How not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios

David Bernhard[1], Olivier Pereira[2], and Bogdan Warinschi[1]

[1] University of Bristol, {csxdb,csxbw}@bristol.ac.uk
[2] Université Catholique de Louvain, olivier.pereira@uclouvain.be

Abstract. The Fiat-Shamir transformation is the most efficient construction of non-interactive zero-knowledge proofs.
This paper is concerned with two variants of the transformation that appear but have not been clearly delineated in existing literature. Both variants start with the prover making a commitment. The strong variant then hashes both the commitment and the statement to be proved, whereas the weak variant hashes only the commitment. This minor change yields dramatically different security guarantees: in situations where malicious provers can select their statements adaptively, the weak Fiat-Shamir transformation yields unsound/unextractable proofs. Yet such settings naturally occur in systems when zero-knowledge proofs are used to enforce honest behavior. We illustrate this point by showing that the use of the weak Fiat-Shamir transformation in the Helios cryptographic voting system leads to several possible security breaches: for some standard types of elections, under plausible circumstances, malicious parties can cause the tallying procedure to run indefinitely and even tamper with the result of the election.
On the positive side, we define a form of adaptive security for zero-knowledge proofs in the random oracle model (essentially simulation-sound extractability), and show that a variant which we call *strong Fiat-Shamir* yields secure non-interactive proofs.
This level of security was assumed in previous works on Helios and our results are then necessary for these analyses to be valid. Additionally, we show that strong proofs in Helios achieve non-malleable encryption and satisfy ballot privacy, improving on previous results that required CCA security.

1 Introduction

Zero-knowledge proofs of knowledge allow a prover to convince a verifier that she holds information satisfying some desirable properties without revealing anything else. To be useful, such proof systems should satisfy completeness (the prover can convince the verifier that a true statement is indeed true) and soundness (the prover cannot convince the verifier that a false statement is true). Zero-knowledge proofs can either be interactive or non-interactive; for the latter the prover only sends his proof and the verifier decides to accept or reject the statement without any further interaction.

The focus of this paper is on the most common and efficient construction of non-interactive proofs, namely the Fiat-Shamir heuristic [1]. Here, one begins with an interactive sigma protocol, a special type of three-move protocol in which the prover sends a commitment, the verifier answers with a random challenge and the prover completes the protocol with a response. The idea behind the transformation is simple and appealing: have the prover compute the message of the verifier as the hash of the message sent by the prover — if the hash is modelled as a random oracle the message computed this way should look random as in an interactive execution, hence the properties of the original proof system should somehow be preserved.

The transformation appears in the literature in two different forms, depending on what is hashed. In the formalization of Bellare and Rogaway [2], which we refer to as the weak Fiat-Shamir transformation (wFS), the hash takes only the prover's first message as input. Other papers e.g. [3, 4] suggest including the statement to be proved in the hash input. In the remainder of the paper we call this the strong Fiat-Shamir transformation (sFS).

Contributions. The contributions of this paper fall into two main categories. First we identify weaknesses of the weak (sic!) Fiat-Shamir transformation and show that in applications it can be a serious source of insecurity. Secondly, we provide several positive results regarding the strong Fiat-Shamir transformation and its uses in applications.

Insecurity of wFS *and attacks on Helios.* Our first results show that the security proofs commonly given for Fiat-Shamir proofs do not hold when applied to weak proofs and when the prover can chose his statement(s) to prove adaptively. This may or may not render a protocol using them insecure, as a protocol may have other means of dealing with adaptivity. For example, in the original application to identification protocols, weak proofs are sufficient.

As an example where weak proofs do not yield security, we consider Helios [5, 6], a cryptographic voting protocol that has been used in practice. Versions of Helios have been employed, for example, for the election of the president of the Université catholique de Louvain [6], the Princeton University Undergraduate Student Government [7] and the board of the IACR [8]. We focus on the zero-knowledge protocols implemented since Helios 2.0 [6] for elections based on homomorphic tallying, which are still used in the latest version of Helios as documented on [9] at the time of writing. In brief, those elections work as follows. Trustees first jointly generate an election public key, using NIZK proofs to make sure that this key actually includes contributions from all trustees. Then, to cast a ballot, a voter encrypts a vote and attaches NIZK proofs that the vote is legal. All ballots are placed on a publicly readable bulletin board. Eventually, the election administrators homomorphically add all ballots, decrypt the result and use NIZK to prove the correctness of their actions. The encryption scheme is exponential ElGamal and the particular NIZKs involved are obtained by applying the weak Fiat-Shamir transformation to the Schnorr [10], Chaum-Pedersen [11] and disjunctive Chaum-Pedersen protocols (and variants thereof). These proofs are used to guarantee that the privacy of the votes rests on all trustees, to enforce that voters create ballots containing valid votes and to prevent dishonest administrators from claiming a wrong result. We show that the use of the wFS transformation is the source of three types of insecurity:

a) breaking verifiability by allowing colluding administrators to cast a single ballot that is not well-formed and contains any chosen number of votes for a specific candidate,
b) breaking liveness of the system by allowing colluding administrators to fail providing the election outcome while proving that they behave honestly, or by allowing voters to cast a random vote which leads to tallying taking superpolynomial time, and
c) breaking privacy by allowing the casting of related ballots that do not contain mere copies of previously submitted ciphertexts.

The first two of these attacks are undetectable under normal circumstances.

While our focus is on Helios which is our motivating application, in the full version of our paper we also show attacks against schemes constructed via the Naor-Yung paradigm and via the encrypt-then-prove construction: when using proofs derived through wFS these constructions may yield malleable encryption schemes.

Security of Strong Fiat-Shamir and Applications. The problems that we have identified in the use of the wFS do not apply to proofs obtained through the strong version of the transformation. It is then natural to ask what level of security does one get from these proofs. We provide several results. First, we formulate a security notion for non-interactive zero-knowledge proofs of knowledge which captures adversaries that can choose their statements adaptively. In essence, this notion is the analogue of simulation-sound extractability defined by Groth in the common reference string model [12]. Informally, a malicious prover is allowed to see simulated proofs (of potentially fake statements) and aims to provide valid looking proofs for adaptively chosen statements in such a way that an extractor cannot obtain witnesses. Interestingly, our definition is not simply a rehashing of the notion in [12]. In the random-oracle model, extraction requires the rewinding of the prover (as opposed to merely using a trapdoor) and in turn, this implies complex interaction between the adversary, the simulator and the extractor. We then show that applying sFS to Σ-protocols results in protocols that are simulation-sound extractable. Our result seems to be the first thorough investigation on the precise security guarantees offered by such proofs.

As a first application of this result, we investigate the security of non-malleable encryption schemes that are built by combining an IND-CPA encryption scheme with a proof of knowledge of the randomness used in the encryption process. We refer to this construction as the Enc+PoK approach. A well-known instantiation is the TDH0 scheme introduced and studied by Shoup and Gennaro [13]. Intuitively the construction should achieve IND-CCA security but so far, all attempts have failed to confirm or disprove this under natural assumptions (e.g., DDH in the random oracle model) [14, 15]. As a consequence, the form of non-malleability ensured by Enc+PoK schemes is, surprisingly, still unknown. We provide a lower-bound on the answer to this question: if the proof of knowledge used in the encryption process is simulation-sound extractable, then the resulting scheme is NM-CPA secure. An immediate corollary is that the TDH0 scheme is NM-CPA secure in the random oracle model under the DDH assumption.

We then turn to the analysis of ballot privacy in Helios. Prior work shows that ballot privacy is guaranteed if the encryption scheme used in the construction is IND-CCA [16, 17]. Since ballots in Helios use the Enc+PoK paradigm (which, as discussed above, is not known to be IND-CCA) a natural suggestion is then to replace it with something stronger. For example, Bernhard et al. suggested applying the Naor-Yung transformation to the underlying ElGamal encryption [17], while Bulens et al. used a variant of the TDH2 scheme [18]. These modifications both substantially increase the computational costs of the system and require major changes in the implementation.

Our final result is to show that although the NM-CPA notion is strictly weaker than IND-CCA [19], it is sufficient to ensure ballot privacy. In particular a minor tweak of the Enc+PoK construction currently used in Helios where we replace wFS with its strong counterpart and check for repeated ciphertexts is sufficient. The change that we require is easily accomplished by including additional elements in the inputs of the hash function and preserves the current level of efficiency.

2 The Fiat-Shamir/Blum Transformation

In this section we introduce the two variants of the Fiat-Shamir heuristic that we analyze. We start by fixing notation and recalling some standard notions. In the following we let $R \subseteq \mathcal{P}(\{0,1\}^* \times \{0,1\}^*)$ be an efficiently computable relation. R defines a language $\mathcal{L}_R = \{Y \in \{0,1\}^* | \exists w : R(w,Y)\}$ in NP. We further assume that there is a well-defined set $\Lambda \supseteq \mathcal{L}$ decidable in polynomial time.[3]

A non-interactive proof system for language $\mathcal{L}_R$ is a pair algorithms (Prove, Verify). Such a proof system is complete for $\mathcal{L}_R$ if for every $(w,Y) \in R$, with overwhelming probability if $\pi \leftarrow \mathsf{Prove}(w,Y)$ then $\mathsf{Verify}(Y,\pi) = 1$. We define soundness of such proof systems (the property that a cheating prover cannot make the verifier accept a false statement) later in the paper. Here we recall the notion of zero-knowledge in the random oracle model [20].

