Running mixnet-based elections with Helios

Philippe Bulens
BlueKrypt, OAdeo
Belgium

Damien Giry
BlueKrypt, OAdeo
Belgium

Olivier Pereira
Université catholique de Louvain
ICTEAM – Crypto Group
B-1348 Louvain-la-Neuve – Belgium

Abstract

The Helios voting system is an open-audit web-based voting system that has been used by various institutions in real-stake elections during the last few years. While targeting the simplicity of the election workflow, the homomorphic tallying process used in Helios limits its suitability for many elections (large number of candidates, specific ballot filling rules, . . .).

We present a variant of Helios that allows an efficient mixnet-based tallying procedure, and document the various choices we made in terms of election workflow and algorithm selection. In particular, we propose a modified version the TDH2 scheme of Shoup and Gennaro that we found particularly suitable for the encryption of the ballots.

Our Helios variant has been tested in two multi-thousand voter elections. The lessons taken from the first of these elections motivated some changes into our procedure, which have been successfully experimented during the second election. Voter survey data are also presented.

1 Introduction

Helios [1, 2] is an open-audit web-based voting system, designed for low coercion elections. The correctness of Helios elections is guaranteed through a series of audit procedures. Essentially, voters can (i) audit the ballots they prepare in order to be convinced that they actually reflect their vote intention, (ii) verify that their ballot was correctly recorded by the voting server on a web bulletin board, (iii) check on the bulletin board that all the ballots that will be included in the tally correspond to an actual voter (unless this is explicitly prevented by the election organizers) and (iv) verify that the tallying procedure has been run correctly from the ballots displayed on the bulletin board.

The privacy of the votes relies on the honesty of the voting client, on the security of various cryptographic algorithms, and on the honesty of a set of trustees who own shares of a decryption key.

Since Helios 2.0 [2], and in all real-stake Helios elections, the election tally is computed by the public homomorphic aggregation of individual encrypted ballots into an encryption of the election outcome [8], which is then decrypted in a distributed way by a set of trustees. From a cryptographic point of view, the implemented protocol follows the general ideas of Cramer et al. [10], but treats multiple candidate through separate ciphertexts, as proposed by Hirt [20] for instance. It also incorporates several ideas from Benaloh [6] that enable auditing the ballot preparation system.

While this approach has huge advantages in terms of simplicity, it also comes with important limitations.

- The validity of the ballots needs to be proved by the voters, and the complexity of those proofs typically grows linearly with the number of candidates. As the Helios code is mostly interpreted in JavaScript, this becomes an issue as soon as a few dozen of candidates are to be considered.
- The validity proofs need to be modified for every type of ballot filling rule, which can become quite complex in many cases (see Groth [18] for instance).

Mix networks [7] offer another approach for the tallying of an election. They provide anonymization services for encrypted ballots, by shuffling and rerandomizing them in a verifiable way. The anonymized ballots can then be safely decrypted, and the ballot validity verification and counting procedure can be achieved publicly on the decrypted ballots rather than in the encrypted domain.

A crucial advantage of this approach is that it does not bring any limitation on the format of the ballots that are encrypted. This makes the ballot preparation procedure much simpler and efficient, which is highly desirable for computationally limited voting clients.

We note however that the decryption of all ballots comes with an important limitation from a security point of view: much more information is revealed about the voter's intentions than in the case of homomorphic tallying techniques. While this amount of information is identical to the one revealed in traditional paper-based elections, this however brings several difficulties:

- It makes it impossible to satisfy the requirements of some tallying procedures, e.g., procedures involving weighting of votes while fully hiding the level of support of the candidates as a function of the different weights [2].
- It increases the risk of coercion compared to approaches that only disclose the election result, e.g., facilitating the so-called "Italian attack" in which a coercer requires the presence of a ballot with a very specific and improbable voting pattern in the urn.
- It requires more care about independent ballot submission as, for instance, a voter who submits a (possibly re-randomized) copy of someone else's ballot could then search for two identical decrypted ballots once the tally is completed in order to find out who that voter voted for.

The situation can sometimes be improved by splitting each ballot into several sub-ballots that are mixed independently, but this should not prevent the eventual validity verification of the ballots nor increase too much the cost of the ballot preparation.

Nevertheless, we feel that the possibility to adopt mixnet-based tallying offers a valuable middle-ground for different types of elections, which would be much more difficult to handle otherwise.

Contributions We describe the design of a Helios variant that allows mixnet-based tallying, and motivate the choice of the various algorithms and procedures that we use for ensuring the security of the ballot submission procedure, running the ballot mixing, and computing the tally. In particular, we propose a variant of the TDH2 encryption scheme of Shoup and Gennaro [27] that preserves submission security and embeds a homomorphic ElGamal ciphertext while preserving the efficiency of the original scheme.

We also document two experiences in running mixnet-based elections involving several thousands of voters and relying on different cryptographic algorithms and election procedures. The strengths and limitation of the tested approaches are discussed, and voter survey data are provided. We believe that most of the lessons we learned by running these elections and from the survey data are of interest for e-voting and verifiable elections in general and, hence, have a scope much broader than the specific use of Helios or mixnets.

Roadmap We start by reviewing the workflow of a Helios election in Section 2. Then, Section 3 describes and motivates the workflow we propose for Helios elections with mixnet-based tallying. The specifics of the algorithmic choices we made is then documented in Section 4. Eventually, our experiences in running mixnet-based elections are described in Section 5.

2 Overview of a Helios election workflow

We review the workflow of a Helios 3.1 homomorphic tallying-based election, as released in March 2011, highlighting the aspects that are most important for the introduction of our mixnet-based variant.

Election setup. The election administrator creates a Helios election by defining a series of parameters: the election description and URL, the election questions and possible answers together with ballot filling rules (approval voting, limited number of answers, ...), the set of trustees who hold shares of the election private key, the election public key, the list of voters or the fact that this list is kept open and the choice to mask (or not mask) the voter names through aliases on the bulletin board.

The Election Fingerprint is then computed by hashing all these parameters. This fingerprint is expected to be distributed to the voters through as many channels as possible, as it enables them to verify that they are preparing their ballot in the correct way.

Ballot preparation and submission. Once the election is open, voters can prepare ballots for submission to the voting server. To this purpose, they can either use their own ballot preparation system (or the one proposed by someone they trust, e.g., the candidate they would like to support), or use the ballot preparation system proposed by Helios.

The Helios ballot preparation system (BPS) downloads all election parameters, then recomputes and displays the Election Fingerprint for verification by the voters. Voters can then make their choices. The ballot is then encrypted, and ballot validity proofs are computed. This phase can quickly become computationally intensive as soon as the number of candidates increases: it essentially requires 6 modular exponentiations per candidate, to which must be added the cost of the proofs that guarantee that a limited number of candidates have been selected, if such proofs are needed.

The BPS then commits on the resulting ballot by displaying the hash of that ballot for the voter, and proposes to either submit the ballot to the voting server or to audit it, following an approach proposed by Benaloh [6]. If the voter chooses to audit the ballot, the BPS is requested to display the ballot content together with all the randomness that has been used to build it. This enables the voter to use any ballot verification software he likes to verify whether the ballot was built correctly. As one possible cause of failure could be the presence of malware on the client machine (as acknowledged by the Helios authors [2] and later illustrated by Estehghari et al. [13]), Helios 3.x simplifies the verification of ballots on different devices by allowing to post spoiled ballots on a bulletin board from which anyone, including the voter himself, can later download and verify the ballots. This procedure offers correctness guarantees as long as voters perform ballot audits using uncompromised machines, but of course does not help for privacy. Helios, targeting the simplicity of the voting procedure, has not been designed to provide a full protection of voters against compromised voting clients and should not be used in elections in which this is a concern.

