knowledge*. The main ideas in the proof remain unchanged although we use techniques from [20] to simplify the presentation.

Proof (Naor-Yung). Let $\mathcal{C}$ be a challenger for the IND-CPA game of the original encryption and $\mathcal{A}$ be an adversary for the IND-CCA1 game of the Naor-Yung transformed scheme. We construct an algorithm $\mathcal{B}$ that attacks $\mathcal{C}$ using $\mathcal{A}$.

When $\mathcal{B}$ is invoked by $\mathcal{A}$, it acts as follows.

Setup
1. Receive a public key y from $\mathcal{C}$.
2. Create a key pair (x', y') using Gen.
3. Pick a bit $b' \xleftarrow{R} \{0,1\}$.
4. If $b' = 0$, let $(y_1, y_2) \leftarrow (y, y')$. Otherwise, let $(y_1, y_2) \leftarrow (y', y)$ i.e. b' determines the order of the keys.
5. Send (y_1, y_2) to $\mathcal{A}$.

$\mathsf{Dec}(\mathsf{c})$ Parse c as (c_0, c_1, π). Check π is valid for c_0, c_1 and abort returning $\perp$ if this fails. Decrypt $c_{1-b'}$ with x' and return the result m.

$\mathsf{Chal}(\mathsf{m_0}, \mathsf{m_1})$
1. Pass (m_0, m_1) to the Chal interface of $\mathcal{C}$ and get a ciphertext c back.
2. Create a new ciphertext $c' \leftarrow \mathsf{Enc}(y', m_{b'})$.

3. If $b' = 0$, let $(c_0, c_1) \leftarrow (c, c')$. If $b' = 1$, swap the order of ciphertexts: $(c_0, c_1) \leftarrow (c', c)$.
4. Simulate a proof $\pi \leftarrow \mathsf{Sim}(c_0, c_1)$.
5. Return (c_0, c_1, π) to the adversary.

Result When $\mathcal{A}$ produces a result $d \in \{0, 1\}$ forward this to $\mathcal{C}$.

The public keys in **Setup** are distributed identically to those in the Naor-Yung transformed encryption scheme. The decryption oracle will work correctly for $\mathcal{A}$ as long as he does not manage to forge a proof and submit an invalid ciphertext; the probability of this happening is negligible as we assume the proof system to be sound.

Consider the distributions $\mathcal{A}$ sees based on the bits b of $\mathcal{C}$ and b' of $\mathcal{B}$. For $x, y \in \{0, 1\}$ define the distribution

$$\delta(x, y) := (\mathsf{Enc}(y_0, m_x), \mathsf{Enc}(y_1, m_y))$$

over the random choices in both encryptions, assuming the public keys and messages are fixed. $\mathcal{A}$ sees two elements (c_0, c_1) and a proof π in response to his **Chal** query. The distribution of these ciphertexts will be

	$b = 0$	$b = 1$
$b' = 0$	$\delta(0, 0)$	$\delta(1, 0)$
$b' = 1$	$\delta(1, 0)$	$\delta(1, 1)$

The advantage of $\mathcal{A}$ against IND-CCA1 security of the Naor-Yung transformed scheme is his advantage in distinguishing $\delta(0, 0)$ from $\delta(1, 1)$. If this is non-negligible, then by the triangular inequality his advantage between two successive distributions in the sequence

$$\delta(0, 0) \;-\; \delta(1, 0) \;-\; \delta(1, 1)$$

will also be non-negligible. In fact, as $\mathcal{B}$ chooses b' uniformly at random from $\{0, 1\}$ we can estimate

$$\mathbf{Adv}^{\mathsf{CPA}}(\mathcal{B}) \geq \frac{1}{2} \cdot \mathbf{Adv}^{\mathsf{CCA1}}(\mathcal{A}) - \mathbf{Adv}^{\mathsf{P}}(\mathcal{A})$$

Where $\mathbf{Adv}^{\mathsf{P}}$ is the probability that $\mathcal{A}$ can attack the zero-knowledge proof, either by forging a proof for an incorrect encryption or by distinguishing a real proof from a simulated one (on a correct or incorrect encryption). Assuming the proof system is sound and zero-knowledge, this quantity is negligible. □

We now prove Sahai's version of the theorem that gives us IND-CCA2 security. Sahai proved the theorem directly whereas we just need to extend from IND-CCA1 to IND-CCA2 security. Sahai used proofs of language membership, as we consider proofs of knowledge we can again omit many of the technicalities. The central argument remains the same, namely showing that the adversary cannot ask for a decryption of an invalid ciphertext.

Proof (Sahai). We claim that for any triple (c_0, c_1, π) submitted to the decryption oracle, if the proof verifies then with overwhelming probability both c_0 and c_1 are encryptions of the same message.