In this setting, a simulator $\mathcal{S}$ for a proof system is an algorithm in charge of answering random oracle queries and producing valid proofs for any statement $Y \in \Lambda$ with respect to this oracle. In particular, it can "patch" the oracle to create its simulated proofs. Such a simulator responds to the following queries:

[3] Suppose that $\mathcal{L}$ is the set of DDH triples (G^a, G^b, G^{ab}) over some group $\mathbb{G}$. Then Λ could be $\mathbb{G}^3$. The reason for defining this formally is that we will later expect our zero-knowledge simulator to produce valid "proofs" for some "false statements", but which ones? Can it produce a proof for the statement consisting of the empty string, for example? We use Λ as the class of statements on which the simulator can produce proofs.

$\mathcal{H}(s)$ $\mathcal{S}$ maintains a list of oracle query/response pairs. For repeated queries, $\mathcal{S}$ answers consistently; for fresh queries, $\mathcal{S}$ draws a random value r, adds (s, r) to its list and returns r.

$\mathsf{Simulate}(Y)$ For $Y \in \Lambda$, the simulator returns a proof π such that $\mathsf{Verify}(Y, \pi) = 1$ if the verifier uses the simulator for its oracle queries. $\mathcal{S}$ can add query/response pairs to its oracle list to process a simulation query.

Definition 1 (Zero-Knowledge). *A proof system is zero-knowledge if there is a simulator $\mathcal{S}$ such that no adversary who can make queries to the random oracle and queries of the form* $\mathsf{create\text{-}proof}(w, Y)$ *can distinguish the following two settings with non-negligibly better than* $1/2$ *probability.*

1. Random oracle queries are answered by a random oracle. In response to $\mathsf{create\text{-}proof}(w, Y)$, the challenger checks that $R(w, Y)$. If not, he returns $\perp$. Otherwise, he returns $\mathsf{Prove}(w, Y)$.
2. The challenger runs a copy of the simulator $\mathcal{S}$. It forwards random oracle queries to $\mathcal{S}$ directly. For $\mathsf{create\text{-}proof}(w, Y)$, the challenger checks if $R(w, Y)$ holds: if not, the challenger returns $\perp$; if it holds, the challenger sends $\mathsf{Simulate}(Y)$ to $\mathcal{S}$ and returns the result to the adversary.

Sigma Protocols. A sigma protocol for a language $\mathcal{L}_R$ is a protocol for two parties, a prover and a verifier. Both share a statement $Y \in \mathcal{L}_R$ as input and the prover may additionally hold a witness w.

The prover begins by sending a value A known as the commitment. The verifier replies with a challenge c drawn uniformly from a fixed challenge set. The prover finishes the protocol with a response f whereupon the verifier applies a deterministic algorithm Verify to Y, A, c and f which can accept or reject this execution.

A sigma protocol is correct (w.r.t. $\mathcal{L}_R$) if the prover, on input a pair (w, Y) satisfying R and interacting with the verifier who has input Y, gets the verifier to accept with probability 1.

A sigma protocol has *special honest verifier zero knowledge* if there is an algorithm $\mathsf{Simulate}$ that takes as input a statement $Y \in \Lambda$, challenge c and response f and outputs a commitment A such that $\mathsf{Verify}(Y, A, c, f) = 1$ and furthermore, if c and f where chosen uniformly at random from their respective domains then the triple (A, c, f) is distributed identically to that of an execution between the prover and the verifier. Notice that the verifier is supposed to work with statements that may be false.

A sigma protocol has *special soundness* if there is an algorithm $\mathsf{Extract}$ that takes as input a statement Y and any two triples (A, c, f) and (A', c', f') such that both verify w.r.t. Y, $A = A'$ and $c \neq c'$, and returns a witness w such that $R(w, Y)$.

The Fiat-Shamir Transformation. The Fiat-Shamir transformation [1] (which [2] attributes to Blum) is a technique to make sigma protocols non-interactive using a cryptographic hash function. There are two commonly used descriptions of this technique that we call weak and strong Fiat-Shamir and which we describe together in the following definition.

Definition 2 (Fiat-Shamir Transformation). *Let $\Sigma = (\mathsf{Prove}_\Sigma, \mathsf{Verify}_\Sigma)$ be a sigma protocol and $\mathcal{H}$ a hash function. The weak Fiat-Shamir transformation of Σ is the proof system $\mathsf{wFS}_\mathcal{H}(\Sigma) = (\mathsf{Prove}, \mathsf{Verify})$ defined as follows:*

$\mathsf{Prove}(w, Y)$ *Run $\mathsf{Prove}_\Sigma(w, Y)$ to obtain commitment A. Compute $c \leftarrow \mathcal{H}(A)$. Complete the run of Prover_Σ with c as input to get the response f. Output the pair (c, f).*

$\mathsf{Verify}(Y, c, f)$ *Compute A from (Y, c, f), then run $\mathsf{Verify}_\Sigma(Y, A, c, f)$.*

The strong Fiat-Shamir transformation of Σ, i.e., $\mathsf{sFS}(\Sigma) = (\mathsf{Prove}, \mathsf{Verify})$ is obtained as above with the difference that c is computed by $c \leftarrow \mathcal{H}(Y, A)$.

3 Pitfalls of the Weak Fiat-Shamir Transformation

We now describe various standard protocols in which the use of the weak Fiat-Shamir transformation can have undesirable effects. We illustrate these effects through several new practical attacks on various components of the Helios voting system, which relies on these protocols.

Schnorr Proofs. The Schnorr [10] signature scheme is the weak Fiat-Shamir transformation of the Schnorr identification protocol. In a group $\mathbb{G}$ of order q generated by G, it proves knowledge of an exponent x satisfying the equation $X = G^x$ for a known X. Viewing (x, X) as a signing/verification key pair and including a message in the hash input yields a signature of knowledge.

To create a proof, the prover picks a random $a \leftarrow \mathbb{Z}_q$ and computes $A = G^a$. He then hashes A to create a challenge $c = \mathcal{H}(A)$. Finally he computes $f = a + cx$; the proof is the pair (c, f) and the verification procedure consists in checking the equation $c \overset{?}{=} \mathcal{H}(\frac{G^f}{X^c})$.

The weak Fiat-Shamir transformation can safely be used here, as discussed in previous analysis [10, 21], since the public key X is selected first and given as input to the adversary who tries to produce a forgery.

However, if the goal of the adversary is to build a valid triple (X, c, f) for any X of his choice, then this protocol is not a proof of knowledge anymore unless the discrete logarithm problem is easy in $\mathbb{G}$. Suppose indeed that there is an extractor $\mathcal{K}$ that, by interacting with any prover $\mathcal{P}$ that provides a valid triple (X, c, f), extracts $x = \log_G(X)$. This extractor can be used to solve an instance Y of the discrete logarithm problem with respect to $(\mathbb{G}, G)$ as follows: use Y as the proof commitment, compute $c = \mathcal{H}(Y)$, choose $f \leftarrow \mathbb{Z}_q$ and set $X = (\frac{G^f}{Y})^{\frac{1}{c}}$. Since the proof (Y, c, f) passes the verification procedure for statement X, the extractor $\mathcal{K}$ should be able to compute $x = \log_G(X)$ by interacting with our prover. We now observe that, by taking the discrete logarithm in base G on both sides of the definition of X, we obtain the solution $\log_G(Y) = f - cx$ to the discrete logarithm challenge.

Application to Helios. Schnorr proofs are used during the key generation procedure of Helios as a way to prevent trustees from choosing their public key as a function of the public key of the other trustees, which could give them the possibility to select the election private key at will and to decrypt all individual votes [22]. While the scenario above shows that trustees who publish a public key together with a Schnorr proof for that public key do not necessarily know the corresponding private key, the fact that our scenario does not allow the prover to choose his statement (but just to compute it as a function of the elements of the proof) does not seem to give rise to any practical attack. These weak Schnorr proofs would, however, break the proof of ballot privacy that we give later in this paper (assuming strong proofs).

Chaum-Pedersen Proofs. Chaum and Pedersen [11] introduced a proof of discrete logarithm equality, which they make non-interactive using the strong form of the Fiat-Shamir transformation. More precisely, given two group elements (G, X), a prover who knows the discrete logarithm $x = \log_G(X)$ can prove that two group elements (R, S) satisfy the relation $\log_G(X) = \log_R(S)$ as follows. He picks a random $a \leftarrow \mathbb{Z}_q$, computes $A = G^a$, $B = R^a$, $c = \mathcal{H}(R, S, A, B)$ and $f = a + cx$. The proof is the pair (c, f) and the verification procedure consists in checking the equation $c \overset{?}{=} \mathcal{H}(R, S, \frac{G^f}{X^c}, \frac{R^f}{S^c})$.

We observe that this proof is not sound anymore if it is used as a proof that three elements (X, R, S) are such that $\log_G(X) = \log_R(S)$, that is, if the prover also has the possibility to choose X in the process of building his proof. Indeed, a prover could select $(a, b, r, s) \leftarrow \mathbb{Z}_q^4$ at random, compute $A = G^a$, $B = G^b$, $R = G^r$ and $S = G^s$ from which he can compute $c = \mathcal{H}(R, S, A, B)$ and $f = \frac{b+cs}{r}$. He now completes the proof by computing $x = (f - a)/c$ and setting $X = G^x$. Now, we observe that $\log_G(X) = \frac{s}{r} + \frac{b-ar}{rc}$ while $\log_R(S) = \frac{s}{r}$, which differ with overwhelming probability.

Application to Helios. Chaum-Pedersen proofs instantiated with the weak Fiat-Shamir transformation (that is, $c = \mathcal{H}(A, B)$) are used during the ElGamal decryption procedure of Helios, in order to demonstrate that the decryption of the product of the votes that is computed by the trustees is consistent with the public key. More precisely, given a public key X and a ciphertext (R, S) that encrypts the sum of all votes, a trustee is required to compute $T = R^x$ where $x = \log_G(X)$ and to publish it together with a Chaum-Pedersen proof that $\log_G(X) = \log_R(S)$. The ElGamal decryption is then computed as $\log_G(S/T)$.