Eventually, if the voter decides to submit the prepared ballot, then the voter authentication process takes place, the ballot is sent to the voting server, and the voter can verify that it was correctly received on the election web bulletin board.

Election tallying. Once the ballot submission phase is complete and everyone agrees on the web bulletin board content, the tallying phase starts.

This phase begins with the verification of the validity of all ballots, which is a completely public operation. Then, all encrypted ballots are homomorphically aggregated into encryptions of the number of votes received by each candidate. These ciphertexts (one per candidate) are eventually decrypted by the trustees, who also provide a proof of the correctness of the decryption procedure.

The amount of computation that the trustees need to perform is actually quite limited here: each trustee needs to compute 3 modular exponentiations per candidate, which is even cheaper than the cost of preparing a ballot. This operation can therefore be performed conveniently through the Helios web interface, like the rest of the election process.

3 Workflow for mixnet-based elections

We now discuss the general workflow of mixnet-based elections, as we designed it, and leave for the next section the discussion of the concrete algorithms chosen to realize this workflow.

3.1 Election setup

The setup for a mixnet-based election is similar to the one of a homomorphic tallying-based election, except for the role of the trustees.

While trustees were only needed for the decryption of the election outcome in homomorphic tallying-based elections, two different operations need to be completed now: the shuffling and the decryption of the shuffled ballots.

While several mixnets offer the possibility to shuffle and decrypt at the same time (starting with the initial work of Chaum [7]), we prefer adopting a more modular approach and decided to separate the two concerns and to have two groups of trustees:

1. *Shuffling trustees* are responsible of shuffling the ballots. These trustees do not need to store any secret: they need to internally produce secret data (a random permutation, …) during the shuffling process, but should delete these data as soon as their task is completed.

2. *Decryption trustees* are responsible of decrypting the shuffled ballots. These trustees each hold a share of the election private key, just as in homomorphic tallying-based elections.

As a result, we need to rely on the decryption trustees to not violate the privacy of the voters, which they could do by jointly decrypting the ballots before shuffle, for instance. We also need to rely on the mixing trustees to keep their shuffling permutations secret in order to make sure that nobody can link the submitted ballots to the decrypted ballots. In both cases, a coalition of all the trustees of the same group is needed to violate the privacy of the voters. The correctness of the tallying procedure however only relies on computational assumptions (through the soundness of the zero-knowledge proofs and the collision resistance of the hash functions that are used to check integrity) and not on the honesty of the trustees.

From an organizational point of view, it is clear that the decryption trustees must be chosen in advance, as they need to jointly produce the election public key before the election starts. The shuffling trustees, however, do not need to be determined in advance as they are not required to generate any secret before the election starts. So, in order to increase the flexibility of the election process, we decide to leave the shuffling trustees undetermined during the setup phase (meaning, they do not need to be included in the Election Fingerprint), and rely on the trustees to only accept to decrypt ballots when there is an agreement that a satisfactory shuffling procedure happened: we need to trust them on the privacy of the votes anyway.

As running a verifiable shuffle is a computationally intensive tasks, we believe that keeping the freedom to adapt the shuffling procedure on-the-fly is an interesting feature. It also opens the possibility to have a set of servers offering verifiable shuffling services available around the world, and to let election organizers query those servers at tallying time, depending on their availability.

3.2 Ballot preparation and submission

The ballot preparation workflow remains identical to the one used for elections targeting homomorphic tallying. The cryptographic algorithms used for the ballot preparation however serve very different purposes.

Simpler ballots. The ballot format required for homomorphic tallying needs to enable everyone to check, in the encrypted domain, the validity of all the votes, and to compute an encryption of the election outcome.

In contrast, since a mixnet based tallying procedure involves decrypting all individual ballots after shuffling, the validity of the ballot can be checked after decryption and not at submission time. Furthermore, the format of the encrypted messages does not need to allow homomorphic ballot aggregation.

While the decryption of every individual ballot considerably simplifies the ballot preparation by obliterating the need to demonstrate the validity of the encrypted ballots, it also makes public the full content of all ballots, which opens the way to some attacks, like the Italian attack mentioned earlier for instance.

While this cannot be fully prevented with mixnet-based tallying, the privacy of the voters could also be compromised through more sophisticated attacks in which some voters would copy ballots submitted by other voters and look for identical ballots after decryption, which should allow them to recognize the copied ballots with high probability when there is a large number of ways to fill in a ballot. Furthermore, the detection of non independent ballots by observers can actually be made quite difficult as, instead of simply copying somebody else's ballot, voters could produce different but related ballots and/or rerandomize previously submitted ballots, as originally demonstrated by Pfitzmann and Pfitzmann [23].

Submission Secure Augmented cryptosystems. This last attack can be prevented by making sure that ballots are submitted independently of each other, which can be enforced by using a submission secure augmented (SSA) cryptosystem [32]. Such a cryptosystem is essentially an IND-CCA2 encryption scheme whose ciphertexts contain ciphertexts from another basic encryption scheme. This basic encryption scheme is typically chosen to be homomorphic, and a stripping procedure must enable everyone to check the validity of an augmented ciphertext without compromising the confidentiality of the basic ciphertext (see Section 4.1.1 for a precise definition). The non-malleability of the IND-CCA2 augmented ciphertext now guarantees that the only way to submit a ciphertext that would be related to another ciphertext is to submit an exact copy of it, which is easy to detect.

Now, as trustees might decide to use the same private key in several contexts (questions, elections, . . .), it is desirable to use an SSA cryptosystem in which the verification of ciphertext validity includes the verification of a scope in which that ciphertext can be used: the search for exact copies can then be reduced to only include ciphertexts that are valid within that specific scope. To this purpose, we also embed the Election Fingerprint as label in each ciphertext, which prevents copying a ciphertext from one context and submitting it in another that would use the same key. So the search for duplicate ciphertexts can be restricted by usage context.

It would be appealing to go one step further and to also include the identity of the voters as part of the label of each ciphertext, as suggested by Cramer et al. [10] for instance: a copied ciphertext would then be declared invalid immediately because it would contain the wrong voter identity. This solution has not been adopted here, however: in order to prevent the voting client from adapting its bahaviour according to the identity of the voter, the identification phase is performed only after the ciphertexts have been produced and committed to by the BPS.

Submission independence in elections based on homomorphic tallying. As we discussed above, independent ballot submission is often an important element to ensure the privacy of the votes when mixnet-based tallying is used and the space of valid ballots is large compared to the number of voters.

The situation is however substantially different when homomorphic tallying is used and when the number of candidates is small compared to the number of voters, e.g., when thousands or even millions of voters must choose between a few candidates (as it happens for the presidential elections in many countries for instance), or when it can be considered unrealistic that a large enough number of voters would be accepting to give up their vote and transform it into a copy of somebody else's vote in order to violate the privacy of that voter.

