Cryptographic Voting — A Gentle Introduction

David Bernhard and Bogdan Warinschi

University of Bristol, England

Abstract. These lecture notes survey some of the main ideas and techniques used in cryptographic voting systems. The write-up is geared towards readers with little knowledge of cryptography and it focuses on the broad principles that guide the design and analysis of cryptographic systems, especially the need for properly designed security models.
We use a system proposed by Fujioka, Okamoto and Ohta as starting example to introduce some basic building blocks and desirable security properties. We then slowly build towards a comprehensive description of the Helios voting system, one of the few systems deployed in practice and briefly discuss a few of its security properties.

1 Introduction

A potential solution to the problem of decrease turn-out in elections is the use of remote voting systems. The increased comfort that permits voters to cast their ballot from anywhere and at any time (as long as they have an internet connection) should translate in higher voter participation and even reduce the costs of running elections. Just like with any advanced technology, bringing electronic voting[1] to widespread practice faces both technological and societal challenges.

The latter is well illustrated by Norway's recent decision, that follows a couple of trials, not to expand internet voting to nationwide elections because of the impact that public's trust in this technology may have on the democratic process. Related problems raised questions regarding the constitutionality of internet voting in Germany. While the deployment and usability issues (e.g. dealing with malware on user's voting device, overcoming social inertia, gaining public trust) still need to be solved, it is fair to say that the cryptographic technology that goes into electronic voting is by now in a reasonably stable condition. In particular existing schemes build upon a powerful and versatile toolbox of cryptographic techniques which are combined to achieve the desired functionality and security.

These notes survey some of the most commonly used cryptographic primitives used in electronic voting. We explain and motivate the functionality and security they achieve through their use within electronic voting schemes. In this context, perhaps the most important message we wish to convey is the crucial

[1] We remark that in this write-up we use the term electronic voting to refer to online voting schemes. The term is also used to refer to kiosk-voting that involves some form of digital recording machine, potentially combined with other mechanisms like optical scanners, paper-based recording, etc.

role that rigorous models play in the development of secure systems. For protocols with complex security requirements, as is the case of electronic voting, models are necessary for understanding the security these protocols offer and for enabling rigorous security proofs. Most of the discussion is driven by two voting schemes from the literature, one of which is in current use. We aim to provide a comprehensive overview of the latter.

These notes are intended for readers with little to no knowledge of cryptography, but who wish to understand some of the underpinnings of cryptographic voting. We assume some familiarity with basic abstract algebra and ease in using mathematics.

Primitives and models. One of the main goals of this write-up is to present the building blocks used in voting systems. Our presentation is not exhaustive but it includes some of the most used primitives: digital and blind signatures, homomorphic encryption, threshold techniques and zero-knowledge proofs. For each of the primitives we present a rigorous security model where we clarify what is an adversary, how it can interact with the primitive, and what constitutes a breach against the primitive. To ease understanding, we find it useful to define the functionality and security of these primitives by analogy with systems which achieve similar levels of security in the physical world. In addition, for increased clarity but at the expense of some precision, we chose not to give the typical fully formal cryptographic definitions.

We remark that, perhaps surprisingly, not all of the (many) security properties that voting systems should satisfy have formal definitions. Indeed, most of the recent research on cryptographic voting focuses on the design of such models and the analysis of existing schemes. In these notes we recall ballot privacy, a security notion that captures the idea that users' votes stay private. In addition, we briefly comment on the equally important property of verifiability.

Instantiations and Helios. For all of the primitives that we cover we show how they can be (securely) instantiated; each instantiation is preceded by the necessary mathematical background. In particular, we show how to realize all of the primitives that go into the design of the Helios voting systems, one of the few voting systems that are actually used in practice. We justify the design of this system via an incremental construction. We start with a basic voting system which is functional but not secure, identify obvious attacks which we then defend against by using appropriate cryptographic techniques. The resulting design together with the instantiations of the different primitives that we present yield a scheme which has ballot privacy. We briefly discuss its verifiability properties.

Conventions. To allow the reader to zoom-in on the parts that may be of more interest to him/her we separate and clearly indicate the following. Sections giving mathematical background information are marked with a frame and a large Σ. Sections that introduce basic cryptographic primitives are marked with a lock symbol (🔒). Sections giving applications specific to the Helios voting scheme are marked with a ☼ symbol.

2 First steps

2.1 Example

Here is one way to run a poll. Voters enter a polling station and pick up a voting card on which the candidates standing for election or choices in a referendum are printed. They fill in their card by placing crosses in boxes. Then they take their card and put it in an opaque envelope which they seal. In keeping with the cryptographic constructions we will describe later, we call such an envelope containing a filled in vote card a "ballot". This completes the first step, ballot creation.

To cast their ballots, voters present them to an official along with some identification. The official checks that the voter is registered at this polling station and has not cast a vote yet, but the official does not get to see the vote itself. Then the official places the ballot-envelope in a stamping machine and stamps the envelope, in such a way that the imprint of the stamp is not only visible on the envelope but also transferred to the vote card within.

Voters post their ballots to a counting centre. The postal service agrees to send any correctly stamped envelope free of charge from anywhere in the country so voters can post their envelope anonymously in any post box that they choose. The counting centre shuffles all received envelopes, opens them and counts all vote cards that contain an imprint of the official stamp.

2.2 🔒 Digital signatures

We will now start to develop the tools that we will use to build a cryptographic version of the protocol sketched in the last section. Along the way we will introduce the cryptographer's method of defining and reasoning about security of schemes and protocols.

Digital signatures are the cryptographer's replacement for signatures or stamps. If we know what someone's signature looks like and believe that it would be hard for anyone but the owner to produce such a signature, the presence of such a signature on a document attests that the owner has seen and signed it. Similarly, the imprint of a stamp on a document attests that someone with the appropriate stamp has stamped the document — although as we will see soon this does not have to mean that the stamp-owner has seen the document.