In this proof, a malicious trustee does not have the possibility to choose his private key at decryption time, but has the possibility to select T as part of the proof computation process. He can do so as follows. Select $(a, b) \leftarrow \mathbb{Z}_q^2$ at random, compute the proof commitments $A = G^a$ and $B = G^b$, the challenge $c = \mathcal{H}(A, B)$ and the response $f = a + cx$. Eventually, compute the decryption factor $T = (\frac{R^f}{B})^{\frac{1}{c}}$. It is easy to verify that the proof (c, f) is valid for the tuple (G, X, R, S), but that $\log_R(T) = x + \frac{ar-b}{c}$, which will be different from x with overwhelming probability. As a result, the decryption procedure will provide an aberrant result: an essentially random element of $\mathbb{Z}_q$. This strategy provides a way to build a denial of service attack against a Helios election, without anyone being able to detect who was responsible.

A more dangerous attack can be mounted if we assume that the trustees have the possibility to passively eavesdrop on the randomness of all voters. Though demanding, such an attack is still easier to mount and harder to detect than a full active attack. We would expect the impact of such a scenario to be "only" a complete loss of privacy by the voters but we show that it actually provides a way for the trustees to announce any election outcome of their choice as soon as they can actively corrupt a single voter (which can happen simply if a trustee is a voter himself).

Consider an election with trustees who would like to announce the election outcome m. These trustees select a private key x and publish the public key X. They also select $(a, b) \leftarrow \mathbb{Z}_q^2$, compute $A = G^a$, $B = G^b$, $c = H(A, B)$ and $f = a + cx$. Then all the voters submit their votes, except the corrupted one who waits until the last minute of the election. At that time, the trustees compute the product of all encrypted votes that have been submitted and obtain a ciphertext $(R', S') = (G^{r'}, G^{m'} \cdot G^{xr'})$ for some values r' and m' that they can compute using the randomness of the voters. They now compute $r = \frac{b+c(m'-m)}{f-cx}$, as well as a ciphertext $(G^{r-r'}, G^{x(r-r')})$ which is an encryption of 0 for which they can compute a proof of validity since they know $r - r'$. This ciphertext and proof are submitted by the corrupted voter, with the effect that the product of all encrypted votes is $(R, S) = (G^r, G^{m'} \cdot G^{xr})$. It can now be verified that (c, f) form a valid proof that $\log_G(X) = \log_R(\frac{S}{G^m})$, which indicates that m is the outcome of the election.

Disjunctive Chaum-Pedersen Proofs. Disjunctive proofs allow proving that one of two statements holds without revealing which one is correct. These proofs have numerous applications. For instance, they can be used by a voter to demonstrate that a ciphertext he produced is an encryption of either 0 or 1 (but nothing else), expressing whether or not he supports a candidate.

Suppose that a voter builds an exponential ElGamal ciphertext (R, S) with respect to public key X and wants to prove that it encrypts 0 or 1. We consider the case where it is an encryption of 1 (the other case is similar). First, the voter simulates a proof that $\log_R(S) = x$ by selecting a random proof $(c_0, f_0) \leftarrow \mathbb{Z}_q^2$ and computing $A_0 = G^{f_0}/R^{c_0}$ and $B_0 = X^{f_0}/S^{c_0}$. Then he selects $a_1 \leftarrow \mathbb{Z}_q$, computes $A_1 = G^{a_1}$, $B_1 = X^{a_1}$, $c = \mathcal{H}(A_0, B_0, A_1, B_1)$, $c_1 = c - c_0$ and $f_1 = a_1 + c_1 r$. The proof consists of (c_0, c_1, f_0, f_1) and verification consists of verifying whether $c_0 + c_1 = \mathcal{H}(\frac{G^{f_0}}{R^{c_0}}, \frac{X^{f_0}}{S^{c_0}}, \frac{G^{f_1}}{R^{c_1}}, \frac{X^{f_1}}{(S/G)^{c_1}})$.

Application to Helios. The proof we just described is exactly the one used in Helios to guarantee that voters encode at most one vote for each candidate and it exhibits weaknesses that are similar to those described above, but with an even more dangerous effect. Consider an election organized by corrupted trustees who would like to influence the election outcome by adding (or removing) m approvals to a candidate of their choice. These trustees now have the freedom to choose any public key and ciphertext of their choice that would allow them to compute an encryption of m and to prove that it is an encryption of 0 or 1. They can achieve this as follows.

They first select $(a_0, b_0, a_1, b_1) \leftarrow \mathbb{Z}_q^4$, from which they compute the commitments $A_0 = G^{a_0}$, $B_0 = G^{b_0}$, $A_1 = G^{a_1}$, $B_1 = G^{b_1}$, the challenge $c = \mathcal{H}(A_0, B_0, A_1, B_1)$ and the private key $x = \frac{(b_0+cm)(1-m)-b_1 m}{a_0(1-m)-a_1 m}$ and the public key $X = G^x$. Using this public key, they select a random encryption of m by selecting $r \leftarrow \mathbb{Z}_q$ and computing $(R, S) = (G^r, G^m X^r)$. Eventually, they compute the challenges $c_1 = \frac{b_1 - a_1 x}{1-m}$ and $c_0 = c - c_1$ and the responses $f_0 = a_0 + c_0 r$ and $f_1 = a_1 + c_1 r$. It can be verified that (c_0, c_1, f_0, f_1) form a proof that (R, S) encrypt 0 or 1, while it actually encrypts an arbitrary m.

Furthermore, it can be observed that this proof, like the others that we presented above, is indistinguishable from a regular one. We can indeed define this ciphertext and proof through 11 parameters: $(x, r, s, a_0, a_1, b_0, b_1, c_0, c_1, f_0, f_1)$ which, in order to provide a valid proof and encrypt m, must satisfy the 6 equations below, leaving 5 degrees of freedom.

$$\begin{array}{rl} s = rx + m & f_1 = a_1 + rc_1 \\ c_0 + c_1 = H(A_0, B_0, A_1, B_1) & f_0 x = b_0 + c_0 s \\ f_0 = a_0 + rc_0 & f_1 x = b_1 + c_1(s-1) \end{array}$$

A simulator can produce 11 parameters from the exact same distribution as follows. Given random x and r, compute $s = rx + m$, then pick $(c_0, c_1, f_0, f_1) \leftarrow \mathbb{Z}_q^4$ at random, compute the unique values a_0, b_0, a_1, b_1 that satisfy the last four equations above, and patch the random oracle to make sure that $\mathcal{H}(A_0, B_0, A_1, B_1) = c_0 + c_1$. In order to see that these parameters are distributed like those from the malicious proof, we observe that they satisfy the same equations and are generated from the same number of degrees of freedom except that the simulator has the benefit of one extra degree coming from its possibility to patch the random oracle. Now, we also observe that this simulator proceeds exactly as the simulator of a normal disjunctive Chaum-Pedersen proof, which shows that our malicious proof is indistinguishable from a regular one: they are both indistinguishable from the simulated one. Since the simulated proofs can be computed from the ciphertexts only, and since an encryption of m is computationally indistinguishable from an encryption of 0 or 1, our malicious ciphertext and proof are indistinguishable from regular ones.

Other attack possibilities exist, based on the same techniques. For instance, a voter who does not know the election private key can build a ciphertext that encrypts a random value in $\mathbb{Z}_q$ and prove that it encrypts 0 or 1, which would make the decryption procedure fail. We do not know however whether it is possible to build such a proof in a way that is indistinguishable from a regular one.[4]

Encrypt + PoK. Adding a proof of knowledge of the plaintext/randomness to a ciphertext in an IND-CPA secure public key encryption scheme is a common way to yield a non-malleable encryption scheme.[5] We formalise this construction and show that using wFS does *not* yield non-malleable encryption.

[4] Our current technique involves setting $c_0 = 0$. While such a ballot passes the current Helios verifier, this could be detected in an audit.

[5] Though the exact form of non-malleability that is provided is unclear [13].

Definition 3 (Encrypt+PoK). *Let $\mathfrak{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme. Let $R((m, r), (Y, pk)) := (Y = \mathsf{Enc}(pk, m; r))$ be the relation that Y is an encryption of m with randomness r for public key pk, let Λ be the ciphertext space (or some suitable superset thereof) and let $\mathfrak{P} = (\mathsf{Prove}, \mathsf{Verify})$ be a NIZK-PoK for this relation.*

The Encrypt+PoK transformation $\mathfrak{E}_{\mathfrak{P}}$ is the following encryption scheme.

KeyGen' Run KeyGen.

Enc'(pk, m) Draw some randomness r and create a ciphertext $E = \mathsf{Enc}(pk, m; r)$. Create a proof $\pi \leftarrow \mathsf{Prove}(pk, E, m, r)$. The ciphertext is the pair (E, π).

Dec'(sk, E, π) First run $\mathsf{Verify}(pk, E, \pi)$. If this fails (returns 0), output $\perp$ and halt. Otherwise, return $\mathsf{Dec}(sk, E)$.