Indeed, a homomorphic tally only discloses the election outcome instead of individual ballots, which prevents the search for related ballot completion patterns and strongly limits the privacy impact of ballot dependence. Special care needs however to be taken in order to prevent unexpected behaviors of the adopted cryptographic schemes, as demonstrated by Benaloh [5] for instance, or in very specific elections, as illustrated by Ben-Or and Linial [4] or by Cortier and Smyth in the case of Helios for instance [9]. For example, if a voter copies somebody else's vote in a three voters election, this voter will always be able to determine the choices of the other two voters by inspection of the election outcome.

In some cases, these issues can be further mitigated by running a tally using secure multiparty computation techniques that would only reveal the identity of the election winner without disclosing the repartition of the votes between the candidates.

In that context, one could consider the possibility to submit a ballot that would be an (unlinkable) copy of somebody else's ballot as an interesting feature that could be offered by voting systems that keep track of the link between voters and their encrypted ballot. As the purpose of elections is typically the delegation of some decision power to the election winner(s), it does not seem very different to already include the possibility of this delegation as part of the voting process.

While today's classical paper-based voting systems enforce ballot independence, it is not clear whether this is just a side effect of the adoption of urns for ensuring the confidentiality of the votes, or a desired feature. It indeed appears that a system that would allow copying somebody else's vote intention in a private way would not come short of the lists of requirements for voting systems that we consulted, e.g, those of the Council of Europe [14] and Wolkamer [34].

By contrast, we observe that ballot independence is enforced by most technical security definitions for voting systems that have been proposed in the computer security and cryptography communities. However, as in the paper voting case, it often seems to come as a side effect of the most immediate way to specify the security properties of a voting system, especially in the commonly adopted simulation based security setting [5, 12, 17, 31, 32]: security definitions that would allow the submission of non-independent ballots would definitely be more complex than the current ones.

We believe it would be an interesting question to further investigate the implications of the use of a voting system allowing the submission of (unlinkable) copies of ballots in terms of security and quality of the resulting choices.

3.3 Election tallying

Once everyone agrees on the election bulletin board content (this could be made official by publishing a signed version of its content [2]), the tallying phase starts.

At first, the validity and uniqueness of all SSA ciphertexts included in the ballots is verified, and the embedded homomorphic ciphertexts are extracted before entering into the shuffling phase.

Shuffling. A verifiable reencryption mixnet [22, 24] is made of a sequence of mix servers that each take a sequence of homomorphic ciphertexts as input and compute a shuffled and reencrypted version of these ciphertexts together with a proof of valid shuffling.

The computational load of a mixing server is fairly large, compared to the work needed to produce one ballot or to decrypt a single ciphertext: most efficient verifiable shuffles require 6 to 8 modular exponentiations per ElGamal ciphertext (see Groth [19] for some performance analyses, for instance). As a result, we need to depart from the browser-only approach of Helios, and ask the trustees to run the shuffle out of their browser. A Python script is proposed to this purpose, with code portability and simplicity in mind.

Decryption and ballot counting. Once the decryption trustees are convinced that proper mixing happened, that is, that the correct ballots have been mixed by a set of shuffling trustees whose coalition is unlikely, they proceed to the decryption of the resulting ciphertexts.

Again, this task is much more intensive than the election outcome decryption procedure in elections based on homomorphic tallying, as we now need to decrypt all shuffled ballots. For that reason, we propose another Python script that can be used by each trustee to compute locally their share of the decryption of each ballot, together with the corresponding validity proofs.

Eventually, the decryption shares are gathered and combined, allowing everyone to publicly recover the plaintext of each individual ballot. At this point, the validity of each ballot with respect to the ballot filling rules can be publicly verified, and the final tally computed.

4 Algorithm selection

We now describe the algorithmic choices we made for our implementations of the above described election workflow.

4.1 Ballot preparation

As discussed in the previous section, we need to use a submission secure augmented cryptosystem for encrypting votes. One ciphertext is computed per question, instead of one ciphertext per candidate as it is done in Helios 3.x.

4.1.1 Submission Secure Augmented cryptosystems

We recall the definition of augmented cryptosystem [32].

Definition 1. *A cryptosystem* $\mathsf{CS} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is an augmentation of a cryptosystem* $\mathsf{CS}^B = (\mathsf{Gen}^B, \mathsf{Enc}^B, \mathsf{Dec}^B)$ *if there exists an augmentation algorithm* $\mathsf{Aug} \in PPT$ *and a stripping algorithm* $\mathsf{Strip} \in PT$ *such that:*

1. *On input* 1^n*,* Gen *computes* $(pk^B, sk^B) = \mathsf{Gen}^B(1^n)$ *and* $(pk^A, sk^A) = \mathsf{Aug}^B(pk^B)$ *and outputs* $(pk, sk) = ((pk^A : pk^B), (sk^A : sk^B))$.
2. *On input* $((sk^A : sk^B), c)$*,* Dec *outputs* $\mathsf{Dec}^B_{sk^B}(\mathsf{Strip}_{pk^A, sk^A}(c))$.

The general idea for such a scheme is to choose CS^B as a standard homomorphic encryption scheme, e.g., ElGamal in our case, and then to augment it into an IND-CCA2-secure cryptosystem CS in such a way that a ciphertext for CS can be stripped from the non-malleable layer provided through the augmentation in such a way that only a CS^B ciphertext remains.

In order to be submission secure, an augmented cryptosystem must prevent any adversary from gaining a non-negligible advantage in a game that follows the IND-CCA2 security game, except that the adversary is allowed to interact with a series of augmentations of the same basic private key, and to query for some augmented private key as long as he does not perform any further decryption query for that specific augmentation. We refer to Appendix A for the precise definition of the submission security game, as it is enough for our purpose to consider that an SSA cryptosystem is an IND-CCA2 secure augmented cryptosystem.

4.1.2 Selection criteria

Various schemes could be used as SSA cryptosystems (a list is provided by Wikström [32].) We adopted the following selection criteria:

1. Stick as much as possible to the Helios 3.x security model for the cryptographic primitives.
2. Give priority to efficient solutions for the client.

The security of the cryptographic primitives used in Helios 3.x can be roughly summarized to rely on the DDH assumption in the random oracle model: the security of ElGamal encryption relies on the DDH problem, while the proofs are based on the hardness of the DL problem and on the Fiat-Shamir heuristic.

Choosing to stick to this model rules out a common and efficient choice for an SSA cryptosystem that consists in adding a Schnorr proof of knowledge [25] of the randomness used for the ElGamal encryption. This approach of building a CCA2 secure encryption scheme has been studied by Tsiounis and Yung [29], and by Schnorr and Jakobsson [26], but comes at the price of either a nonstandard extraction assumption in the first analysis, or of considering G in the generic group model in the second analysis (see Shoup and Gennaro [27] for discussion). It is unknown whether a new proof technique would allow proving the security of this solution in the random oracle model based on a standard assumption. So, we preferred to not adopt that solution here.

Another common generic approach for building a CCA2 secure encryption scheme from any CPA-secure scheme is the Naor-Yung transform [21] (with threshold variants discussed by Fouque and Pointcheval [16]), which consists in encrypting a message twice with different public keys and proving that the two ciphertexts are encryptions of the same plaintext. While this solution presents the advantage of being independent of the specific CPA secure scheme (i.e., it can be used for Paillier encryption as well as ElGamal) and to satisfy our cryptgraphic model requirements, it appeared to be possible to use more efficient solutions.

We now describe in more details two such solutions that we actually used in two elections.