Digital signatures differ from physical ones in that they are not placed on an original document, modifying the original, but are separate objects that can be provided alongside the original. As a consequence, to prevent someone from transferring a signature from one document to another, digital signatures for different documents will be completely different objects.

We follow the cryptographic convention of first defining a class of schemes (that is, digital signature schemes) and then, in a later step, defining what we mean when we say that a member of this class is "secure". Keeping functionality and security separate has many advantages including that we can reason about several different levels of security for the same class of schemes. We will give some examples of this relating to signature schemes in particular.

To be able to create digital signatures, a signer first has to generate a pair of keys called the signing key (or secret key) and verification key (or public key). To do this, a digital signature scheme defines a key generation algorithm. The signing key is like a stamp with which the signer can stamp documents. Such a stamp on a document does not mean much on its own (anyone can create their own stamps) but if you know what a particular person's or organisation's stamp looks like, you can verify any stamped document to see if it was really stamped by the person or organisation you know, by comparing the imprint on the document with the imprint you know to be theirs. The verification key plays a similar role for digital signatures.

A digital signature scheme comes with two more algorithms. The signing algorithm takes a document and a signing key as input and returns a signature for the document. The verification algorithm takes a document, a signature and a verification key and outputs 1 if the signature is valid for the given key and document, otherwise 0.

It is the signer's responsibility that all verifiers have an authentic copy of the verification key. For example, in some government e-ID card schemes every citizen gets a smartcard containing a signing key and the government maintains a public database of verification keys. For a digital election, if the election authorities need to sign ballots they can publish their verification key as part of the election specification.

Definition 1. *A digital signature scheme Σ is a triple of algorithms*

$$\Sigma = (\texttt{KeyGen}, \texttt{Sign}, \texttt{Verify})$$

known as the key generation, signing and verification algorithms and satisfying the correctness condition below.

The key generation algorithm takes no input and produces a pair of keys $(sk, vk) \leftarrow \texttt{KeyGen}()$ known as the signing and verification keys. The signing algorithm takes a signing key sk and a message m as inputs and produces a signature $\sigma \leftarrow \texttt{Sign}(sk, m)$. The verification algorithm must be deterministic. It takes a verification key vk, a message m and a signature σ as inputs and returns 0 or 1. We say that σ is a (valid) signature for m under key vk if $\texttt{Verify}(vk, m, \sigma) = 1$.

A digital signature scheme must satisfy the following correctness condition which means that correctly generated signatures are always valid. For any message m, if you run the following sequence of algorithms then you get $b = 1$:

$$(sk, vk) \leftarrow \texttt{KeyGen}();\ \ \sigma \leftarrow \texttt{Sign}(sk, m);\ \ b \leftarrow \texttt{Verify}(vk, m, \sigma)$$

We will present a concrete digital signature scheme later in this work when we have developed the necessary mathematics to motivate it. For now, we briefly change our focus to talk about security notions and models.

2.3 Security models

We introduce the cryptographer's viewpoint of security using digital signatures as an example. Security means that an certain kind of attacker can not do certain things, like create a signature on a document that verifies under someone else's key.

Cryptographic games. The core of a security notion, at least in this work, is a cryptographic game. A game formalises two main aspects of a notion. First, it defines exactly what we want an attacker not to be able to do: a scheme will be called secure (w.r.t. a notion or game) if we can show that no attacker can win the given game. Secondly, a game specifies what we assume the attacker can do, by giving a set of moves allowed in the game and conditions on when and how often the attacker can use them.

Security games are defined in three parts. First, the game begins with some setup algorithm. Secondly, we give one or more moves that the attacker can play in the game. Finally, we state the winning conditions for the attacker.

For example, the two security notions for digital signatures that we use in this work both start by having the game playing the role of a signer and creating a signature key pair. They also both end by saying that the attacker wins if she can forge a signature but they differ in what other signatures the attacker may legitimately obtain: the security notion for "no-message" attackers considers attackers that never see any valid signatures whereas "chosen message" attackers may ask the signer to sign any message of their choice and win if they can forge a signature on a message that was never signed by the signer.

Cryptographers use two kinds of security games. The first, which could be called "trace games", are games in which the attacker wins if she does something that should be impossible in a secure system (like obtain someone's secret key or forge a signature). Here, the security definition calls a scheme secure if no attacker can win the game. The second type of game is the indistinguishability game where the attacker is asked to guess which of two things the game did. In an indistinguishability game, the attacker can always make a guess at random so the security definition says that a scheme is secure if no attacker can win the game with more than the probability 1/2 of guessing at random. It will always be clear from our description of games and their winning conditions which type of game is meant.

From games to security notions. The second main ingredient in a security notion is the definition of the resources available to a hypothetical attacker. These resources are composed of two factors: first, the moves available to the attacker in the game and secondly, the computational resources that the attacker can use "during her turns". Thus, the difference between a security game and a security notion is that a game specifies an interface with which the attacker can interact but says nothing about the attacker herself whereas a security notion describes both a game and a class of attackers, usually in a statement of the form “no attacker of a given class can win the given game (with more than a certain probability)”.

There are two principal classes of attackers. The first are computationally unbounded attackers who may have unlimited resources; only a very small number of cryptographic constructions can be secured against unbounded attackers. This does not include digital signatures or indeed any scheme using a fixed-length secret key — an unbounded attacker can always break such schemes by trying all possible keys. Commitment schemes which we will introduce later can however be made secure even if one of the two players involved has unbounded resources.

The second class of attackers is that of polynomially bounded attackers, commonly called efficient attackers. This class follows the notion of efficiency from complexity theory: an algorithm taking a bitstring s as input if there is some polynomial $p(x)$ such that on input a string s, the algorithm completes in at most $p(|s|)$ steps where $|s|$ is the length of s in bits. This allows us to introduce cryptographic keys since an n-bit key can be chosen in 2^n possible ways and 2^n grows much faster than any polynomial in n.