Consider the ElGamal encryption scheme with weak Schnorr proofs of the randomness used for encryption (which would allow one to extract the message), which would be a weak variant of the TDH0 scheme [13]. In other words, a ciphertext for a message M under public key X is $(G^r, M \cdot X^r, c, f)$ where $c = \mathcal{H}(G^f/(G^r)^c)$. We can rerandomise such a ciphertext (R, S, c, f) by picking a random u and setting the new ciphertext to be $(R \cdot G^u, S \cdot X^u, c, f + cu)$. The new plaintext is the same as the old one as $S/R^x = M \cdot X^{r+u}/(G^{r+u})^x = M$ and the proof still verifies as $c = \mathcal{H}\left(\frac{G^{f+cu}}{(G^{r+u})^c}\right) = \mathcal{H}\left(\frac{G^f}{(G^r)^c}\right)$. Clearly, this encryption scheme is malleable.

Application to Helios. The same rerandomisation technique can be applied to current Helios ballots, giving a ballot privacy attack in the style of Cortier and Smyth [23] (based on the same principles as the attacks described in [24, 25].) Helios ballots contain ElGamal ciphertexts (R, S) with disjunctive Chaum-Pedersen proofs (c_0, c_1, f_0, f_1). To rerandomise such a ciphertext, pick a random u and set $R' = R \cdot G^u$, $S' = S \cdot Y^u$, $f'_0 = f_0 + c_0 u$ and $f'_1 = f_1 + c_1 u$. Unlike previously known rerandomisation techniques, this one does not make use of a repeated ElGamal ciphertext or proof. It can be detected however by checking for repeated hash values, just as for the previous attacks.

FURTHER EXAMPLES. The various attacks that we described above focus on applications to the Helios voting system, which uses the weak Fiat-Shamir transformation in all proofs. We believe that these examples provide clear evidence that the weak Fiat-Shamir transformation should not be used in that context: in particular, we showed that malicious authorities can arbitrarily influence the outcome of an election, which is in clear contradiction with the universal verifiability properties expected from that system. In the next sections, we will focus on the properties of the strong Fiat-Shamir transformation and show the benefits that its adoption would provide for the Helios system.

We stress that there are various other contexts in which the weak Fiat-Shamir transformation should not be used. For instance, similarly to our observation for the weak variant of the TDH0 scheme, the scheme resulting from the Naor-Yung transformation [26] applied to ElGamal encryption may become malleable if the weak Fiat-Shamir transformation is used, contradicting the level of desired security. We provide in Appendix A attacks against a concrete instantiation of that transformation.

4 Simulation Sound Extractable Proofs

The examples discussed in the previous section show that the wFS transform fails to offer even the most basic soundness properties in many contexts. We now investigate the soundness properties of the sFS transform. More precisely, we formulate the notion of *simulation sound extractable proofs* in the random oracle model and show its applications to the sFS transformation. Our definition draws inspiration from that of witness-extended emulation [4] in which the existence of an extractor is demanded such that for any adversary returning a vector of statements and proofs, the extractor returns identically distributed elements along with the witnesses to the proven statements. However, the definition is perhaps more appropriately viewed as the analogue definition of simulation sound extractability defined by Groth [12] which combines the simulation soundess approach of Sahai [27], with proofs of knowledge [28].

We consider a malicious prover who may ask to see simulated proofs (as in simulation-soundness). The extractor that we consider gets the transcript of a run of the prover where the prover outputs several valid proofs together with the transcipt of random oracle queries. His goal is to extract witnesses of these proofs. In the process, we allow the extractor to invoke and communicate with copies of the prover that use the same randomness as the run it is trying to extract from. This ability is what permits the knowledge extractor to fork the prover's execution without giving the extractor access to the coins of the prover.[6] A bit more precisely, a malicious prover for a proof system $\mathfrak{P} = (\mathsf{Prove}, \mathsf{Verify})$ is an algorithm $\mathcal{A}$ that expects access to two oracles: a hashing oracle and a simulation oracle. Thus $\mathcal{A}$ may submit some string s and expects $\mathcal{H}(s)$ in return and it may also make simulation calls $\mathsf{Simulate}(Y)$ for any statement $Y \in \Lambda$ and expects to obtain a proof π such that $\mathsf{Verify}(Y, \pi) = 1$. The prover returns a pair of vectors $(\boldsymbol{Y}, \boldsymbol{\pi})$.

Definition 4 (Simulation Sound Extractability). *Let $\mathfrak{P}$ be a zero-knowledge proof system with simulator $\mathcal{S}$. We say that $\mathfrak{P}$ is simulation sound extractable (SSE) if there exists an extractor $\mathcal{K}$ such that for every prover $\mathcal{A}$, $\mathcal{K}$ wins the following game with non-negligible probability.*

1. *(Initial run.)* The game selects a random string ω for $\mathcal{A}$. It runs an instance of $\mathcal{A}$ with the simulator $\mathcal{S}$ until $\mathcal{A}$ makes his output and halts. If $\mathcal{A}$ does not output any proofs, any of the proofs do not verify (w.r.t. the instance of $\mathcal{S}$ used as the random oracle) or any of $\mathcal{A}$'s statement/proof pairs (Y, π) is such that π was the result of a $\mathsf{Simulate}(Y)$ query, then $\mathcal{K}$ wins the game directly.

[6] Although not necessary for this paper, hiding the adversary's randomness from the extractor can be helpful in other contexts to prove separation results.

2. *(Extraction.)* The game runs an instance of $\mathcal{K}$, giving it the transcript of all queries in the initial run and the produced $(\boldsymbol{Y}, \boldsymbol{\pi})$ as input. $\mathcal{K}$ may repeatedly make one type of query invoke in response to which the game runs a new invocation of $\mathcal{A}$ on the same randomness ω that it chose for the initial run. All queries made by these instances are forwarded to $\mathcal{K}$ who can reply to them.
3. $\mathcal{K}$ wins the game if it can output a vector of witnesses $\boldsymbol{w}$ that match the statements $\boldsymbol{Y}$ of the initial run, i.e. for all i we have $R(w_i, Y_i)$.

The following theorem confirms that the strong Fiat-Shamir transformation yields proof systems that satisfy the notion we described above.

Theorem 1. *Let Σ be a sigma protocol with a challenge space that is exponentially large in the security parameter, special soundness and special honest verifier zero-knowledge. Then $\mathsf{sFS}(\Sigma)$ is zero-knowledge and simulation sound extractable with respect to expected polynomial-time adversaries.*

Applications. The Schnorr and Chaum-Pedersen protocols are clearly both sigma protocols with special soundness and special honest verifier zero knowledge so Theorem 1 applies and the sFS versions of these protocols are SSE proofs. For disjunctive Chaum-Pedersen, the challenge is the actual c obtained from the verifier and the response is the tuple $f = (f_0, f_1)$. This is a sigma protocol with special soundness and almost special honest verifier zero knowledge — almost, because in our definition the simulator chooses c, f independently and uniformly at random yet if $c \neq c_0 + c_1$ then the resulting proof will not verify, patched oracle or not. We could fix this by not sending c_1 in the response and having the verifier recompute $c_1 = c - c_0$. We will ignore this point as it is easy to see that, if the simulator chooses c, f at random and then adjusts c_0, all relevant theorems still hold. In particular, the sFS transformation of disjunctive Chaum-Pedersen is still a simulation-sound extractable proof.

Encrypt + PoK.

With this notion we can restore the folklore result that appending a PoK to an $\mathsf{IND\text{-}CPA}$ scheme gives a $\mathsf{NM\text{-}CPA}$ one, if the PoK is simulation-sound extractable. For space reasons the proof is in the appendices.

Theorem 2. *Let $\mathfrak{E}$ be an $\mathsf{IND\text{-}CPA}$ secure encryption scheme with respect to expected polynomial-time adversaries and $\mathfrak{P}$ be a simulation-sound extractable NIZK-PoK for the encryption relation. Then $\mathfrak{E}_{\mathfrak{P}}$ is non-malleable ($\mathsf{NM\text{-}CPA}$) secure.*

5 Ballot Privacy in Helios

In this section we propose a modification to Helios and prove that it satisfies ballot privacy in the model of single-pass voting of Bernhard et al. [17].

Single-Pass Schemes. A single-pass voting scheme is a protocol consisting of the following algorithms and execution protocol for a set $\mathcal{V}$ of voters, a set $\mathcal{T}$

of trustees and a bulletin board $\mathcal{B}$. The class of single-pass schemes includes not only Helios [5] but also several other cryptographic voting schemes [29–31]. Single-pass voting models two of the most popular approaches to cryptographic voting, homomorphic tallying and mix-nets. Voters need only read a single message off the board (the election specification and public keys) and post a single message (their ballot) in return. We assume some underlying voter authentication mechanism.[7]

$\mathsf{Setup}(1^\lambda)$ is an algorithm to create public parameters for the election and secret ones for all trustees.
Setup produces one public output Y known as the public key of the election and a secret output x_i for each trustee $\mathcal{T}_i$ in the set of trustees $\mathcal{T}$. The secret outputs of all trustees together are known as the secret key of the election.

$\mathsf{Vote}(id, v, Y)$ is a probabilistic algorithm run by voters to prepare their votes for submission. It takes as input a voter's identity id, a vote v and public information Y and outputs a ballot $s \leftarrow \mathsf{Vote}(id, v, Y)$.

$\mathsf{Validate}(b, s)$ models the algorithm run by the bulletin board during voting. Its inputs are the current board state b and the submitted ballot s. It returns 1 if the submission is deemed valid (given the current state of the board) and 0 otherwise.

$\mathsf{Tally}(b)$ is a tallying protocol that is run by the trustees. Its inputs are the board so far and the private data kept by the trustees from the setup phase.

$\mathsf{Result}(b)$ is a deterministic algorithm that takes a bulletin board b of a completed election and returns the result of the election, or a special symbol $\perp$ if the board does not contain a valid result.

A single-pass scheme is executed as follows.