4.1.3 A solution based on the Cramer-Shoup cryptosystem.

Cramer-Shoup as an SSA cryptosystem. Wikström [32] proposes a solution for input submission that is secure in the standard model (that is, without random oracle), based on the Cramer-Shoup cryptosystem [11]. This solution is as follows (for clarity, we only describe the non-distributed version of the cryptosystems; the extension to the distributed setting is immediate from the way ElGamal is distributed in Helios).

1. On input 1^n, Gen^B selects a prime integer p such that $|p| = n$, $q = (p-1)/2$ is prime and $p = 3 \bmod 4$, together with independent generators g and g_0 of the subgroup G of $\mathbb{Z}_p^*$ of order q. A collision resistant hash function $H : G^3 \times \{0,1\}^n \to \mathbb{Z}_q$ is also selected. A private exponent $z \in \mathbb{Z}_q$ is selected, and the public group element $h := g^z$ is computed. The output (pk^B, sk^B) of Gen^B is then defined as $((p, g, g_0, H, h), (p, z))$.
2. On input $pk^B := (p, g, g_0, H, h)$, Aug^B selects private exponents $(x, x_0, y, y_0) \in \mathbb{Z}_q^4$, and computes the public group elements $c := g^x g_0^{x_0}$ and $d := g^y g_0^{y_0}$. The pair (pk^A, sk^A) is then defined as $((p, c, d), (x, x_0, y, y_0))$.

3. On input $(pk^A : pk^B) := ((p,c,d),(p,g,g_0,H,h))$, message $m \in \mathbb{Z}_q^*$, and public label $L \in \{0,1\}^n$, Enc chooses a random $r \in \mathbb{Z}_q$ and computes a Cramer-Shoup ciphertext $(L,u,u_0,e,v) := (L,g^r,g_0^r,m^2h^r,c^rd^{rH(u,u_0,e,L)})$.

4. On input $(pk^A,sk^A,cs) := ((p,c,d),(x,x_0,y,y_0),(L,u,u_0,e,v))$, Strip returns (u,e) if $v = u^x u_0^{x_0}(u^y u_0^{y_0})^{H(u,u_0,e,L)}$ or $(\perp,\perp)$ otherwise.

5. On input $(sk^B,eg) := ((p,z),(u,e))$, Dec^B outputs $\perp$ if $(u,e) = (\perp,\perp)$ or, otherwise, computes $m' = e/u^z$ and outputs the unique value in $\{m'^{\frac{p+1}{4}}, p - m'^{\frac{p+1}{4}}\}$ that lies between 1 and q.

It can be observed that we are considering a specific instance of the Cramer-Shoup scheme here, which allows the efficient encryption and decryption of messages in $\mathbb{Z}_q^*$ (through the message squaring and square root extraction in Items 3 and 5.) Other choices would be possible, e.g., if the message space is small, one could use exponential ElGamal as in Helios 2.x and 3.x for instance.

Using Cramer-Shoup for ballot submission. We use the Cramer-Shoup scheme described above as follows.

First, the decryption trustees generate a Cramer-Shoup key pair in a distributed way and submit the public key, which is then included in the election description. Then voters encrypt their ballots using the Election Fingerprint as label.

Now, once the ballot submission phase is complete, the private key $sk^A = (x,x_0,y,y_0)$ is published, and everyone can run the Strip algorithm on the ciphertexts published on the bulletin board to verify the validity of the encryptions, and verify the uniqueness of each ciphertext inside the election. In case of non uniqueness, the most recent ballot is declared invalid and its sender notified.

The output of the Strip algorithm is made of standard ElGamal ciphertexts, which can then be sent to the mix servers.

It can be observed that, as soon as sk^A becomes public, everyone becomes able to execute the Strip algorithm and then becomes able to rerandomize ciphertexts. Therefore, no ballot encrypted with the $(pk^A : pk^B)$ public key can be accepted as soon as sk^A is disclosed. The submission security game [32] however requires that new augmentations $((pk^A)',(sk^A)')$ can be built from the same secret key sk^B without raising any issue. This would allow splitting an election into different phases, keeping the same ElGamal key during the whole process but updating the augmentation regularly (e.g., every day) in order to be able to perform ballot validity verifications before the end of the election.

Performance. As multiexponentiation algorithms are not available among the methods of the Java BigInteger class used in Helios, an encryption takes 5 modular exponentiations to a voter. As several hundred of candidates can typically be encoded into one single ciphertext, this provides a substantial gain compared to the current cost of 6 modular exponentiations per candidate in a Helios 3.1 election. We note however that the cost of these exponentiations is not the same: the exponential encoding of the voter choices used in Helios 3.1 allows working in small subgroups of $\mathbb{Z}_p^*$ (e.g., choosing $|p| = 2048$ and $|q| = 256$), while the message encoding technique that we are using here requires using $q = (p-1)/2$.

We used this scheme in an election during which we tallied 4488 votes, which we will discuss in the next section. For reasons presented there, we decided to search for another efficient SSA cryptosystem that would not need any augmented private key sk^A.

4.1.4 A solution based on the HTDH2 cryptosystem

Shoup and Gennaro [27] proposed the TDH2 cryptosystem, which is an efficient CCA2 threshold encryption scheme that offers security relying on the hardness of the DDH problem in the random oracle model. While the security guarantees offered for this scheme are weaker than those of the Cramer-Shoup scheme, we observe that it does not degrade the security model in which the rest of the cryptographic algorithms used in Helios offer security.

The TDH2 scheme however cannot be immediately used for our purpose, as it is based on the Hash-ElGamal cryptosystem, which is not homomorphic. More precisely, the basic ciphertexts have the form $(g^r, m \oplus H(h^r))$, where the hash function H is modeled as a random oracle. This scheme can however be modified to use standard ElGamal encryption instead, while remaining secure in the same security model.

We call that variant the HTDH2 scheme, where the initial “H” emphasizes the presence of a homomorphic embedded encryption scheme. As far as we know, this is the most efficient SSA (or IND-CCA2) cryptosystem that is based on a homomorphic basic cryptosystem, offers security under standard computational assumptions (DDH in our case) in the random oracle model, and does not require a private key augmentation sk^A.

We describe the HTDH2 scheme as follows.

1. On input 1^n, Gen^B selects integers p and q, z, generators g and g_0, and computes h as in the Cramer-Shoup scheme. A collision resistant hash function is also selected: $H : G^5 \times \{0,1\}^n \to \mathbb{Z}_q$. The output (pk^B,sk^B) of Gen^B is then defined as $((p,g,g_0,H,h),(p,z))$.

2. On input pk^B, Aug^B outputs is the pair $(pk^A, sk^A) = ((p,g,g_0),\perp)$.

3. On input $(pk^A : pk^B) := ((p,g,g_0),(p,g,g_0,H,h))$, $m \in \mathbb{Z}_q^*$, and $L \in \{0,1\}^n$, Enc selects a random pair $(r,s) \in \mathbb{Z}_q^2$ and computes the ciphertext $(L,u,e,u_0,c,f) := (L,g^r,m^2h^r,g_0^r,H(u,e,w,u_0,w_0,L),s+rc)$ where $w := g^s$ and $w_0 = g_0^s$.

4. On input $(pk^A, sk^A, tdh) := ((p,g,g_0),\perp, (L,u,e,u_0,c,f))$, Strip computes $w := g^f/u^c$ and $w_0 := g_0^f/u_0^c$ and returns (u,e) if $c = H(u,e,w,u_0,w_0,L)$ or $(\perp,\perp)$ otherwise.