A fully formal treatment of this approach, which can be called asymptotic security, gets complex very quickly. We cannot talk about the security of any one fixed scheme but instead have to reason about families of schemes indexed by a so-called security parameter, which can very informally be thought of as the bit-length of keys in a scheme. Further, an asymptotic security notion typically says that the attacker's probability of winning the game is smaller than the inverse of any polynomial in the security parameter, what complexity theorists would call a negligible function[2].

In this work, we largely sweep such considerations under the carpet in favour of a more readable (we hope) introduction to the concepts and high-level connections that make up a cryptographic voting scheme. For the same reason we omit all formal security proofs which we could not present without fully formal definitions.

2.4 Security of digital signatures

An obvious property that signatures should have is that you cannot forge a signature on a message that verifies under someone else's key. We call such a forgery an existential forgery and we call an attacker that produces such a forgery a no-message attacker (we will see why in a moment). The security game and notion for this property have the game create a key pair and give the adversary the verification key, which is supposed to be public. The adversary wins if she produces a forgery:

Definition 2. *A digital signature scheme is existentially unforgeable under no-message attacks (EF-NMA) if no attacker can win the following game.*

[2] A negligible quantity is not the same thing as an exponentially small one like 2^{-n}, but an exponentially small quantity is always negligible.

Setup *The game creates a key pair* $(sk, vk) \leftarrow \texttt{KeyGen}()$ *and saves them; the attacker gets the verification key* vk*.*

Moves *None in this game.*

Winning conditions *The attacker wins the game if she provides a message/signature pair* (m^*, sk^*) *such that this pair verifies under the game's key:* $\texttt{Verify}(vk, m^*, \sigma^*) = 1$*.*

This definition is considered necessary but not sufficient. The attacker may be a participant in some system using digital signatures in which she gets to see messages legitimately signed by some other person; she should still not be able to forge anyone else's signature on any message they did not sign. This includes such attacks as taking a signature off one message and claiming that the signer actually signed some other message. Cryptographers model this with the chosen-message attack game. Here the adversary gets an extra move: she may ask the game to sign any messages of her choice and wins if she can forge a signature on any message that the game did not sign.

Definition 3. *A digital signature scheme is existentially unforgeable under chosen message attacks (EF-CMA) if no attacker can win the following game.*

Setup *The game creates a key pair* $(sk, vk) \leftarrow \texttt{KeyGen}()$ *and saves them; the attacker gets the verification key* vk*. The game also makes an empty list* L *of signed messages.*

Moves *The attacker may, any number of times, send the game a message* m *of her choice. The game signs this message producing a signature* $\sigma \leftarrow \texttt{Sign}(sk, m)$*, adds* m *to* L *and returns* σ *to the attacker.*

Winning conditions *The attacker wins the game if she provides a message/signature pair* (m^*, sk^*) *such that (1) this pair verifies under the game's key:* $\texttt{Verify}(vk, m^*, \sigma^*) = 1$ *and (2) the game never signed the message* m^**, i.e.* $m^* \notin L$*.*

In neither of the above games would it make any difference if we gave the attacker an extra move to verify signatures: she already knows the verification key vk so she can do this herself.

🔒 **One-time signatures.** There are several reasons why it is useful to define several security notions of increasing strength for the same class of scheme, rather than just go with the strongest known definition. For signature schemes in particular, some protocols use a construction called a one-time signature: a signer who has a personal signing key pair (pk, sk) of some signature scheme generates, for each action that she performs in the protocol, a new key pair (pk', sk') of a one-time signature scheme and uses sk' to sign exactly one message whereas she signs the one-time public key pk' under her long-term key pk. One reason why one might do such a construction is for greater anonymity: in a voting scheme, a voter could send her ballot with a one-time signature under sk' to the ballot counting authority and her signature on pk' to another, independent authority.

The ballot is now anonymous in the sense that it is not linked to the voter's public key pk but in the case of a dispute, the information held by the two authorities together can be used to trace the ballot. Since sk' is only ever used once, a scheme secure under no-message attacks is sufficient for this application and in some cases this allows one to choose a more efficient signature scheme and/or reduce the signature size.

2.5 🔒 Blind signatures

Voting is one of several applications where it is useful to be able to sign messages without knowing their content. To ensure that only authorized voters cast ballots, one could ask voters to authenticate themselves with an authority who holds a signing key and signs the ballots of authorized voters. Unfortunately, a straightforward use of digital signatures here would reveal everyone's votes to the authority. Instead, one can use blind signatures: each voter fills in her ballot and blinds it — we will define this formally in a moment but think of blinding for now as placing the ballot in an envelope — then authenticates herself to the authority, who signs the blinded ballot without knowing its contents. The voter then turns the signature on the blinded ballot into a signature on the real ballot and casts this ballot along with the signature.

Blind signatures will require two security properties. Security for the signer requires that no-one can forge signatures on messages that the signer has not blind-signed, even though the signer will not usually know which messages she has signed. Security for the user (in our case, the voter) requires that the signer cannot learn which messages she has signed. We follow Fujioka et al. [FOO92] in the definition of blind signatures and Schröder and Unruh [US11] in the definition of security properties.

Definition 4. *A blind signature scheme is a tuple*

$$BS = (\texttt{KeyGen}, \texttt{Blind}, \texttt{Sign}, \texttt{Unblind}, \texttt{Verify})$$

of algorithms where `Verify` *is deterministic and the rest may be randomized. The key generation algorithm outputs a keypair* $(sk, vk) \leftarrow \texttt{KeyGen}()$. *The blinding algorithm takes a message* m *and a verification key* vk *and outputs a blinded message* b *and an unblinding factor* u*:* $(b, u) \leftarrow \texttt{Blind}(m, vk)$. *The signing algorithm takes a signing key* sk *and a blinded message* b *and outputs a blinded signature* $s \leftarrow \texttt{Sign}(b, sk)$. *The unblinding algorithm takes a verification key* vk*, a blinded signature* s *and an unblinding factor* u *and outputs a signature* $\sigma \leftarrow \texttt{Unblind}(vk, s, u)$. *The verification algorithm finally takes a verification key* vk*, a message* m *and a signature* σ *and outputs a bit* $v \leftarrow \texttt{Verify}(vk, m, \sigma)$ *that is 1 if the signature verifies.*