1. *Setup phase.* The trustees run the Setup algorithm and post the public key Y to the bulletin board.
2. *Voting phase.* Each voter may proceed as follows or abstain. He reads public key Y off the board and computes a ballot $s \leftarrow \mathsf{Vote}(id, v, Y)$ where id is his identity and v is his vote, and submits s to the board.
   The board runs $\mathsf{Validate}(b, s)$ on every submission it receives and appends valid ones to its state.
3. *Tallying phase.* The trustees run the Tally protocol and may post to the board.

A single-pass protocol is correct w.r.t. a result function ρ if as long as everyone follows the protocol, with overwhelming probability (in the security parameter) none of the algorithms abort, Result returns a result when executed on the board at the end of the tallying phase and this result corresponds to ρ evaluated on the votes cast by the voters.[8]

[7] In the election of the president of UC Louvain [6] using Helios, authentication was handled by the university's existing infrastructure. As such it escapes cryptographic modelling and we choose not to model authentication (in particular we do not wish to assume a PKI).

[8] One may also want ρ to operate on (v, id) pairs: in the Helios election at UC Louvain, votes from students, faculty and staff were weighted differently.

Ballot Privacy. We base our definition of ballot privacy on previous work in this area by Bernhard et al. [17, 32]. Ballot privacy is defined by means of a cryptographic indistinguishability game. The new feature of our definition is that it can deal with dishonest trustees; we introduce a simulator to handle tallying in this case.

Definition 5 (Ballot Privacy). *A single-pass protocol for n trustees and any number of voters has* ballot privacy *against up to $m < n$ dishonest trustees if there is a simulator $\mathcal{S}$ such that for any efficient adversary $\mathcal{A}$, the advantage* $\mathbf{Pr}[\mathcal{A} \textit{ wins}] - 1/2$ *against the following indistinguishability game is negligible (as a function of the security parameter). The simulator $\mathcal{S}$ is given black-box access to the adversary $\mathcal{A}$ and may invoke further copies of $\mathcal{A}$ using the same randomness as was used in the main run in the security game. We assume static corruption of trustees: the sets of honest and dishonest trustees are fixed in advance. The adversary can adaptively choose voters to be honest or dishonest, however.*

Setup phase. The challenger picks a bit $\beta \leftarrow \{0, 1\}$ uniformly at random. He sets up two bulletin boards $\mathcal{L}$ and $\mathcal{R}$. The adversary is given access to either $\mathcal{L}$ if $\beta = 0$ or to $\mathcal{R}$ if $\beta = 1$.
The trustees jointly run the Setup protocol, the challenger playing the honest trustees and the adversary, the corrupt ones. This produces some output Y on the visible board. The challenger then copies Y from the visible board to the hidden one. If the setup phase fails to complete, the adversary loses the game.

Voting phase. The adversary may make two types of queries.

$\mathsf{Vote}(id, v_{\mathcal{L}}, v_{\mathcal{R}})$ queries. The adversary provides a voter identity id and two votes $(v_{\mathcal{L}}, v_{\mathcal{R}})$. The challenger runs $b_{\mathcal{L}} \leftarrow \mathsf{Vote}(id, v_{\mathcal{L}}, Y)$ and $b_{\mathcal{R}} \leftarrow \mathsf{Vote}(id, v_{\mathcal{R}}, Y)$, where Y is the public key of the election that can be computed using Keys on the public information on the board from the setup phase.
The ballots $b_{\mathcal{L}}$ and $b_{\mathcal{R}}$ are submitted to the corresponding boards which process them normally (run $\mathsf{Validate}$ and append the ballot if it passes validation).

$\mathsf{Ballot}(id, b)$ queries. These are queries made on behalf of corrupt voters. Here the adversary provides a ballot b. The challenger first submits b to the board visible to the adversary, which validates it and appends it if validation is successful. If the ballot successfully validates on the visible board, the challenger also submits the ballot to the invisible board which again validates the ballot and appends it if successful.

Tallying phase. If the adversary sees the $\mathcal{L}$ board ($\beta = 0$) then tallying can take place as normal. The trustees execute the Tally protocol, the challenger playing the honest ones and the adversary, the dishonest ones.
If the adversary sees the $\mathcal{R}$ board, the challenger starts up the simulator $\mathcal{S}$ and passes it both the $\mathcal{L}$ and $\mathcal{R}$ boards and the state of the honest trustees. In the random oracle model, the simulator is responsible for the random

oracle from this point onwards (and gets a list of all previously queried input/output pairs). The simulator acts on behalf of the honest trustees from now onwards and may post to the board.

At the end of the game the adversary may make a guess of β and wins if his guess is correct.

HELIOS AS A SINGLE-PASS SCHEME. We describe our proposed, fixed Helios protocol in the language of single-pass voting schemes. It is identical to the current version of Helios except for two changes:[9]

– All weak proofs are replaced with their strong counterparts, i.e. the statements (ciphertexts (R, S), public key Y and generator G) are hashed as well.[10]
– Ballots are rejected by the board if they contain repeated ciphertexts.

Setup We assume that a cyclic group $\mathbb{G}$ of order prime q with generator G is fixed and public. We also assume that the election parameters (names of candidates, election type i.e. approval or single-choice etc.) are fixed and public. Each of the n trustees $\mathcal{T}_i$ picks a random secret key $x_i \leftarrow \mathbb{Z}_q$ and computes a public key $Y_i = G^{x_i}$. $\mathcal{T}_i$ then computes a strong Schnorr proof of knowledge π_i of an x_i such that $G^{x_i} = Y_i$ and posts (Y_i, π_i) to the bulletin board. The actual election public key is $Y = \prod_{1 \leq i \leq n} Y_i$ which can be computed from the values on the board.

Vote The voter creates a ballot by encrypting each of his choices as an exponential ElGamal ciphertext, i.e., to encrypt a vote v he chooses a random r and computes $R = G^r, S = G^v \cdot Y^r$. He then creates a disjunctive Chaum-Pedersen proof $\pi = (c_0, c_1, f_0, f_1)$ that his ciphertext contains an encryption of 0 or 1. If the election specification contains further rules, for example that voters may pick at most one candidate in each question, the voter may have to complete further proofs. A more detailed description of Helios ballots and the process by which the disjunctive proofs are made is in Appendix B. In our proposed Helios, all the voters' proofs are *strong* Fiat-Shamir proofs.

Validate The bulletin board accepts a ballot if it is of the correct format, i.e. the number of ciphertexts and proofs match the election parameters, and all proofs verify. Furthermore, a ballot is invalid if any of the ciphertexts (R, S) in this ballot (including the implicit sum-ciphertext, if overall proofs are present) have been used in any previous ballot, either explicitly or as an implicit sum-ciphertext, if overall proofs are used.

[9] All proofs used in Helios currently include the commitments as part of the proof representation. A third but minor change is that we remove these commitments in order to obtain more compact proofs. This has no security impact, though.

[10] A potentially simpler solution would be to use, instead of Y and G, the election id that is computed as soon as a Helios election is freezed, which is a hash of all public election parameters, including the public key and group description but also the list of candidates, ballot posting URL, ...

Tally Each trustee first checks that the board is valid so far, i.e., that each ballot verifies and that no ciphertexts (explicit or implicit) are repeated. If these checks fail, the trustee refuses to tally.
The following procedure is repeated once for every candidate or question. The trustee $\mathcal{T}_i$ computes the result-ciphertext $(R_\Sigma, S_\Sigma) = \big(\prod_{V \in \mathcal{V}} R_V, \prod_{V \in \mathcal{V}} S_V\big)$. He produces a decryption factor $D_i = (R_\Sigma)^{x_i}$ and proves it correct with a strong Chaum-Pedersen proof of knowledge π_i^D of an x_i such that $Y_i = G^{x_i}$ and $D_i = (R_\Sigma)^{x_i}$. The hash is taken as $\mathcal{H}(G, Y_i, D, R_\Sigma, S_\Sigma)$. The trustee $\mathcal{T}_i$ then posts the pair (D_i, π_i^D) to the board.
To compute the actual tally t, any trustee can compute $T = S_\Sigma / \prod_{1 \le i \le n} D_i$. To extract t, he takes the discrete logarithm of T which he can do efficiently as $t \le |\mathcal{V}|$.

Result Verify all proofs on the board and return the result(s) t (for each question) on the board if verification succeeds, otherwise return $\perp$.

One can inspect the algorithms to deduce that if everyone acts honestly, an election will run correctly.

Theorem 3. *In the random oracle model, the modified Helios described above satisfies ballot privacy against up to $m = n - 1$ dishonest trustees, assuming that DDH is hard in the underlying group.*

6 Conclusion

The prominence of Helios (it has been used in several real elections, notably in the election of the IACR board of directors) justifies the level of attention it has recently received. Results are divided between finding attacks against ballot privacy (e.g. the method of casting related ballots [23, 33] which we further refine in this paper) and proposing modifications that enable rigorous security proofs [23, 17, 34]. Our paper seems to be the natural convergence point. We identify the use of weak Fiat-Shamir proofs as a source of attacks much stronger than all those previously proposed: we have presented new and unforeseen consequences of these weak proofs and we have shown that switching to their strong counterpart allows for a proof of ballot secrecy for Helios, and provides a crucial assumption on which existing verifiability analyses of Helios rely [34]. In the process, we have made several conceptual contributions: we have defined simulation sound extractability in the random oracle model, proved that the strong Fiat-Shamir transformation yields secure non-interactive zero-knowledge proofs of knowledge, and justified the new notion through applications that include the $\mathsf{Enc}{+}\mathsf{PoK}$ paradigm.

In the remainder of this section we discuss two points that naturally arise from our work.