There is only one case when the adversary sees a proof he has not created himself (so we cannot use the extractor of the proof of knowledge to get a contradiction) and that is the proof returned from the challenge query.

Uniquely applicable proofs give us that the adversary cannot create a triple (c_0^*, c_1^*, π) distinct from the response of the challenge oracle, passing verification using the same proof π. In the definition of IND-CCA2, the adversary is not allowed to ask for a decryption of the triple he got from the challenge oracle either.

For any triple with a proof $\pi^* \neq \pi$ distinct from that we returned from the challenge query, if the underlying statement is false then the adversary has produced a forgery which we assume can happen with at most negligible probability. □

(The property that the adversary may not prove a false statement even after seeing a proof for a false statement is called simulation-soundness in the literature. For a proof of knowledge with uniquely applicable proofs, this is trivial.)

We can now prove our theorem that we get a voting-friendly scheme.

Proof (Theorem 3). IND-CCA2 follows from Sahai's theorem. The ExtractKey and Extract algorithms extract the first key pair and ciphertext respectively; this yields the embedding property. EAdd is just the homomorphic addition algorithm of the original scheme which clearly can still be applied.

D Reusing randomness

Recall that a homomorphic ElGamal ciphertext is a pair of the form $(g^r, g^m y^r)$ mod p. For the Naor-Yung construction, we could simply use a 4-tuple of the form

$$(g^{r_1}, g^m y_1^{r_1}, g^{r_2}, g^m y_2^{r_2}) \mod p$$

but in fact we can do better thanks to a theorem of Bellare, Boldyreva and Staddon [20] which allows us to reuse the random value r.

We state the crucial condition for randomness reuse and the theorem informally and without proof, as it covers a much more general case than we require.

An encryption scheme is said to be *reproducible* if there is an efficient algorithm Reproduce that, on input a public key and an encryption of some unknown message under this public key together with a new message and a new public/secret key pair, produces an encryption of the new message under the new public key using *the same randomness* as the old encryption.

More formally, let $\mathsf{Enc}(pk, m; r)$ denote the *deterministic* encryption function, where r is the randomness. Then we demand $\forall (pk, sk), m, r, (pk', sk')$ where (pk, sk) and (pk', sk') are valid key pairs

$$c = \mathsf{Enc}(pk, m; r) \rightarrow \mathsf{Reproduce}(pk, c, m', pk', sk') = \mathsf{Enc}(pk', m'; r)$$

ElGamal is reproducible: Given a ciphertext $c = (a, b)$ we can reproduce one for a new key as

$$\mathsf{Reproduce}(pk, c, m', pk', sk') := (a, g^{m'} a^{sk'})$$

Writing $a = g^r, b = g^m pk^r$ we find that the new element is $g^{m'}(g^r)^{sk'} = g^{m'}(pk')^r$ as required.

Informally, the relevant theorem in [20] states that if a public-key encryption is reproducible and IND-CPA, CCA1 or CCA2 then it retains this security if encryption is performed with the same randomness for multiple keys. Their definition of multi-recipient IND-CPA/CCA also takes into account that the adversary may be a legitimate participant in the system and know some of the decryption keys too. We refer to the original paper for the precise definitions and the full proof, we do not require these for our construction.

An encryption of message m with randomness r under keys y_1, y_2 will be the triple

$$(g^r, g^m y_1^r, g^m y_2^r) \mod p$$

Apart from reducing bandwidth and computation required for the actual encryption compared to two separate encryptions, this also allows us to construct an easier zero-knowledge proof for the Naor-Yung transformation.

The proof of the Naor-Yung transformation is almost identical when reusing randomness, the only change is that we use $\mathsf{Randomize}$ in the challenge oracle to create the second ciphertext.

E Encryption

E.1 Public-Key Encryption

A public-key encryption scheme is a triple of probabilistic algorithms

$$(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$$

satisfying the correctness condition for all messages m in the message space with probability 1.

The algorithms and their input/output values are as follows.

$\mathsf{Gen}(1^\lambda)$ The key-generation algorithm takes a security parameter λ and generates a secret key x and a public key y.

$\mathsf{Enc}(y, m)$ The encryption algorithm takes a public key y and a message m and outputs a ciphertext c.

$\mathsf{Dec}(x, c)$ The decryption algorithm takes a secret key x and a ciphertext c and outputs a decryption d.