A blind signature scheme is correct if the following outputs $v = 1$ *for any message* m*, i.e. a correctly generated signature verifies:*

$$\begin{aligned}&(sk, vk) \leftarrow \texttt{KeyGen}();\ (b, u) \leftarrow \texttt{Blind}(vk, m);\ s \leftarrow \texttt{Sign}(sk, b);\\ &\sigma \leftarrow \texttt{Unblind}(vk, s, u);\ v \leftarrow \texttt{Verify}(vk, m, \sigma)\end{aligned}$$

Definition 5. *A blind signature scheme is unforgeable (secure for the signer) if no attacker can win the following game.*

Setup *The game creates a key pair* $(sk, vk) \leftarrow \texttt{KeyGen}$ *and saves them. It also creates a list* L *of signed messages which starts out empty. The attacker gets* vk*.*

Moves *The attacker may submit a message* m *for signing as long as* $m \notin L$*. The game runs* $(b, u) \leftarrow \texttt{Blind}(vk, m)$; $s \leftarrow \texttt{Sign}(sk, b)$; $\sigma \leftarrow \texttt{Unblind}(vk, s, u)$, *adds* m *to* L *and returns the signature* σ*. The attacker may use this move as many times as she likes.*

Winning conditions *The attacker wins if she can output a list of message/signature pairs*

$$((m_1, \sigma_1), (m_2, \sigma_2), \ldots, (m_{k+1}, \sigma_{k+1}))$$

satisfying the following conditions: (1) all messages are distinct: $m_i \neq m_j$ *for all pairs* (i, j) *with* $i \neq j$ *(2) all pairs verify i.e.* $\texttt{Verify}(vk, m_i, \sigma_i) = 1$ *for all* i *and (3) the attacker has made at most* k *signature moves, i.e. fewer than the number of messages she returns.*

The list L here serves a slightly different purpose than for plain digital signatures: it prevents the attacker from submitting the same message twice. The winning condition is that the attacker has produced signatures on more messages than she has used in signing moves, so at least one of her output pairs is a genuine forgery. The reason for this formulation is that some blind signature schemes allow you to take a message/signature pair (m, σ) and create a new signature $\sigma' \neq \sigma$ on the same message such that (m, σ') is still a valid message/signature pair on the same key.

In the blindness game, the attacker takes the role of the signer. She may interact with two users bringing messages of the attacker's choice to be signed; her aim is to guess which order the users come in.

Definition 6. *A blind signature scheme is blind (secure for the user) if no attacker can guess the bit* b *in the following game with better probability than one half.*

Setup *The game picks a bit* b *at random from the set* $\{0, 1\}$*.*

Moves *The attacker has only one move and she may use it only once. First, the attacker may send the game a verification key* vk*. The attacker may then choose a pair of messages* (m_0, m_1) *and send them to the game. The game runs* $(b_0, u_0) \leftarrow \texttt{Blind}(vk, m_0)$ *and* $(b_1, u_1) \leftarrow \texttt{Blind}(vk, m_1)$ *and sends* (b_b, b_{1-b}) *to the attacker. If the attacker returns a pair* (s_b, s_{1-b}) *then the game sets* $\sigma_0 \leftarrow \texttt{Unblind}(vk, s_0, u_0)$ *and* $\sigma_1 \leftarrow \texttt{Unblind}(vk, s_1, u_1)$*. If both* σ_0 *and* σ_1 *are valid, the game sends* (σ_0, σ_1) *to the attacker.*

Winning conditions *The adversary may make a guess for* b *at any time. This stops the game. The adversary wins if the guess is correct.*

Our presentation of blind signatures is that of Fujioka et al. [FOO92] that was used in their voting protocol which we are working towards. There is a more general notion of blind signatures where the `Blind`, `Unblind` and `Sign` algorithms are replaced by interactive algorithms for the user and signer, however not all blind signatures of the more general type can be used to construct a voting protocol in the manner that we do in this work.

We now turn our attention to a possible implementation of standard and blind digital signatures based on the famous RSA construction.

$\sum$

RSA. In 1978, Rivest, Shamir and Adleman constructed the first public-key encryption scheme [RSA78]. In 1985, Chaum used RSA to propose and construct a blind signature scheme which we will present in the next section; let us first describe the RSA construction.

Pick two prime numbers p, q and multiply them together to get $N = pq$. The RSA construction lives in the ring $\mathbb{Z}_N^*$: the elements are the integers $\{1, 2, \ldots, N-1\}$ with the operation of multiplication modulo N. This ring is not a field (p, q are zero-divisors after all) but for large N, if we pick a random element x from $\{1, \ldots, N-1\}$ the chance of hitting a non-invertible element is small. One idea behind RSA is that if you know N but not p and q, you can treat $\mathbb{Z}_N^*$ as if it were a field. Specifically, you can try and invert any element with Euclid's algorithm and if you find a non-invertible element then you can factor N (you've found a multiple of p or q that's coprime to the other factor of N) and vice versa. Factoring is arguably the most famous computationally hard problem in mathematics.

If you pick an element $x \in \mathbb{Z}_N^*$ coprime to N (not a multiple of p or q) and look at the subgroup $\{x^k \pmod{N} \mid k \in \mathbb{N}\}$ that it generates then this subgroup has order exactly $\phi(N) = (p-1)(q-1)$ where ϕ is the Euler totient function, i.e. $x^{(\phi(N)} = 1 \pmod{N}$. The RSA construction makes use of exponentiation modulo N as its basic operation. The idea is that if you pick a pair of integers e, d satisfying $e \cdot d = 1 \pmod{\phi(N)}$ then for any invertible $x \in \mathbb{Z}_N^*$ the equation $(x^e)^d = x^{e \cdot d} = x \pmod{N}$ holds, i.e. exponentiating with e and d are mutually inverse operations. Crucially, given N and any e that is coprime to N, it is considered hard to find the corresponding d or to compute $x^d \pmod{N}$ for random x. A cryptographer would say that $x \mapsto x^e \pmod{N}$ is a trapdoor one-way function: one-way because it is easy to compute yet hard to invert; "trapdoor" because given d it becomes easy to invert. Upon such a function one can construct much of modern cryptography. While it is clear that if you can factor N you can also invert the function $x \mapsto x^e \pmod{N}$ for any $e > 0$, it is less clear whether an attack on RSA implies the ability to factor N. However, RSA has stood the test of time in that no-one has managed to attack properly generated RSA keys of decent key sizes, either through factoring or any other means, since the system was first proposed.