5. Dec^B works as above.

The SSA security of this scheme is shown in Appendix B.

Using the HTDH2 scheme for ballot submission. The HTDH2 scheme can be used essentially in the same way as Cramer-Shoup. Important differences appear though, through the fact that no sk^A key is needed. Indeed, in that case, the Strip algorithm does not need any private input, with the implication that the validity of a ciphertext can be verified as soon as a ciphertext has been submitted and not when the ballot submission phase is complete. This simplifies the election workflow substantially, as will be discussed in the next section.

From a performance point of view, the cost of a HTDH2 encryption is similar to the one of a Cramer-Shoup encryption, being dominated by the 5 modular exponentiations.

We used this scheme in an election during which we tallied 3951 ballots, which we will also discuss in the next section.

4.2 Ballot shuffling

We use the proof of shuffle recently proposed by Terelius and Wikström [28, 33]. This choice was motivated by the conceptual simplicity of the proposed proofs (even though proofs of shuffle are remarkably complex algorithms), by the performance of their solution, and by the presumed absence of patents that could restrict the use of this solution.

The task of each mix server can be separated into two components.

1. Each mix server first selects a random permutation and publishes a commitment on that permutation, together with a proof that it knows an opening of the commitment that actually is a permutation.

2. Then, each mix server reencrypts and shuffles the input ciphertexts using the committed permutation, and provides a proof that the shuffle is consistent with that committed permutation.

The first step is the most computationally intensive one and is independent of the actual ballots to be shuffled. This suggests realizing it as part of a precomputation stage, in order to be able to complete the proof of shuffle much faster when the actual ciphertexts become available.

However, it requires the shuffling trustees to proceed in two steps, between which a secret permutation needs to be stored. As this would actually increase the complexity of the tallying procedure, we rather decided to not separate the two components of the proof, and perform them in one step.

Implementation. We do not provide a full description of this shuffle but instead refer the reader to the original papers. While being fairly complex, the proof of shuffle itself takes less than 200 lines of Python code.

Our implementation relies on the Python Cryptography Toolkit[1] for pooling secure randomness, and on the gmpy[2] wrapper for multiprecision arithmetic. Both are widely available Python extensions.

The efficiently of our code is reasonable for our purpose, as it allows shuffling around 25 ballots per second using a single processor thread on a standard laptop, using a prime modulus p of 2048 bits.

Verificatum. Wikström proposes a very complete and much more efficient implementation of the same algorithms as part of the Verificatum library [30]. This implementation has been written in Java, but also includes a Modular Exponentiation Extension of the GMP library (GMPMEE) that speeds up several operations in Verificatum.

While Verificatum would be extremely useful for large elections, we believe that our simple Python script offers more portability and simplicity, and is easier to audit, while being efficient enough for elections requiring the shuffle of a few thousand ciphertexts.

4.3 Comparison with Helios 1.0

Helios 1.0 [1] already implemented an election procedure using mixnets for tallying. The algorithms and election workflow chosen there had been mainly selected for simplicity and ease of education.

[1]`http://www.pycrypto.org/`
[2]`http://www.gmpy.org/`

In particular, in order to simplify the election procedure, the Helios server was in charge of all privacy related aspects: (i) it selected the ElGamal key pair used to encrypt the ballots; (ii) it was mixing the ballots; (iii) it was decrypting the mixed ballots.

This placed a large amount of trust in the voting server, but also avoided the questions of trustees that we had to take care of.

Besides, in order to make the proof of shuffle easier to explain, the Sako-Kilian [24] proof of shuffle was implemented. This solution is quite inefficient compared to the solution we adopted, and was reported to require around 20 seconds per ballot [1], using an older computer but also smaller security parameters ($|p| = 1024$.) This solution would have been unpractical for the elections we will describe now.

5 Experiences with mixnet-based elections

We experimented with the election procedures described in this paper during two elections, one using the Cramer-Shoup cryptosystem as described in Section 4.1.3 and the other using the HTDH2 cryptosystem described in Section 4.1.4. We now report on these two experiences.

5.1 Election description

Student elections. The two elections we organized were university student elections, with around 25000 potential voters each, with different races: all students were entitled to vote for representatives in the general student council, but also to vote for representatives to the faculty council in each of the faculty in which they are registered (student pursuing several degrees concurrently in different faculties were allowed to vote for the faculty council of each of these faculties).

The ballots of these races were particularly large: the candidates were organized in lists, some of these lists counting 127 candidates with up to 259 candidates on a single ballot. Voters were allowed to select as many candidates as they wanted as long as they selected them inside a single list. The size of the ballots was the technical motivation to use mixnet-based tallying in these elections.

Electronic and paper voting. The elections were organized in two steps. First, during four days (including a weekend), students were allowed to vote electronically. Then, during the next two days, standard paper voting took place. One of the key motivations for the use of electronic voting was to enable students in exchange programs or doing internships to vote without needing to be on campus. It was however decided to maintain paper voting days in order to be able to organize voting desks in the main hall of each faculty of the university, allowing to prompt walking students to vote immediately: paper voting enables a large number of votes to be collected in parallel at the same public place (e.g., at the end of a class) without needing to install a large number of computers.

In order to prevent double voting, voter lists were printed at the end of the electronic voting phase with a clear mark indicating who voted already.

We actually decided to take benefit of these paper voting days as a period of audit for the bulletin board: the students who voted electronically were allowed to request to vote by paper by submitting a signed document demanding the cancellation of their electronic vote. This was made possible thanks to the fact that Helios keeps track, through the web bulletin board, of the link between the identity of the voters and their encrypted vote. So, cancellation of electronic votes was made visible to everyone on the bulletin board, but was done only when the election commission held a document proving that the cancellation was performed by request of the voter.

Implementation architecture. The electronic voting system was hosted on a standard Debian server inside the university infrastructure. The computational load remained extremely limited on the server during the whole election.

Authentication of the voters was performed using the standard login and passwords used by the students to access the university web portal, as well as with their student card number.

The election URL was widely advertised through different channels (other websites, emails, public display on the campus, . . .) and the voting server authentication was strengthened through DNSSEC. The security of the server itself was improved through various security hardening techniques, and no substantial hacking attempt has been detected during the elections.

Tallying procedure The task of shuffling the ballots was devoted to three shuffling trustees. Besides, as for other elections [2], six decryption trustees were designated to generate and hold the private election keys eventually used to decrypt the shuffled ballots. Three keys were generated independently of each other and held by two trustees each. This procedure was preferred to a fully threshold procedure as it simplified the organisational matters: each pair of trustee was allowed to produce their key at any time and in one step, while a threshold procedure would require more than one round and some level of synchronization.

After verification of the validity and uniqueness of all ciphertexts, the ballots were gathered on the voting server in a single tarball. The shuffling trustee then, sequentially, downloaded the ballots, computed the verifiable shuffle, and uploaded the resulting ballots and proofs to the voting server. Each shuffling trustee verified the proofs provided by the other trustees.

The decryption trustees proceeded in a similar way: they each downloaded the tarball containing the last shuffled ballots, computed the decryption and decryption proofs locally, and uploaded the result on the voting server. That phase was much faster than the previous one, as the computational load was smaller and, more importantly, because all the work was performed in parallel without any synchronization.

The decryption proofs were eventually verified and the election outcome computed and published. That outcome was also transmitted to the team in charge of coordinating the paper tally for integration with their results.