The scheme is IND-CPA, IND-CCA1 or IND-CCA2 secure if no efficient adversary can win the security game with non-negligible advantage, defined as

$$\mathbf{Adv}(\mathcal{A}) = \left| \mathbf{Pr}[\mathcal{A} \text{ wins}] - \frac{1}{2} \right|$$

Algorithm 8 Correctness of Public-Key Encryption

Parameters: m

$(x, y) \leftarrow \mathsf{Gen}$
$c \leftarrow \mathsf{Enc}(y, m)$
$m' \leftarrow \mathsf{Dec}(x, c)$
return $m = m'$

The idea underlying all three is that the adversary may pick any two plaintexts and submit them to a challenge oracle, receiving an encryption of one of the two in return. He must then decide which of the two messages the encryption represents. The difference between the three notions lies in when the adversary may make decryption queries:

Game	Before challenge call	After challenge call
IND-CPA	no	no
IND-CCA1	yes	no
IND-CCA2	yes	yes (*)

(*) The adversary may not ask for a decryption of the ciphertext he got from the challenge call.

Algorithm 9 Security of Public-Key Encryption

Adversary: $\mathcal{A}$

$(x, y) \leftarrow \mathsf{Gen}$
$\mathcal{A} \leftarrow y$
if IND-CCA1 or IND-CCA2 **then**
  while $c \leftarrow \mathcal{A}$ **do** $\mathcal{A} \leftarrow \mathsf{Dec}(c)$ **end while**
end if
$(m_0, m_1) \leftarrow \mathcal{A}$
$b \xleftarrow{R} \{0, 1\}$
$c^* \leftarrow \mathsf{Enc}(y, m_b)$
$\mathcal{A} \leftarrow c^*$
if IND-CCA2 **then**
  while $c \leftarrow \mathcal{A}$ **do**
    if $c = c^*$ **then** $\mathcal{A} \leftarrow \perp$ **else** $\mathcal{A} \leftarrow \mathsf{Dec}(c)$ **end if**
  end while
end if
$b' \leftarrow \mathcal{A}$
return $b = b'$

Security in any of these three models implies a probabilistic encryption function (and sufficient entropy), otherwise the adversary could just recompute encryptions of the two messages he sent to the challenge oracle himself and compare these with the ciphertext he received from it.

E.2 Homomorphic Encryption

A homomorphic public-key encryption scheme consists of four algorithms

$$(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$$

such that the message space is a group (denote the group operation by $\oplus$), the first three algorithms are a public-key encryption scheme and the fourth takes two ciphertexts and produces a ciphertext for the message corresponding to the group operation evaluated on the original two plaintexts, without access to the secret key. More formally, for any messages m_0, m_1 the additional correctness property is satisfied with probability 1.

Algorithm 10 Correctness of Homomorphic Encryption

Parameters: m_0, m_1
  $(x, y) \leftarrow \mathsf{Gen}$
  $c_0 \leftarrow \mathsf{Enc}(y, m_0)$
  $c_1 \leftarrow \mathsf{Enc}(y, m_1)$
  $c \leftarrow \mathsf{Add}(c_0, c_1)$
  $m' \leftarrow \mathsf{Dec}(x, c)$
  return $m' = m_0 \oplus m_1$

A homomorphic encryption scheme can never be IND-CCA2 secure. If the adversary chooses m_0, m_1 as his two challenge messages and gets a ciphertext c as a result, in a homomorphic scheme he can always pick any m' that is not the unit of the group, create $c' \leftarrow \mathsf{Enc}(y, m')$ and $c'' \leftarrow \mathsf{Add}(c, c')$. He can then send c'' to the decryption oracle (as the underlying plaintext differs from that of c, so must the ciphertext) to get m'' back and check if $m'' = m_0 + m'$ or $m'' = m_1 + m'$ holds.

E.3 Threshold Decryption

A public-key encryption scheme with *threshold decryption* for t out of n shares consists of algorithms

$$(\mathsf{Setup}, \mathsf{GenShare}, \mathsf{CombineKey}, \mathsf{Enc}, \mathsf{DecShare}, \mathsf{Combine})$$

for a given threshold t out of a number n of shares.

Setup The setup algorithm takes a security parameter as input and creates public parameters *params* which are implicitly available to all further algorithms.

GenShare This generates a pair (x, y) consisting of a secret key share and a public key share.

CombineKey This algorithm takes a list of n public key shares $(y_i)_{i=1}^n$ as input and returns a public key y.

Enc The encryption algorithm is identical to normal public-key encryption.

DecShare The shared decryption algorithm takes a private key share x_i and a ciphertext c as input and produces a partial decryption d_i.

Combine This takes a ciphertext c and a list of t partial decryptions and produces a plaintext.

The correctness game must return 1 with probability 1 for any message m and any set T of exactly t indices to use for decryption.