To generate an RSA keypair (whether for encryption, signing or many other applications), the key generation algorithm `KeyGen` performs the following steps:

1. Pick large enough primes p and q and compute $N = pq$. The bitlength of N is your key length.
2. Pick any unit e of $\mathbb{Z}_N^*$ — choices such as $e = 3$ are common as they are efficient to work with.
3. Find d such that $ed = 1 \pmod{(p-1)(q-1)}$ (since you know p, q this can be done with a variation on Euclid's algorithm).
4. The public key is the pair (N, e). People can share e but everyone gets their own N. The private key is the tuple (N, e, d, p, q) — most of the time, the pair (N, d) suffices to work with though.

2.6 A blind signature based on RSA

We start with the construction of a digital signature scheme, which then we show how to turn into a blind one. The scheme is based on the RSA group described above and is as follows. The private signing key is (N, d) and public verification key is (N, e). To sign a message $m \in \mathbb{Z}_N^*$ (without blinding) with key (N, d), one computes

$$\texttt{Sign}((N,d), m) := H(m)^d \pmod N$$

and to verify a signature σ, check that the following returns 1:

$$\texttt{Verify}((N,e), m, \sigma) := \begin{cases} 1, \text{if} & \sigma^e = H(m) \pmod N \\ 0, \text{otherwise} \end{cases}$$

Here, H is a hash function that serves two purposes. First, it allows a signature of constant size on a message of any length. Secondly, the hash function is a necessary part of the security of this construction: without it, you could take any value x, compute $y = x^e \pmod N$ and claim that x is a signature on y.

Chaum's blind signature has the user blind the message m (or, more precisely, the value of $H(m)$) with a random value r before sending it to the signer and strip this factor out again afterwards:

`KeyGen`**:** Standard RSA key generation.
`Blind`$((N,e), m)$**:** pick a random r from $\mathbb{Z}_N^*$ and set $b := H(m) \cdot r^e \pmod N$, $u := r$.
`Sign`**:** as for the basic RSA signature.
`Unblind`$((N,e), s, u)$**:** Compute $\sigma := s/u \pmod N$.
`Verify`**:** as for the basic RSA signature.

Let us check correctness of the blind signature. We have

$$\sigma^e = (s/u)^e = ((H(m) \cdot r^e)^d)^e = H(m)^{e \cdot d}(r^{e \cdot d})^e = 1 \cdot 1^e = 1 \pmod N$$

Note that σ is exactly the standard RSA signature on m for verification key (N, e). The analysis for this scheme is due to Bellare et al. and [**?**] and we omit it in this work.

2.7 🔒 Commitment schemes

A commitment scheme is the cryptographer's equivalent of placing a message in an opaque envelope and placing this envelope on the table: no-one else can can read your message until you open the envelope but you cannot change the message that you have placed in the envelope either: you are committed to the message.

Definition 7. *A commitment scheme CS is a triple of algorithms*

$$CS = (\texttt{Setup}, \texttt{Commit}, \texttt{Open})$$

called the setup, commitment and opening algorithms. The setup algorithm outputs some commitment parameter $p \leftarrow \texttt{Setup}()$. The commitment algorithm takes a parameter p and a message m and returns a commitment c and an opening key k: $(c, k) \leftarrow \texttt{Commit}(p, m)$. The opening algorithm takes a parameter p, a message m, a commitment c and a key k and returns a bit b to indicate whether the commitment matches the message: $b \leftarrow \texttt{Open}(p, m, c, k)$. The opening algorithm must be deterministic.

A commitment scheme must satisfy the following correctness property. For any message m, if you run

$$p \leftarrow \texttt{Setup}();\ \ (c, k) \leftarrow \texttt{Commit}(p, m);\ \ b \leftarrow \texttt{Open}(p, m, c, k)$$

then $b = 1$ i.e. correctly commited messages also open correctly.

Security of commitment schemes has two parts. A commitment is hiding if you cannot extract a committed message from a commitment until it is opened. A commitment is binding if you cannot change it once committed.

In more detail, the hiding property says that for any two messages of your choice, if you are given a commitment to one of them then you cannot guess better than at random which message was committed to.

Definition 8. *A commitment scheme $CS = (\texttt{Setup}, \texttt{Commit}, \texttt{Open})$ is hiding if no attacker can win the following game with better probability than one half.*

Setup *The game picks a bit b at random and creates a parameter $p \leftarrow \texttt{Setup}()$. The attacker gets p.*

Moves *The attacker may, once only, send a pair of messages m_0, m_1. The game runs $(c, k) \leftarrow \texttt{Commit}(p, m_b)$ and returns c to the attacker.*

Winning conditions *The attacker wins if she guesses b. A guess stops the game.*

The binding property asks the attacker to produce one commitment c and two different messages m, m' to which she can open the commitment, i.e. keys k, k' (which may or may not be the same) such that `Open` returns 1 on both triples involved.

Definition 9. *A commitment scheme* $CS = (\texttt{Setup}, \texttt{Commit}, \texttt{Open})$ *is binding if no attacker can win the following game.*

Setup *The game creates parameters* $p \leftarrow \texttt{Setup}()$*. The attacker gets* p*.*

Moves *No moves.*

Winning conditions *The attacker may provide a string* c*, two messages* m, m' *and two keys* k, k'*. She wins if (1)* $m \neq m'$ *and (2) both* $\texttt{Open}(p, c, m, k)$ *and* $\texttt{Open}(p, c, m', k')$ *return 1.*

Commitment is one of the few cryptographic primitives that can be built securely against attackers with unlimited resources, however a commitment scheme can only be either “perfectly hiding” or “perfectly binding”, not both at once. All decent commitment schemes are both hiding and binding against computationally bounded attackers.