5.2 Election statistics

The following number of votes were tallied during the two elections:

	Election 1	Election 2
Electronic	4488	3951
Paper	2564	3016

The difference in the repartition of the votes between the two polling mechanisms in the two elections seemed to follow the level of advertising activity exercised by the students organizing the election. A graph displaying the evolution of the number of collected electronic votes with time is shown in Figure 1. The strong increase in the voting rate that followed the advertising performed during the last 8 hours of the electronic election can be observed easily.

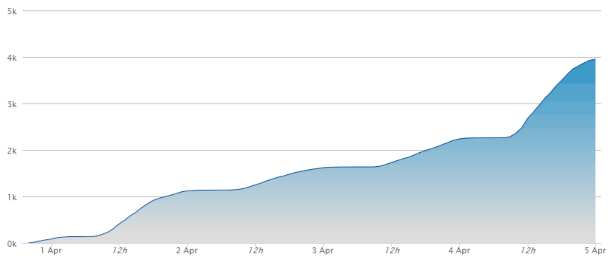

Figure 1: Evolution of the number of votes submitted during the four electronic voting days of the second election.

Very few voters requested the cancellation of their electronic vote: 4 ballots were canceled during the second election. Each time, the motivation of the voter was to be able to change his/her mind about the candidates (s)he wanted to support rather than the conviction of a failure of the voting system.

During the two elections, all submitted ballots were declared valid, that is, the Strip algorithm always succeeded, no duplicate ciphertexts were submitted, and the decrypted ballots were all valid. This contrasts with the paper elections, during which a few dozen of ballots were declared invalid due to violation of the ballot filling rules.

5.3 Lessons from election days

Mixnets vs. homomorphic tallying. Running a mixnet-based tallying showed to be remarkably more challenging than computing a homomorphic tally, which can be expected from the comparison of the two tallying procedures.

The tallying procedure of both elections took a few hours, even though the tallying of the second election went much faster for reasons we describe below. Most of this time was actually not used by computation, as we expected, but by organizational matters: setting up the trustee's machines, transferring data in a reliable way, making sure to not break the confidentiality of the keys, . . .

It would be highly desirable to make the tallying procedure faster, as its length increases the risks of errors in the manipulation of sensitive data, but also because voters expect that an electronic tally, being automated, would require a matter of seconds.

Cramer-Shoup vs. HTDH2. The motivation for the adoption of the HTDH2 cryptosystem rather than Cramer-Shoup in the second election was to simplify the audit and tallying procedures.

At first, it allowed verifying the validity and independence of the ballots as soon as they were submitted and not after the release of private key augmentation sk^A. This is much more convenient as it allows resolving any potential conflict on that matter immediately during the ballot submission phase rather than during the tallying phase.

It also removed the steps needed to open and publish the shares of sk^A held by the trustees, which were needed to check the validity of the ballots. Simplifying the role of the trustees showed to be very useful too.

Student survey. Students were invited to complete an online survey after the election. 158 responses were collected.

97% of the responses came from students who voted, and 95% of them actually voted electronically. We suppose that this bias comes from the form of the survey: we can imagine that students accepting to fill an electronic survey are more likely to also be willing to fill a ballot electronically.

58% of the responders report having verified the presence of their ballot on the web bulletin board. Among them, 89% did so immediately after submitting their ballot, while only 7% did another verification later. While this second proportion is much lower, it still appears to be high enough to provide strong confidence in the correctness of the bulletin board.

On asking whether an electronic only voting solution, a paper only voting solution, or a mixed voting solution should be kept, 23% of the responses supported electronic only voting, 1% supported paper only voting, and 77% supported the mixed solution. This last solution is the one the student committees decided to keep for the next years.

6 Conclusion

Mixnet-based tallying shows to be very useful for the organization of elections as soon as the number of candidates increases, or when the ballot filling rules become too esoteric to enable efficient zero-knowledge validity proofs.

Our experiences convinced us however that it is desirable to stick to homomorphic tallying techniques as long as they are feasible, as a mixnet-based tallying is substantially more complex and delicate to organize properly.

As part of an ongoing effort to improve the modularity of Helios, which started with the 3.1 release, we plan to integrate our mixnet workflow and algorithms into the Helios trunk, which would then offer the possibility to choose between homomorphic or mixnet-based tallying according to the specific needs of each election.

In order to be able to benefit of the performances of the Verificatum library, we also plan to build an interface that will allow converting the format of our Helios-style ElGamal ciphertexts into a data format that could be managed by Verificatum. This would also provide two independent implementations of the same proof of shuffle, relying on mostly independent software stacks (up to the GMP library).

Acknowledgements

We thank Ben Adida, Benoît Libert, and Douglas Wikström for many interesting interactions on the integration of mixnet-based election into Helios, on the TDH2 cryptosystem, and on the proof of shuffle in the Verificatum library.

We are also very grateful to David Bernhard, Véronique Cortier, Ben Smyth, and Bogdan Warinschi for our discussions on ballot independence and techniques to enforce it.

We really thank the anonymous EVT/WOTE referees for their numerous comments and sugguestions, which helped improving this document.

This work was partially supported by the Interuniversity Attraction Pole P6/26 BCRYPT. Olivier Pereira is a research associate of the Belgian Funds for Scientific Research F.R.S.–FNRS.

References

[1] Adida, B. Helios: web-based open-audit voting. In *SS'08: Proceedings of the 17th conference on Security symposium* (Berkeley, CA, USA, 2008), USENIX Association, pp. 335–348.

[2] Adida, B., de Marneffe, O., Pereira, O., and 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* (Aug. 2009), T. M. D. Jefferson, J.L. Hall, Ed., Usenix.

[3] Bellare, M., and Rogaway, P. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security* (1993), pp. 62–73.

[4] Ben-Or, M., and Linial, N. Collective coin flipping, robust voting schemes and minima of banzhaf values. In *26th Annual Symposium on Foundations of Computer Science* (1985), IEEE, pp. 408–416.

[5] Benaloh, J. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, jan 1987.

[6] Benaloh, J. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop* (Berkeley, CA, USA, 2006), USENIX Association.

[7] Chaum, D. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM 24*, 2 (1981), 84–88.

[8] Cohen, J. D., and Fischer, M. J. A robust and verifiable cryptographically secure election scheme (extended abstract). In *26th Annual Symposium on Foundations of Computer Science* (1985), IEEE, pp. 372–382.

[9] Cortier, V., and Smyth, B. Attacking and fixing helios: An analysis of ballot secrecy. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium (CSF'11)* (jun 2011), IEEE.

[10] Cramer, R., Gennaro, R., and Schoenmakers, B. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology - EUROCRYPT '97* (1997), W. Fumy, Ed., vol. 1233 of *Lecture Notes in Computer Science*, Springer, pp. 103–118.

[11] Cramer, R., and Shoup, V. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology - CRYPTO '98* (1998), H. Krawczyk, Ed., vol. 1462 of *Lecture Notes in Computer Science*, Springer, pp. 13–25.

[12] de Marneffe, O., Pereira, O., and Quisquater, J.-J. Simulation-based analysis of e2e voting systems. In *Proceedings of the First Conference on E-Voting and Identity (VOTE-ID 2007)* (Oct. 2007), A. Alkasar and M. Volkamer, Eds., no. 4896 in LNCS, Springer, pp. 137–149.