Usability of wFS. Our results discourage the use of wFS proofs as they may lead to failures in the systems that employ them. Nonetheless, the transformation works well for its original application (and its generalizations [35]) in constructing signature schemes from identification schemes [1], since the statement (essentially the verification key) is fixed in advance. It is interesting to find other settings where wFS can actually be used safely. An intriguing possibility is to exploit malleability of wFS proofs as, for example, in the recent work of Chase et al. [36] that relies on controlled malleability of (standard model) non-interactive zero-knowledge proofs. A necessary first step in this direction is understanding precisely what is the level of malleability of wFS proofs, which we leave for further work.

Practical impact of our attacks. As Helios in its current form has been used in real elections, a discussion of the impact of our attacks in practice is in order. We note that our attacks have been tested and succeeded on the current version of the Helios system on `http://vote.heliosvoting.org`.

Our denial of service attacks may only have an impact on future elections: as far as we know, all Helios elections led to the successful computation of a tally. Regarding our attack on privacy, the scale and outcome of all known real-world elections based on Helios rule out the possibility of effectively violating the privacy of voters through ballot copying. We also checked the 2010 IACR bulletin board and verified that it does not contain any copied ballot.

Our most realistic new attack challenges the verifiability of elections: we showed that corrupted authorities colluding with a single voter can submit an encryption of an arbitrary (positive or negative) number of approvals for any candidate, and that this encryption is indistinguishable from a normal one. This attack could have a decisive impact on approval elections, where the addition of a reasonable number of votes for a single candidate can easily remain undetected.

Many important Helios elections did not use approval voting, though (e.g., the UCL president election and the IACR 2010 election): in those elections, voters were only allowed to select a limited number of candidates. The capability to submit a single malicious ciphertext has a much more limited effect in that case, due to the need to produce an overall proof of the validity of the ballot besides the individual 0/1 proofs. In this context, two possibilities are left to an attacker: either (1) cheat on an individual proof, that is, if allowed to choose up to n candidates, encrypt n votes for a single candidate, 0 for all others, and the overall proof could still be built normally; or (2) cheat on the overall proof, that is, select as many candidates as desired and fake the overall proof. The result of these limited manipulations could not have changed the outcome of the two particular elections mentioned above.

Extending our attack to more than a single ciphertext does not seem immediate. Indeed, our attack requires selecting the private key as a function of the hash of all the commitments in one proof. As a result, building two proofs based on different commitments would require using different election keys, which would not be possible in a single election.

Our second most damaging attack relies on authorities that gain access to the randomness that is used by all voters in order to encrypt their messages. This could possibly be achieved by hiding a function that sends this randomness in the JavaScript code sent by the Helios server to the voters for the ballot preparation, or by forcing server-side encryption. Though more demanding, the effect of this attack can also be more severe as the single actively corrupted voter now only needs to submit a regular ballot.

In all cases, including for approval elections (such as the 2011 IACR election) for which our first attack on verifiability applies, there remain possibilities to remove the concerns that our attacks may raise. For instance, a (possibly independent) set of trustees could be asked to run a mixnet on the ciphertexts posted on the bulletin board of the considered elections, which could then be followed by the individual decryption of all shuffled ballots. An invalid ballot would then be detected immediately, and the trustees would not be able to cheat on the decryption of a second ciphertext.

The existence of such a possibility shows that we still are in a better situation than the one obtained with postal voting. Here, the trustees still have a possibility to demonstrate that they did not manipulate the election. That would be much harder for postal voting, where there is no practical way for the tallying officers to demonstrate that the tally they announce actually corresponds to the authentic ballots.

Acknowledgement. We would like to thank Markulf Kohlweiss for helpful comments. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) under the ICT-2 007-216676 European Network of Excellence in Cryptology II and under the HOME/2010/ISEC/AG/INT-011 B-CCENTRE project. Olivier Pereira is a Research Associate of the F.R.S.-FNRS.

References

1. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature schemes. In: Advances in Cryptology — Crypto'86. Volume 263 of Lecture Notes in Computer Science., Springer-Verlag (1986) 186–194
2. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the Annual Conference on Computer and Communications Security (CCS), ACM Press (1993)
3. Fouque, P.A., Pointcheval, D.: Threshold Cryptosystems Secure against Chosen-Ciphertext Attacks. In Boyd, C., ed.: Advances in Cryptology - Proceedings of ASIACRYPT '01. Volume 2248 of Lecture Notes in Computer Science., Gold Coast, Australia, Springer-Verlag (2001) 351–368
4. Groth, J.: Evaluating security of voting schemes in the universal composability framework. In: Applied Cryptography and Network Security, Second International Conference. Volume 3089 of Lecture Notes in Computer Science., Springer (2004) 46–60
5. Adida, B.: Helios: Web-based open-audit voting. In: In Proceedings of the 17th USENIX Security Symposium (Security 08. (2008) 335–348
6. Adida, B., de Marneffe, O., Pereira, O., Quisquater, J.J.: Electing a university president using open-audit voting: Analysis of real-world use of helios. In: Electronic Voting Technology Workshop/Workshop on Trustworthy Elections. (2009)
7. Helios Headquarters, Princeton University Undergraduate Student Government.: (2010) `http://usg.princeton.edu/officers/elections-center/helios-headquarters.html`.

8. International Association for Cryptologic Research: `http://www.iacr.org/elections/2010`.
9. Helios Specification. `http://documentation.heliosvoting.org/verification-specs`
10. Schnorr, C.: Efficient signature generation by smart cards. Journal of Cryptology **4** (1991) 161–174
11. Chaum, D., Pedersen, T.: Wallet databases with observers. In: Advances in Cryptology — Crypto'92. Volume 740 of Lecture Notes in Computer Science., Springer-Verlag (1992) 89–105
12. Groth, J.: Simulation-sound nizk proofs for a practical language and constant size group signatures. In: Advances in Cryptology — Asiacrypt. Volume 4284 of Lecture Notes in Computer Science., Springer-Verlag (2006) 444–459
13. Shoup, V., Gennaro, R.: Securing threshold cryptosystems against chosen ciphertext attack. Journal of Cryptology **15**(2) (2002) 75–96
14. Tsiounis, Y., Yung, M.: On the security of elgamal based encryption. In Imai, H., Zheng, Y., eds.: Public Key Cryptography, First International Workshop on Practice and Theory in Public Key Cryptography, PKC '98. Volume 1431 of Lecture Notes in Computer Science., Springer (1998) 117–134
15. Schnorr, C.P., Jakobsson, M.: Security of signed elgamal encryption. In Okamoto, T., ed.: Advances in Cryptology - ASIACRYPT 2000. Volume 1976 of Lecture Notes in Computer Science., Springer (2000) 73–89
16. Wikström, D.: Simplified submission of inputs to protocols. In Ostrovsky, R., Prisco, R.D., Visconti, I., eds.: Security and Cryptography for Networks, 6th International Conference, SCN 2008. Volume 5229 of Lecture Notes in Computer Science., Springer (2008) 293–308
17. Bernhard, D., Cortier, V., Pereira, O., Smyth, B., Warinschi, B.: Adapting helios for provable ballot privacy. In: ESORICS. Volume 6879 of Lecture Notes in Computer Science., Springer (2011) 335–354
18. Bulens, P., Giry, D., Pereira, O.: Running mixnet-based elections with helios. In Shacham, H., Teague, V., eds.: Electronic Voting Technology Workshop/Workshop on Trustworthy Elections, Usenix (2011)
19. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among notions of security for public-key encryption schemes. In: Advances in Cryptology - Proceedings of CRYPTO'98, Santa-Barbara, CA, USA, Springer-Verlag - LNCS Vol. 1462 (1998) 26–45
20. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Advances in Cryptology — Crypto'93. Volume 773 of Lecture Notes in Computer Science., Springer-Verlag (1994) 232–249
21. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology **13**(3) (2000) 361–396
22. Pedersen, T.P.: A threshold cryptosystem without a trusted party (extended abstract). In: EUROCRYPT. Volume 547 of Lecture Notes in Computer Science., Springer (1991) 522–526
23. Cortier, V., Smyth, B.: Attacking and fixing helios: An analysis of ballot secrecy. In: CSF, IEEE Computer Society (2011) 297–311
24. Benaloh, J.: Verifiable Secret-Ballot Elections. PhD thesis, Yale University (jan 1987)
25. Pfitzmann, B., Pfitzmann, A.: How to break the direct rsa-implementation of mixes. In Quisquater, J.J., Vandewalle, J., eds.: Advances in Cryptology - EUROCRYPT '89. Volume 434 of Lecture Notes in Computer Science., Springer (1989) 373–381