2.8 The FOO protocol

The cryptographic tools we introduced above allow us to describe the voting scheme presented by Fujioka, Okamoto and Ohta at Auscrypt 1992 [FOO92]. This scheme was also the one that we motivated in the informal example above and has the convenient abbreviation FOO.

The FOO protocol uses two administrators, a counter who publishes all ballots sent to her and an authority who checks voters’ eligibility and can produce blind signatures. Voters must be able to talk anonymously to the counter; this requirement could be achieved with cryptographic tools that we will introduce later such as mix-nets.

The FOO protocol assumes that there is some public-key infrastructure in place in which each voter has a digital signature keypair and the association of verification keys to voters is public. FOO requires each voter to perform four steps:

1. Prepare a ballot on her own, sign it and save a private random value.
2. Authenticate herself to the authority and get a blind signature on the ballot.
3. Submit the ballot and authority signature anonymously to a ballot counter (this is equivalent to publishing the ballot).
4. After voting has closed, submit the private random value from step 1 to the counter, also anonymously.

Definition 10. *The FOO protocol is the following protocol for voters, an authority and a ballot counter.*

Tools The FOO protocol requires a digital signature scheme Σ, a blind signature scheme BS and a commitment scheme CS. We write algorithms with the scheme name as prefix, for example $BS.\mathtt{Sign}$, to avoid ambiguity. We assume that some commitment parameters $p \leftarrow CS.\mathtt{Setup}()$ have been generated.

Voter The voter starts out with a digital signature keypair (sk_V, vk_V) for Σ, a vote v and the authority's blind signature verification key vk_A.

1. She creates a commitment $(c, k) \leftarrow CS.\mathtt{Commit}(p, v)$, blinds it as $(b, u) \leftarrow BS.\mathtt{Blind}(vk_A, c)$ and signs this as $\sigma_V \leftarrow \Sigma.\mathtt{Sign}(sk_V, b)$.
2. She then sends (ID_V, b, σ_V) to the authority and expects a blinded signature s in return. Here ID_V is some string describing the voter's identity.
3. On receipt of s, she creates the blind signature $\sigma_A \leftarrow \mathtt{Unblind}(vk_A, s, u)$ and sends her ballot (c, σ_A) anonymously to the counter. The counter replies with some random identifier i.
4. After voting has closed and the counter has invited the voters to open their ballots, the voter sends (i, v, k) anonymously to the counter.

Authority The authority has a keypair (sk_A, vk_A) for a blind signature scheme. She also has access to a table T of the identities and verification keys of all eligible voters: (ID_V, vk_V) for all voters V. Further, the authority has a list L of the identities, blinded ballots and signatures of all voters who have already voted (this list starts out empty of course). When a voter sends the authority a triple (ID_V, b, σ_V) the authority checks that the voter is eligible to vote, i.e. $ID_V \in T$, and retrieves the corresponding verification key vk_V. The authority then checks that the signature is valid: $\Sigma.\mathtt{Verify}(vk_V, b, \sigma_V) = 1$. If this is correct, the authority checks that the voter has not already voted, i.e. ID_V does not appear in L. The authority then adds (ID_V, b, σ_V) to L and returns a blind signature $s \leftarrow BS.\mathtt{Sign}(sk_A, b)$ to the voter.

At the end of the voting phase, the authority publishes the list L.

Counter The ballot counter starts out with the authority's verification key vk_A. The counter holds no secrets and performs no secret operations: the entire protocol for the counter can be performed in public and therefore checked by anyone.

During the voting phase, the counter anonymously receives ballots (c, σ). She checks that each incoming ballot is valid, i.e. $BS.\mathtt{Verify}(vk_A, c, \sigma) = 1$ and publishes all accepted ballots along with their signatures and a unique identifier i, i.e. the counter maintains a list of entries (i, c, σ). The identifiers could be randomly chosen or simply sequence numbers.

At the end of the election, the counter invites all voters to open their ballots. On receipt of an anonymous message (i, v, k) the counter retrieves the entry (i, c, σ) and if such an entry exists, she computes $x \leftarrow CS.\mathtt{Open}(p, v, c, k)$. If this returns 1, the ballot is valid and the counter adds the vote v to the set of valid votes. Finally, the counter tallies all valid votes.

Privacy of FOO. The FOO protocol was published in 1992, before any formal models for voting schemes had been developed. Consequently, the original paper only offers a "proof sketch" for privacy; no-one has presented a formal analysis of privacy in FOO to date although there is no reason to suspect that this property would not hold. The modelling of anonymous channels is also not standardised; one could use mix-nets (discussed in Section 6) to achieve anonymity in which case the anonymity offered is well understood.

We give a brief sketch of why FOO is private. Before the tallying phase, each voter submits a pair (c, σ). c is a commitment and thus hides the contained vote; σ is a signature on the commitment so it cannot leak more information on the vote than the commitment itself. Even the authority only sees the commitment and not the vote in clear, so votes are kept private.

After tallying, the votes are revealed and FOO argue that no-one, not even the authority and vote counter together, can link a voter to a vote. This is because the items on the board are posted anonymously and the only non-anonymous communication that might link a voter to a ballot occurs when the voter obtains a signature from the authority. However, the authority only sees a blinded commitment and the returned signature is a blind signature, so the authority cannot link the ballots on the board to the commitments that she has signed.

Verifiability of FOO. To date, no formal models for verifiability exist that could be usefully applied to FOO. We will briefly argue informally why FOO is a verifiable protocol. Specifically, we check the following properties.

Individual verifiability Voters can check that their ballot was counted.

Universal verifiability Anyone can check that all ballots were counted correctly.