[13] Estehghari, S., and Desmedt, Y. Exploiting the client vulnerabilities in internet e-voting systems: Hacking helios 2.0 as an example. In *Electronic Voting Technology Workshop/ Workshop on Trustworthy Elections (EVT/WOTE '10)* (2010), D. Jones, J.-J. Quisquater, and E. Rescorla, Eds., USENIX.

[14] Recommendation rec(2004)11 of the committee of ministers to member states on legal, operational and technical standards for e-voting. `https://wcd.coe.int/wcd/ViewDoc.jsp?id=778189&Site=CM`.

[15] Fiat, A., and Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86* (1986), A. M. Odlyzko, Ed., vol. 263 of *Lecture Notes in Computer Science*, Springer, pp. 186–194.

[16] Fouque, P.-A., and Pointcheval, D. Threshold cryptosystems secure against chosen-ciphertext attacks. In *Advances in Cryptology - ASIACRYPT 2001* (2001), C. Boyd, Ed., vol. 2248 of *Lecture Notes in Computer Science*, Springer, pp. 351–368.

[17] Groth, J. Evaluating security of voting schemes in the universal composability framework. In *Applied Cryptography and Network Security, ACNS* (2004), M. Jakobsson, M. Yung, and J. Zhou, Eds., vol. 3089 of *Lecture Notes in Computer Science*, Springer, pp. 46–60.

[18] Groth, J. Non-interactive zero-knowledge arguments for voting. In *Applied Cryptography and Network Security* (2005), J. Ioannidis, A. D. Keromytis, and M. Yung, Eds., vol. 3531 of *LNCS*, pp. 467–482.

[19] Groth, J. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology 23*, 4 (2010), 546–579.

[20] Hirt, M. *Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting*. PhD thesis, ETH Zurich, Sept. 2001. Reprint as vol. 3 of *ETH Series in Information Security and Cryptography*, ISBN 3-89649-747-2, Hartung-Gorre Verlag, Konstanz, 2001.

[21] Naor, M., and Yung, M. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing* (1990), ACM, pp. 427–437.

[22] Park, C., Itoh, K., and Kurosawa, K. Efficient anonymous channel and all/nothing election scheme. In *Advances in Cryptology - EUROCRYPT '93* (1994), T. Helleseth, Ed., vol. 765 of *Lecture Notes in Computer Science*, Springer, pp. 248–259.

[23] Pfitzmann, B., and Pfitzmann, A. How to break the direct rsa-implementation of mixes. In *Advances in Cryptology - EUROCRYPT '89* (1989), J.-J. Quisquater and J. Vandewalle, Eds., vol. 434 of *Lecture Notes in Computer Science*, Springer, pp. 373–381.

[24] Sako, K., and Kilian, J. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *Advances in Cryptology - EUROCRYPT '95* (1995), L. C. Guillou and J.-J. Quisquater, Eds., vol. 921 of *LNCS*, Springer, pp. 393–403.

[25] Schnorr, C.-P. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPTO '89* (1989), G. Brassard, Ed., vol. 435 of *Lecture Notes in Computer Science*, Springer, pp. 239–252.

[26] Schnorr, C.-P., and Jakobsson, M. Security of signed elgamal encryption. In *Advances in Cryptology - ASIACRYPT 2000* (2000), T. Okamoto, Ed., vol. 1976 of *Lecture Notes in Computer Science*, Springer, pp. 73–89.

[27] Shoup, V., and Gennaro, R. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptology 15*, 2 (2002), 75–96.

[28] Terelius, B., and Wikström, D. Proofs of restricted shuffles. In *Progress in Cryptology - AFRICACRYPT 2010* (2010), D. J. Bernstein and T. Lange, Eds., vol. 6055 of *Lecture Notes in Computer Science*, Springer, pp. 100–113.

[29] Tsiounis, Y., and Yung, M. On the security of elgamal based encryption. In *Public Key Cryptography, First International Workshop on Practice and Theory in Public Key Cryptography, PKC '98* (1998), H. Imai and Y. Zheng, Eds., vol. 1431 of *Lecture Notes in Computer Science*, Springer, pp. 117–134.

[30] Wikström, D. Verificatum. `http://www.verificatum.com`.

[31] Wikström, D. A universally composable mix-net. In *Theory of Cryptography* (2004), M. Naor, Ed., vol. 2951 of *Lecture Notes in Computer Science*, Springer, pp. 317–335.

[32] Wikström, D. Simplified submission of inputs to protocols. In *Security and Cryptography for Networks, 6th International Conference* (2008), R. Ostrovsky, R. D. Prisco, and I. Visconti, Eds., vol. 5229 of *Lecture Notes in Computer Science*, Springer, pp. 293–308.

[33] Wikström, D. A commitment-consistent proof of a shuffle. In *Information Security and Privacy – ACISP* (2009), C. Boyd and J. M. G. Nieto, Eds., vol. 5594 of *Lecture Notes in Computer Science*, Springer, pp. 407–421.

[34] Wolkamer, M. *Evaluation of Electronic Voting*, vol. 30 of *Lecture Notes in Business Information Processing*. Springer, 2009.

A Submission Security

We provide the submission security definition from Wikström [32].

In the submission security game, the adversary is given a public key pk^B of the basic cryptosystem. It can request that the experiment generates a fresh augmentation $(pk_j^A, sk_j^A) = \mathsf{Aug}(pk^B)$, stores $(pk_j, sk_j) = ((pk_j^A : pk^B), (sk_j^A, sk^B))$, and returns $pk_j = (pk_j^A : pk^B)$ to the adversary. This is done by submitting the integer j to its $pk_{(\cdot)}^A$-oracle. Any subsequent identical queries on j give the same pk_j. It can ask decryption queries. This is done by submitting an index and ciphertext pair (j, c) to its $\mathsf{Dec}_{sk_{(\cdot)}}(\cdot)$-oracle. It can request that the experiment reveals an augmentation sk_j^A by submitting j to its $sk_{(\cdot)}^A$-oracle, but after such a query no more decryption queries of the form (j, c) for some ciphertext c are allowed. Then the adversary must chose an index i and two challenge messages m_0 and m_1. The game is parameterized by a bit b and returns a challenge ciphertext of the form $c' = \mathsf{Enc}_{pk_i}(m_b)$. The adversary is then again allowed to: ask for more fresh public keys, ask more decryption queries with the exception of decryption of (i, c'), and request more augmentations. Finally, it outputs a guess b' of b. If the cryptosystem is submission secure, then the difference in distributions of b' with $b = 0$ or $b = 1$ respectively should be negligible.

This game is summarized below.

Submission Security Experiment $\mathsf{Exp}^{\mathsf{sub}-b}_{\mathsf{CS},\mathsf{CS}^B,A}(n)$.

$$\begin{array}{rcl}
(pk^B, sk^B) & \leftarrow & \mathsf{Gen}^B(1^n) \\
(pk^A_j, sk^A_j) & \leftarrow & \mathsf{Aug}(pk^B) \text{ for } j = 1,2,3,\ldots \\
(pk_j, sk_j) & \leftarrow & ((pk^A_j : pk^B), (sk^A_j : sk^B)) \\
(i, m_0, m_1, st) & \leftarrow & A^{pk^A_{(\cdot)}, sk^A_{(\cdot)}, \mathsf{Dec}_{sk_{(\cdot)}}(\cdot)}(\mathsf{choose}, pk^B) \\
c' & \leftarrow & \mathsf{Enc}_{pk_i}(m_b) \\
d & \leftarrow & A^{pk^A_{(\cdot)}, sk^A_{(\cdot)}, \mathsf{Dec}_{sk_{(\cdot)}}(\cdot)}(\mathsf{guess}, st)
\end{array}$$

Definition 2 (Submission Security)**.** *An augmentation CS of* CS^B *is said to be submission secure if* $\forall A \in \mathrm{PT}^*$*:* $|\Pr[\mathsf{Exp}^{\mathsf{sub}-0}_{\mathsf{CS},\mathsf{CS}^B,A}(n) = 1] - \Pr[\mathsf{Exp}^{\mathsf{sub}-1}_{\mathsf{CS},\mathsf{CS}^B,A}(n) = 1]|$ *is negligible.*

B Security of the HTDH2 scheme

The submission security of the HTDH2 cryptosystem follows from its IND-CCA2 security, as its augmentation does not involve any secret value.