26. Naor, M., Yung, M.: Public key cryptosystem secure against chosen ciphertext attacks. In: Proceedings of the Annual Symposium on the Theory of Computing (STOC)1990, ACM Press (1990) 33–43
27. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)1999, IEEE Computer Society Press (1999)
28. De Santis, A., Persiano, G.: Zero-knowledge proofs of knowledge without interaction. In: Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)'92, IEEE Computer Society Press (1992) 427–436
29. Cramer, R., Franklin, M.K., Schoenmakers, B., Yung, M.: Multi-autority secret-ballot elections with linear work. In: EUROCRYPT. Volume 1070 of Lecture Notes in Computer Science., Springer (1996) 72–83
30. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: EUROCRYPT. Volume 1233 of Lecture Notes in Computer Science., Springer (1997) 103–118
31. Damgård, I., Groth, J., Salomonsen, G.: The theory and implementation of an electronic voting system. In Gritzalis, D., ed.: Secure Electronic Voting. Volume 7 of Advances in Information Security. Springer (2003) 77–98
32. Bernhard, D., Pereira, O., Warinschi, B.: On necessary and sufficient conditions for private ballot submission. IACR Cryptology ePrint Archive **2012** (2012) 236
33. Smyth, B.: Replay attacks that violate ballot secrecy in helios. IACR Cryptology ePrint Archive **2012** (2012) 185
34. Küsters, R., Truderung, T., Vogt, A.: Clash Attacks on the Verifiability of E-Voting Systems. In: IEEE Symposium on Security and Privacy (S&P 2012), IEEE Computer Society (2012) 395–409
35. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: Advances in Cryptology — Crypto'97. Volume 1296 of Lecture Notes in Computer Science., Springer-Verlag (2000) 410–424
36. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable proof systems and applications. In: EUROCRYPT. Volume 7237 of Lecture Notes in Computer Science., Springer (2012) 281–300
37. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4) (1985) 469–472
38. Bellare, M., Sahai, A.: Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization. In: Advances in Cryptology — Crypto'99. Volume 1666 of Lecture Notes in Computer Science., Springer-Verlag (1999) 519–536

A The Weak Naor-Yung Transformation

The Naor-Yung transformation [26] is a generic transformation from an $\mathsf{IND\text{-}CPA}$ secure encryption scheme into a $\mathsf{IND\text{-}CCA}$ secure one. A transformed ciphertext consists of two ciphertexts for the same message under different keys and a non-interactive proof that both ciphertexts encrypt the same message. To decrypt, one first checks the proof (returning $\perp$ if this fails) and then decrypts the first ciphertext. In the security proof, the challenger chooses the key for the second encryption himself; when the adversary makes decryption queries the challenger uses his second key to answer them. In this step, he relies on the adversary having proven that he encrypted the same message in both ciphertexts. When the adversary makes his challenge query, the challenger uses his IND-CPA challenge query to get a first ciphertext, picks anything he likes for the second ciphertext and forges the proof of equal messages. If the adversary can detect that the challenger is cheating he can either detect simulated proofs or break IND-CPA of the second encryption; if not then the adversary's guess can be used to attack the first encryption.

Consider an implementation using ElGamal as the encryption scheme and a weak generalised Schnorr proof.

KeyGen Pick two secret keys x_1, x_2 and create public keys $Y_1 = G^{x_1}, Y_2 = G^{x_2}$.

Encrypt(M) Pick two random values r_1, r_2 and set $A = G^{r_1}, B = G^{r_2}, C = (Y_1)^{r_1} \cdot M, D = (Y_2)^{r_2} \cdot M$. Create a proof by picking random a_1, a_2 and setting commitments $A_1 = G^{a_1}, A_2 = G^{a_2}, A_3 = (Y_1)^{a_1}/(Y_2)^{a_2}$. Compute $c \leftarrow \mathcal{H}(A_1, A_2, A_3)$ and $f_1 = a_1 + c \cdot r_1, f_2 = a_2 + c \cdot r_2$. The ciphertext is $(A, B, C, D, A_1, A_2, A_3, f_1, f_2)$.

Decrypt(E) Check the verification equations $G^{f_1} = A_1 \cdot A^c, G^{f_2} = A_2 \cdot B^c, (Y_1)^{f_1}/(Y_2)^{f_2} = A_3 \cdot (C/D)^c$ where $c = \mathcal{H}(A_1, A_2, A_3)$.

To forge a ciphertext/proof pair one proceeds as follows.

1. Pick random values $a_1, a_2, a_3, f_1, f_2 \leftarrow \mathbb{Z}_q$. Compute $A_1 = G^{a_1}, A_2 = G^{a_2}, A_3 = G^{a_3}$.
2. Compute $c = \mathcal{H}(A_1, A_2, A_3)$ and $n = 1/c \pmod{q}$.
3. Compute $A = (G^{f_1}/A_1)^n, B = (G^{f_2}/A_2)^n$. Pick a random $c \leftarrow \mathbb{Z}_q$ and set $C = G^c, D = C/((Y_1)^{f_1}/(Y_2)^{f_2}/A_3)^n$.

It can easily be verified that this ciphertext verifies. The first "message" is $C/A^{x_1} = G^{(c-n.x_1 \cdot (f_1-a_1))}$ whereas the second is $D/x_2.B = G^{(c-n\cdot(f_1 \cdot x_1 - f_2.x_2 - a_3) - n \cdot x_2 \cdot (f_2 - a_2))}$. The difference is $G^{n \cdot (a_1 \cdot x_1 - a_2 \cdot x_2 - a_3)}$ from which we see that the two ciphertexts encode the same message if and only if $a_1 \cdot x_1 - a_2 \cdot x_2 = a_3$. This is of course how an honest encryptor chooses his A_3 but it only holds with probability $1/q$ for randomly chosen values. This breaks the proof of the Naor-Yung transformation's IND-CCA security which relies on adaptive soundness of the involved proof. In more detail, because we can recover the "first message" as $C - (Y_1)^{a_1}$ we can distinguish between a real decryptor who will always return this message and the reduction, which will not. We note that hashing the ciphertexts A, B, C, D and the public keys Y_1, Y_2 as well would not only defeat this attack but also restore the security proof.

B The Helios Ballot Format

Briefly, a Helios ballot consists of additively homomorphic ElGamal [37] ciphertexts encoding the vote and disjunctive Fiat-Shamir proofs [1] that the encoded vote is legal. Helios ballots are encoded in a JSON format which is described in more detail in the specification [9].

We assume that there are m candidates standing for election. Helios allows both approval voting and elections where a voter may pick at most a fixed number k of candidates where $1 \leq k \leq m$. A valid vote is thus a vector in $\{0,1\}^m$ with at most k non-zero entries. (Actually Helios elections can consist of several independent "questions", for example to elect candidates to several offices. For simplicity we describe an election with one question only.)

To create a ballot, a voter first obtains the ElGamal public key Y of the election and computes the following values where G is the generator of the group in question.

Ciphertexts For each bit $v \in \{0,1\}$ of the vote (for $i = 1$ to m), the voter picks a random value r and computes the ciphertext $(R, S) \leftarrow (G^r, G^v \cdot Y^r)$.

Individual proofs For each ciphertext the voter proves that it contains 0 or 1. This is done by a disjunctive proof over $v \in \{0,1\}$ of a value r such that $R = G^r$ and $S/G^v = Y^r$.

Let v be the chosen vote and $\overline{v} := 1 - v$.

1. Simulate a proof for $\overline{v}$: pick $c_{\overline{v}}, f_{\overline{v}} \leftarrow \mathbb{Z}_p$ and set $A_{\overline{v}} \leftarrow G^{f_{\overline{v}}}/\alpha^{c_{\overline{v}}}$, $B_{\overline{v}} \leftarrow Y^{f_{\overline{v}}}/(S/\overline{v}G)^{c_{\overline{v}}}$.
2. Pick a nonce $a_v \leftarrow \mathbb{Z}_p$ and compute commitments $A_v \leftarrow G^{a_v}$, $B_v \leftarrow Y^{a_v}$.
3. Compute the challenge $c \leftarrow \mathcal{H}(A_0, B_0, A_1, B_1)$. Note that neither the actual ciphertext (R, S) nor the election public key is hashed.
4. Compute the partial challenge $c_v \leftarrow c - c_{\overline{v}}$ and complete the proof for case v as $f_v \leftarrow a_v + c_v.r$.

The proof is $\pi = (A_0, B_0, A_1, B_1, c_0, c_1, f_0, f_1)$.

Overall proof If the election rules stipulate that a voter may only choose up to k of m candidates, the voter additionally computes the homomorphic sum of all his ciphertexts and executes a further proof on this sum, proving that the contained plaintext is in $\{0, \ldots, k\}$ (rather than $\{0,1\}$). This is done similarly to the individual proofs, simulating proofs for all values except the correct one.

The final ballot consists of the list of ElGamal ciphertexts, the list of individual proofs and if required, the overall proof. We write $b = ((c_1, \ldots, c_m), (\pi_1, \ldots, \pi_m) [, \pi_\Sigma])$ to denote a Helios ballot where $c_i = (R_i, S_i)$ and π_i is as described above.

Verification For each ciphertext with proof

$$(R, S, \pi = (A_0, B_0, A_1, B_1, c_0, c_1, f_0, f_1))$$

check the following equations

$$G^{f_0} = A_0 \cdot R^{c_0} \quad (1)$$
$$G^{f_1} = A_1 \cdot R^{c_1} \quad (2)$$
$$Y^{f_0} = B_0 \cdot S^{c_0} \quad (3)$$
$$Y^{f_1} = B_1 \cdot (S/G)^{c_1} \quad (4)$$
$$c_0 + c_1 = \mathcal{H}(A_0, B_0, A_1, B_1) \quad (5)$$

If an overall proof π_Σ is present, compute the homomorphic sum of all ciphertexts (R_Σ, S_Σ) and check π_Σ against this (for range $\{0,\dots,k\}$):

$$\forall i: \ G^{f_i} = A_i \cdot R^{c_i} \tag{6}$$

$$\forall i: \ Y^{f_i} = B_i \cdot (S/G^i)^{c_i} \tag{7}$$

$$\sum_{i=0}^{k} c_i = \mathcal{H}(A_0, B_0, A_1, B_1, \dots, A_k, B_k) \tag{8}$$

C Proofs Related to Helios Ballot Privacy

C.1 Theorem 2: IND-CPA + SSE-PoK gives NM-CPA

We use a theorem of Bellare and Sahai [38] that NM-CPA is equivalent to the following indistinguishability-based game in which the adversary may make one *parallel* decryption query:

Definition 6. *The following game is the* IND *version of* NM-CPA.