Ballot verifiability Anyone can check that all ballots correspond to correct votes.

Eligibility verifiability Anyone can check that only eligible voters have voted, and only once each.

For individual verifiability, since the counter just publishes all ballots the voter can check if her ballot is included among the published ones. Better still, in case of a dispute the voter can expose a cheating counter if the counter refuses to accept a correctly signed ballot.

Universal verifiability is easy since the counter holds no secrets: anyone can repeat the count of all opened ballots. The same holds for ballot verifiability since the ballots are opened individually.

Indeed, the nature of the FOO protocol is that only eligibility is assured by cryptographic means alone; the other verifiability properties follow from the public nature of the bulletin board and counting operation. This comes at the cost of practicality: voters must interact with the system twice, once to cast their ballots and once again to open them. Later systems such as Helios only require voters to interact with the system once; verifiability will become a more involved matter here.

For eligiblility, first note that anyone can verify that only correctly signed ballots are counted. If we assume that the authority is honest then only eligible voters will receive a signature from the authority and only once each. Even if the authority is dishonest, its log L would show if it had ever blind- signed a ballot that was not accompanied by a correct signature from a legitimate voter key, or if the authority had signed two ballots with a signature from the same voter.

Dispute resolution in FOO. In addition to verifiability, the FOO protocol provides a digital audit trail that can be used to resolve many disputes which could arise. We mention some cases that were addressed in the original paper; we imagine that all these disputes could be brought before a judge or election official. In the following we assume that honest parties' keys are not available to cheaters. This means that in any dispute between a honest and a dishonest party, the honest party will be able to convince a judge that the other side is cheating - more precisely, that either the other side is cheating or her signature has been forged. In the following we give a list of possible accusations and how a judge should respond.

- The authority refuses to give a legitimate voter a signature, or provides her with an invalid signature.
  The voter can publish a blinded ballot and her digital signature on it; a judge can now ask the authority to either sign this ballot or give a reason for refusing to do so. If the judge asks to see the authority's blinded signature, the voter can reveal her vote and blinding factor to the judge who can then check the unblinding step and verify the resulting signature. This way, a cheating authority will always be exposed.
- The authority claims that a voter has already voted.
  A judge can ask for the voter's previous blinded ballot and digital signature as proof. If the authority fails to produce this, she is exposed - if she does produce this, the voter must explain why said blinded ballot carries a signature under her key.
- The authority signs a ballot that does not come from a legitimate voter, or more than one ballot from the same voter.
  The judge checks the counter's published ballots against the authority's list L for any signed ballots that do not have a valid voter-signature in L, or two ballots with the same voter's signature.
- The authority signs something that is not a legitimate ballot.
  If the illegitimate ballot is never opened, it does not contribute to the result and can be ignored. Once a ballot is opened, the judge can check its contents. It is not the authority's fault if its signature is discovered on an invalid ballot: since the authority's signature is blind, the authority has no way of knowing the contents of what it signs.
- A voter tries to vote more than once.
  The judge checks that all the counter's ballots are also referenced in the list L and then checks the list L for two different ballots with the same voter's signature.

- The counter refuses to accept a legitimate ballot.

  The judge checks the signature on the disputed ballot; if it verifies and the counter still refuses then the counter is exposed as a cheater. The same applies to opening keys k where the judge checks using the opening algorithm.
- The counter accepts an illegitimate ballot (without a valid signature).

  The judge checks the signature; if it fails the counter is exposed as a cheater. The same applies to opening information.
- The counter produces a false result.

  The judge recomputes the result and disqualifies the counter if the results do not match.

3 Homomorphic Voting

The FOO protocol from the last section scores well on privacy, verifiability and dispute resolution but has one major drawback: voters need to interact with the voting system at two different times, once to cast their ballot and once again after the voting period has ende to open their ballot. A different approach to cryptographic voting removes this drawback.

3.1 Motivation and example

Here is another sketch of a "physical" voting protocol. Consider a yes/no referendum. Each voter gets two balls of the same size and appearance except that one weighs 1 lb[3]and the other 2 lb. To vote no, the voter writes her name on the lighter of two balls and places it on a tray; to vote yes she does the same for the heavier one. This allows anyone to check that only eligible voters have voted and only once each by comparing the names on the cast ball(ot)s with a list of eligible voters. To tally the election, one first counts the number of balls cast, then weighs the entire tray and derives the number of light and heavy balls from the total amount and weight of the balls. This way, the amount that each individual ball(ot) contributed to the tally is hidden from everyone except the voter who cast it. One point that we will have to take care of in the cryptographic protocol based on this idea is how we prevent a voter from casting a forged ball weighing more than 2 lb to gain an unfair advantage.

The cryptographic tool that we will use to build the equivalent of the balls above goes by the name of "homomorphic asymmetric encryption". The adjective "homomorphic" describes a scheme where ciphertexts (or commitments, signatures etc.) can be added together to create a new ciphertext for the sum of the original messages. Before we can define homomorphic asymmetric encryption, we first need to define what asymmetric encryption is in the first place.

[3] One pound, abbreviated lb, is around 0.454 kg.

3.2 🔒 Asymmetric encryption

Asymmetric or “public key” encryption is perhaps the best-known invention of modern cryptography. It is certainly one of the oldest: it was first suggested by Diffie and Hellman in 1976 [DH76] and implemented successfully by Rivest, Shamir and Adleman in 1978 [RSA78].

There are many ways to explain asymmetric encryption using physical terms: our favourite example is a letter-box. Anyone can send you letters by placing them in your letter-box but only you can get letters out of the box again[4]. Indeed, once someone has placed a letter in your letter-box, even they can’t get it out again.

As for digital signatures, we define the types of algorithms required and the security requirements.