Therefore, we show the following result.

Theorem 1. *In the random oracle model, the HTDH2 augmented cryptosystem is IND-CCA2 secure assuming that the DDH problem is hard in the underlying group G.*

Proof. The proof is organized as a sequence of games, the first of which being the IND-CCA2 game, and the last being such that the adversary is not left with any advantage.

GAME 1. This is the traditional IND-CCA2 game. The adversary Adv receives a HTDH2 public key and can query a decryption oracle. It then makes a challenge query including a label l' and two messages m_0, m_1. The challenger answers this query with a challenge ciphertext $(l', u', e', u'_0, c', f')$ that is an encryption of m_b, where b is a random bit chosen by the challenger. Adv can keep accessing the decryption oracle, as long as he does not query it on the challenge ciphertext. Eventually, Adv outputs a bit b'. The advantage of this adversary is then defined as $|\Pr[b = b'] - 1/2|$. We show that this quantity is negligible by defining a sequence of other games such that the advantage of Adv between two consecutive games only changes by a negligible quantity and the advantage of Adv in the last game is 0.

GAME 2. This game is identical to the previous one except that the experiment is aborted if Adv makes a random oracle query on a message that contains u' before the challenge query happened.

Since u' is independent of Adv's view until the challenge query happened, this restriction can only make a difference if Adv guesses u', which happens with negligible probability.

GAME 3. Observing that c' and f' form a standard NIZK proof that $\log_g(u) = \log_{g_0}(u_0)$, we define this game to be identical to the previous one, except that the elements c' and f' are now computed by running the proof simulator and patching the random oracle H adequately, that is, c' and f' are chosen at random and H is patched in such a way that $H(u', e', g^{f'}/u^{c'}, u'_0, g_0^{f'}/u_0^{c'}, l') = c'$. This patch can always be done thanks to the restriction introduced in the previous game, and the resulting view of Adv is indistinguishable of the previous one thanks to the ZK property of the DL equality proof.

GAME 4. This game is identical to the previous one, except that, in the key generation procedure, the challenger chooses a random value $t \in \mathbb{Z}_q$ and sets $g_0 = h^t$ instead of choosing g_0 as a random generator of G independent of g.

This change does not make any difference in the view of Adv.

GAME 5. This game is identical to the previous one, except for the following change in the way the challenger answers decryption queries:

a. On decryption queries on ciphertexts that contain the elements (u', u'_0), answer as per the specification of the decryption algorithm.
b. On decryption queries on ciphertexts that do not contain these elements, compute m' as $e/u_0^{1/t}$ instead of e/u^z.

This does not make any difference for Adv, except if he is able to produce a fake DL equality proof by himself, which would contradict the soundness of the proof.

GAME 6. This game is identical to the previous one, except that a random element $s \in \mathbb{Z}_q$ is now selected, and e' is computed as $m_b^2 h^s$ while u'_0 is computed as g_0^s.

We show that an adversary who would have an advantage in distinguishing this game from the previous one would have the same advantage in solving the DDH problem in G.

To this purpose, we build a DDH distinguisher that receives a challenge (α, β, γ) with respect to base g (that is, $\alpha = g^a$, $\beta = g^b$ and γ is either equal to g^{ab} or is a random group element), which he uses as follows: (i) h is set to α, (ii) u' is set to β, (iii) e' is set to $m_b^2\gamma$, (iv) u'_0 is set to γ^t, and (v) all other variables are computed as in Game 5. It can now be observed that, if (α, β, γ) is a DH triple, then the view of Adv is exactly as in Game 5 while, if (α, β, γ) is a random triple, the view of Adv is exactly as in Game 6. As a result, an adversary who can make a difference between these two games is also able to solve the DDH problem with the same probability.

To sum up, the challenge ciphertext now looks as: $(l', u' = g^r, e' = m_b^2 h^s, u'_0 = g_0^s, c', f')$, where r, s, c' and f'

are chosen at random. We can also observe that the relation $\log_g(u') = \log_{g_0}(u'_0)$ does not hold in general anymore, even though Adv can verify that the corresponding proof still holds (this comes from the change in Game 3).

GAME 7. This game is identical to the previous one, except that the experiment now aborts if A makes a decryption query on a ciphertext (L, u, e, u_0, c, f) such that $u = u'$, $u_0 = u'_0$ and the ciphertext validity proof checks.

We show that this change will actually cause an abort if the adversary is able to solve the DL problem in G. To this purpose, we make the following observations:

(i) The decryption query must have been made after the challenge query, or the experiment would have been aborted as in Game 2.

(ii) $(l, e, c, f) \neq (l', e', c', f')$, or the query would be invalid.

(iii) If $(c, f) = (c', f')$ then $e \neq e'$ or $l \neq l'$ and the adversary must therefore have been able to forge a proof that $\log_g(u') = \log_{g_0}(u'_0)$ even though this relation does not hold with overwhelming probability, and using a random oracle query different (thanks to e or l) from the one resulting from the computation of the challenge ciphertext. The probability that the random oracle query provides an answer that makes a valid proof is $1/q$.

(iv) If $(c, f) \neq (c', f')$, then $(w, w_0) \neq (w', w'_0)$. Indeed, if any of the two elements of the pair are equal (say, $w = w'$), then Adv could be used to compute two pairs (c, f) and (c', f') such that $g^f u^{-c} = g^{f'} u^{-c'}$, from which one can compute $\log_g(u) = (f - f')/(c - c')$. Now, the differences of the w values allows using the same argument as above.

GAME 8. This game is the same as the previous one, except that the challenger now computes e' as a random group element. It is clear that, in this case, the advantage of Adv must be null as its view is independent of the bit b.

We again show that an adversary who would have an advantage in distinguishing this game from the previous one would have the same advantage in solving the DDH problem in G.

To this purpose, we build a DDH distinguisher that receives a challenge (α, β, γ) with respect to base g, which he uses as follows: (i) a random element $t \in \mathbb{Z}_q$ is selected, (ii) g_0 is set to α^t, (iii) h is set to g^t, (iv) e' is set to $m_b^2 \beta$, (v) u is selected at random, (vi) u'_0 is set to γ, (vii) all other variables are computed as in Game 5. This distinguisher is able to answer all decryption queries by using the secret t, as in the original scheme. This makes a difference with the previous game only if he has to answer a decryption query containing an invalid DL equality proof, which is ruled out by the proof soundness property and the abort condition from the previous game.

We now observe that, if (α, β, γ) is a DH triple, then the view of Adv is exactly as in Game 7 and, if (α, β, γ) is a random triple, then the view of Adv is exactly as in Game 8 since β becomes an independent random group element.

□