1. The challenger runs $(pk, sk) \leftarrow \mathsf{KeyGen}$ and gives pk to the adversary.
2. The adversary picks two messages m_0, m_1 and hands them to the challenger who picks a bit $\beta \leftarrow \{0,1\}$ and returns an encryption c^* of m_β.
3. The adversary may submit a vector $\boldsymbol{c} = (c_i)_i$ of ciphertexts. For each c_i, if $c_i = c^*$ (the challenge ciphertext) then the challenger returns $\perp$, otherwise he returns $\mathsf{Dec}(sk, c_i)$.
4. The adversary may submit a guess β' for β.

The adversary wins if $\beta' = \beta$ and his advantage is defined as $|\mathbf{Pr}[\beta' = \beta] - 1/2|$.

Note that while the adversary may submit as many ciphertexts as he wishes, he may not submit them adaptively, i.e. he may not wait to see the decryption of c_1 before choosing c_2.

Proof. (Theorem 2) We begin with a game-hop from the original game to a game in which, on receipt of (m_0, m_1), we create the $\mathfrak{E}$-ciphertext as before but add a simulated $\mathfrak{P}$-proof. If the adversary can detect this change then he can tell real from simulated proofs and break the zero-knowledge property of $\mathfrak{P}$.

We construct a reduction to IND-CPA security of $\mathfrak{E}$ with black-box access to the adversary. Run a copy of the simulator $\mathcal{S}$ for $\mathfrak{P}$. Obtain a public key pk from the challenger and forward it to the adversary. When the adversary submits messages m_0, m_1, we submit them to our challenger to get a challenge ciphertext E. We use $\mathcal{S}$ to simulate a proof π for E and return the pair $c^* = (E, \pi)$ to the adversary. We let $\mathcal{S}$ handle all random oracle queries made by the adversary. If we can extract the witnesses to the adversary's ciphertexts (as we no longer have the secret key to answer the decryption query) then the reduction is identical from the adversary's point of view to the previous game. Therefore, we can simply forward the adversary's guessed bit to our challenger to break IND-CPA of $\mathfrak{E}$.

At the moment when $\mathcal{A}$ outputs his decryption query, that we can view as vectors $(\boldsymbol{Y}, \boldsymbol{\pi})$ of $\mathfrak{E}$-ciphertexts and $\mathfrak{P}$-proofs, we need to build an adaptive prover $\mathcal{A}'$ that we can hand to the extractor $\mathcal{K}$ along with a tarnscript of an execution that produced exactly $(\boldsymbol{Y}, \boldsymbol{\pi})$. $\mathcal{A}'$ simulates a copy of $\mathcal{A}$, giving it as input the public key pk that we got from our challenger. Random oracle queries of $\mathcal{A}$ we forward to $\mathcal{K}$ and return the result. When our simulated $\mathcal{A}$ makes a challenge query, if the execution so far has been identical to the one of the "main" run of $\mathcal{A}$ against our reduction, $\mathcal{A}'$ submits the challenge ciphertext E that we got earlier as a Simulate call to $\mathcal{K}$ to get a π' and returns (E, π') to $\mathcal{A}$. If the current execution of $\mathcal{A}$ does not match the main one (because $\mathcal{K}$ has forked $\mathcal{A}'$ before the challenge query), $\mathcal{A}'$ takes m_0, m_1, draws a fresh bit β', encrypts $m_{\beta'}$ to get E' and asks this as a Simulate call to $\mathcal{K}$, then returns the ciphertext and proof to the adversary.

We construct a transcript for $\mathcal{K}$ from the $\mathcal{S}$-queries of $\mathcal{A}$ in the main execution including the one Simulate query that $\mathcal{A}'$ would have used to get the proof for the challenge ciphertext. We can now run $\mathcal{K}$ on this transcript and on $\mathcal{A}'$ and obtain the witnesses to $\boldsymbol{Y}$, allowing us to answer the decryption queries of the main run of $\mathcal{A}$ and complete the reduction.

C.2 Proof of Theorem 3

The rest of this subsection is devoted to a proof of Theorem 3. Assuming that DDH is hard, we know that ElGamal encryption is IND-CPA and therefore, ElGamal+SSE-PoK is NM-CPA by Theorem 2.

We first give the simulator $\mathcal{S}$. $\mathcal{S}$ is called by the game if $\beta = 1$, i.e. the R board is visible but the adversary expects to see the result matching the L board.

The Simulator. The first task of $\mathcal{S}$ is to retrieve the secret keys $x_{\mathcal{T}}$ of all trustees. $\mathcal{S}$ gets the keys of all honest trustees as part of its input. For corrupt trustees, $\mathcal{S}$ launches a new instance of the adversary once for each such trustee and uses the adaptive PoK property of this administrator's proof of knowledge of his secret key to extract said key. These proofs must exist on the board as the game would otherwise have aborted at the end of the setup phase. Note that $\mathcal{S}$ only has to run the simulated instances until the end of the setup phase and that until this point, the game itself (which is outside the control of $\mathcal{S}$) only simulates honest trustees, which $\mathcal{S}$ can do as well as it has their state in its input.

$\mathcal{S}$ can now decrypt all the adversary's ballots and obtain the result r_L of the left board. $\mathcal{S}$ picks one honest trustee $\mathcal{T}'$, computes the correct decryption shares $D_{\mathcal{T}}$ for all other trustees (as $\mathcal{S}$ has their secret keys) and solves the equation $G^{r_L} = \beta_R / \prod_{\mathcal{T} \in \mathcal{T}} D_{\mathcal{T}}$ for $D_{\mathcal{T}'}$. $\mathcal{S}$ forges a proof that this decryption share is correct w.r.t. $Y_{\mathcal{T}'}$ and posts this share and proof; all other honest trustees tally correctly.

Indistinguishability. Let G_0 be the ballot privacy game in the case that $\beta = 0$ and G_1 be the same game when $\beta = 1$. We wish to show that G_0 and G_1 are computationally indistinguishable.

Let G_A be G_0 where all proofs made by honest parties are made using the ZK simulator. G_0 and G_A are computationally indistinguishable: we can switch the proofs from real to simulated, one by one. If the adversary can detect any of these steps then we can use him to construct an adversary that breaks the zero-knowledge property of an adaptive proof.

Let G_B be G_A with the following change: let $\mathcal{T}'$ be one of the honest trustees. When $\mathcal{T}'$ is asked to tally, we extract all the adversary's secret keys in the same manner as the simulator. Then we decrypt all the adversary's ballots to get his votes and use these and the honest votes, which we know already, to compute the correct tally r. We compute the ciphertext of the result $(R_\mathcal{R}, S_\mathcal{R})$ and all decryption shares for trustees $\mathcal{T} \neq \mathcal{T}'$ using the secret keys and set $D_{\mathcal{T}'} = S_\mathcal{R} \cdot G^{-r} / \prod_{\mathcal{T} \neq \mathcal{T}'} D_\mathcal{T}$. The adversary has no choice over the decryption shares that he outputs: if he creates anything other than the correct share, which is uniquely determined by the public key of the trustee in question and the ciphertext of the result, then he cannot provide an adaptive proof as the underlying statement is false. As we have not changed any values output to the adversary, only the method by which they were computed, G_B is indistinguishable from G_A.

Let G_C be G_B with the change that we put the right instead of the left ballots on the board for honest voters. To show that the adversary cannot tell G_B from G_C we use a hybrid argument and a reduction to $\mathsf{NM\text{-}CPA}$ of the encryption scheme used in our proposed Helios, "ElGamal with individual strong proofs".

Number all ciphertexts made by the game on behalf of honest voters in the order that they are created. Suppose that an adversary can tell the difference when the i-th ciphertext is switched from "left" to "right". Let R be the following reduction to the indistinguishability with one parallel decryption query variation of the $\mathsf{NM\text{-}CPA}$ game, for the encryption scheme "ElGamal encryption with individual 0/1 proofs".

- Obtain a public key Y from the challenger and set the public key of our designated trustee $\mathcal{T}'$ such that the election public key becomes Y. That is, we set $Y_{\mathcal{T}'} = Y / \prod_{\mathcal{T} \neq \mathcal{T}'} Y_\mathcal{T}$. The distribution of this key is uniform, as the key was in all previous games so the adversary cannot detect a difference here.
- Simulate the game G_B (using "left" votes) until $i - 1$ ciphertexts have been created.
- To create the i-th ciphertext, use the challenge oracle on the left and right plaintexts determined by the corresponding votes.
- Continue simulating the game but use the "right" votes from now on.
- In the tallying phase, all honest adminstrators except $\mathcal{T}'$ tally normally. Submit all the adversary's ciphertexts (with their individual proofs) for all questions to the decryption oracle in one parallel decryption query. This gives the adversary's votes; together with the honest votes compute the "left" result r_L. $\mathcal{T}'$ creates his decryption share $D_{\mathcal{T}'}$ as in G_B in order to produce the result r_L.

– Forward the adversary's guess at the secret bit to the NM-CPA challenger.

If the challenger chose the left message, the i-th ciphertext the adversary sees is a "left" one, otherwise it is a "right" one. All other elements are handled the same way as in G_B so R breaks NM-CPA with the same advantage as the adversary distinguishes the hybrids.

Once all ciphertexts are switched to "right" ones, this is exactly game G_C. We can now switch back all simulated proofs to real ones, except the one in the decryption share of $\mathcal{T}'$. This is again undetectable by the adversary. Once this is done, the adversary is playing against game G_1 as we act identically to the simulator $\mathcal{S}$.

This completes the proof that G_0 and G_1 are indistinguishable.