Algorithm 11 Correctness of Threshold Public-Key Encryption

Parameters: m, $T \subseteq \{1, \ldots, n\}$ with $|T| = t$

$params \leftarrow \mathsf{Setup}$
for $i \leftarrow 1 \ldots n$ **do** $(x_i, y_i) \leftarrow \mathsf{GenShare}$ **end for**
$y \leftarrow \mathsf{combineKey}((y_i)_{i=1}^n)$
$c \leftarrow \mathsf{Enc}(y, m)$
for all $i \in T$ **do** $d_i \leftarrow \mathsf{DecShare}(x_i, c)$ **end for**
$m' \leftarrow \mathsf{Combine}(c, (d_i, i)_{i \in T})$
return $m = m'$

For security, we allow the adversary to obtain any $t-1$ decryption key shares and give him partial decryption oracles for the others. The security notions are IND-TCPA, IND-TCCA1 and IND-TCCA2 where we demand that no efficient adversary can gain a non-negligible advantage in the game.

Definition 10 (Threshold Homomorphic Embedding). *A threshold scheme*

$$(\mathsf{Setup}, \mathsf{GenShare}, \mathsf{CombineKey}, \mathsf{Enc}, \mathsf{DecShare}, \mathsf{Combine})$$

has threshold homomorphic embedding

$$(\mathsf{ESetup}, \mathsf{EGenShare}, \mathsf{ECombineKey}, \mathsf{EEnc}, \mathsf{EDecShare}, \mathsf{ECombine}, \mathsf{EAdd})$$

if there are algorithms

$$(\mathsf{ExtractShare}, \mathsf{ExtractKey}, \mathsf{Extract}, \mathsf{EDecShare}, \mathsf{ECombine}, \mathsf{EAdd})$$

such that

$$\mathsf{EGenShare} = \mathsf{ExtractShare} \circ \mathsf{GenShare}$$

$$\mathsf{ECombineKey} = \mathsf{ExtractKey} \circ \mathsf{CombineKey}$$

$$\mathsf{EEnc} = \mathsf{Extract} \circ \mathsf{Enc}$$

$$\mathsf{Dec} = \mathsf{EDec} \circ \mathsf{Extract}$$

Algorithm 12 Security of Threshold Public-Key Encryption

Adversary: $\mathcal{A}$

$params \leftarrow \mathsf{Setup}$
$\mathcal{A} \leftarrow params$
for $i \leftarrow 1 \ldots n$ **do** $(x_i, y_i) \leftarrow \mathsf{GenShare}$ **end for**
$y \leftarrow \mathsf{CombineKey}((y_i)_{i=1}^n)$
$\mathcal{A} \leftarrow (y, (y_i)_{i=1}^n)$
for $i \leftarrow 1 \ldots t$ **do**
  $\tau \leftarrow \mathcal{A}$
  $\mathcal{A} \leftarrow x_\tau$
end for
if IND-TCCA1 or IND-TCCA2 **then**
  while $(c, \tau) \leftarrow \mathcal{A}$ **do** $\mathcal{A} \leftarrow \mathsf{DecShare}(x_\tau, c)$ **end while**
end if
$(m_0, m_1) \leftarrow \mathcal{A}$
$b \xleftarrow{R} \{0, 1\}$
$c^* \leftarrow \mathsf{Enc}(y, m_b)$
$\mathcal{A} \leftarrow c^*$
if IND-TCCA2 **then**
  while $(c, \tau) \leftarrow \mathcal{A}$ **do**
    if $c = c^*$ **then** $\mathcal{A} \leftarrow \perp$ **else** $\mathcal{A} \leftarrow \mathsf{DecShare}(x_\tau, c)$ **end if**
  end while
end if
$b' \leftarrow \mathcal{A}$
return $b = b'$

E.4 Homomorphic Threshold ElGamal

Given a cyclic group G with generator g, the following is the threshold homomorphic ElGamal encryption scheme.

Setup Create a cyclic group G of prime order q (where q is approximately the size of the security parameter) and a generator g.

GenShare Pick $x \xleftarrow{R} \mathbb{Z}_q$ and compute $y = g^x$ in G.

CombineKey Given a list of public key shares $(y_i)_{i \in I}$, compute $y = \prod_{i \in I} y_i$.

Enc Given a message m, pick a random $r \xleftarrow{R} \mathbb{Z}_q$ and compute $a = g^r$ and $b = g^m y^r$. The ciphertext is $c = (a, b)$.

DecShare Given $c = (a, b)$ and x, compute $d = a^x$.

Combine Given $c = (a, b), (x_i)_{i \in I}$ compute $h = b/(\prod_{i \in I} x_i)$ which will be g^m if everything was done correctly. Use a discrete-logarithm finding algorithm (for small m) to extract m.

Add Given $a_1 = g^{r_1}, b_1 = g^{m_1} y^{r_1}$ and $a_2 = g^{r_2}, b_2 = g^{m_2} y^{r_2}$, compute $a = a_1 \cdot a_2, b = b_1 \cdot b_2$.