Definition 11. *An asymmetric encryption scheme E is a triple of algorithms*

$$E = (\texttt{KeyGen}, \texttt{Encrypt}, \texttt{Decrypt})$$

where the key genration algorithm takes no input and returns a pair $(pk, sk) \leftarrow$ $\texttt{KeyGen}()$ known as the public and secret key. The encryption algorithm takes a public key pk and a message m and returns a ciphertext $c \leftarrow \texttt{Encrypt}(pk, m)$. The decryption algorithm takes a secret key sk and a ciphertext c and outputs either a decrypted message $d \leftarrow \texttt{Decrypt}(sk, c)$ or declares the ciphertext invalid, which we indicate with the special output symbol $\perp$. The decryption algorithm must be deterministic.

The correctness condition is that for any message m, the following operations result in $d = m$:

$$(pk, sk) \leftarrow \texttt{KeyGen}();\ c \leftarrow \texttt{Encrypt}(pk, m);\ d \leftarrow \texttt{Decrypt}(sk, c)$$

3.3 Security of encryption

It took the community of cryptographers some time to come up with the definitions of security for asymmetric encryption that are in use today. The first obvious condition is that given a ciphertext, you should not be able to tell the contained message. Unfortunately this is not sufficient, here is an example why. Alice, a famous cryptographer, wishes to announce the birth of her child to her family while keeping its gender secret from the world at large for now. She encrypts the good news under her family’s public keys and sends out the ciphertexts. Eve, an eavesdropper from the press, obtains Alice’s ciphertexts. This should not matter — this is exactly what encryption is for, after all.

Eve guesses that Alice’s message is either “It’s a boy!” or “It’s a girl!”. Instead of trying to decrypt a completely unknown message, Eve would already

[4] The design of letter-boxes varies a lot between countries; we have in mind the continental European style where letterboxes have a flap to insert letters and the owner can open a door on the letter-box with a key to retrieve letters.

be happy if she could tell which of two messages (that she already knows) Alice has encrypted. Further, Eve might be able to obtain Alice's family's public keys from a directory — they are meant to be public as their name suggests — and Eve can encrypt both her guessed messages herself under these public keys and check if either of her ciphertexts matches the one sent by Alice. If so, Eve has effectively broken Alice's secret.

This story gives rise to two more requirements: given a ciphertext and two candidate messages, you should be unable to guess which of the two messages was encrypted; given two ciphertexts you should not be able to tell if they encrypt the same message or not. The first common security requirement for encryption is known as indistinguishability under chosen plaintext attack, abbreviated IND-CPA. Here, the attacker may chose any two messages, send them to the security game and get an encryption of one of them back; a scheme is called IND-CPA secure if she cannot tell which message the security game chose to encrypt.

Definition 12. *An asymmetric encryption scheme E is IND-CPA secure if no attacker can win the following game with better probability than $1/2$, the probability of guessing at random.*

Setup *The game creates a keypair* $(pk, sk) \leftarrow \texttt{KeyGen}()$ *and gives the attacker the public key pk. The game also picks a bit b randomly from $\{0, 1\}$ and keeps this secret.*

Moves *Once in the game, the attacker may pick a pair of messages m_0 and m_1 of the same length*† *and send them to the game. The game encrypts* $c \leftarrow \texttt{Encrypt}(pk, m_b)$ *and returns this to the attacker.*

Winning conditions. *The attacker may make a guess at b which ends the game. The attacker wins if she guesses correctly.*

(†) The condition that the two messages be of the same length is to avoid the attacker guessing the message from the ciphertext length. In the example above, "boy" has three letters but "girl" has four so any encryption scheme that returns a ciphertext with the same number of characters as the message is vulnerable to such an attack and Alice should pad both her messages to the same length to be safe. In practice, many encryption schemes work not on characters but on blocks of characters in which case the restriction can be weakened to both messages producing the same number of ciphertext blocks; the ElGamal scheme which we will consider later operates on messages in a fixed group where all messages have the same length "1 group element" and this condition is vacuous.

There are stronger notions of security for encryption that we will introduce at the appropriate point later in this work and explain how they relate to keeping encrypted votes private.

3.4 🔒 Homomorphic encryption

A homomorphic encryption scheme offers an additional algorithm `Add` that takes two ciphertexts and a public key and produces a new ciphertext for the "sum" of the two messages in the original ciphertexts. We put "sum" in quotes because the principle can be applied to different operations such as multiplication as well.

Definition 13. *An asymmetric encryption scheme*

$$E = (\texttt{KeyGen}, \texttt{Encrypt}, \texttt{Decrypt})$$

is homomorphic if there are these additional operations:

- *An operation + on the message space.*
- *An algorithm* `Add` *that takes a public key pk and two ciphertexts c_1, c_2 and outputs another ciphertext s.*

The correctness condition is that for any messages m_1, m_2 the following returns $d = m_1 + m_2$:

$$(pk, sk) \leftarrow \texttt{KeyGen}();\ c_1 \leftarrow \texttt{Encrypt}(pk, m_1);\ c_2 \leftarrow \texttt{Encrypt}(pk, m_2);$$
$$c \leftarrow \texttt{Add}(pk, c_1, c_2);\ d \leftarrow \texttt{Decrypt}(sk, c)$$

Actually, we require a slightly stronger condition as the presentation above does not exclude the following degenerate construction: a "ciphertext" is a list of ciphertexts, the encryption algorithm returns a list with one element and `Add` just returns a list containing its two input ciphertexts. The decryption algorithm takes a list, decrypts each element individually and returns the sum of all decryptions. What we require in particular is that sums of ciphertexts look just like ordinary ones and even the legitimate decryptor cannot tell a sum from a simple ciphertext. For example, if a ciphertext decrypts to 2, the decryptor should not be able to tell if this was a direct encryption of 2, a sum of encryptions of 0 and 2 or of 1 and 1 etc.

$\sum$

Prime-order groups. We are now working towards the ElGamal encryption scheme that we will use to build a toy voting scheme called "minivoting" and then extend this to get the Helios voting scheme. ElGamal uses a prime-order group, we sketch a number-theoretic construction. To set up such a group, one typically picks a prime p such that $q = (p-1)/2$ is also prime (this is even more essential than for RSA, to avoid "small subgroup" problems). The group $\mathbb{Z}_p^*$ with multiplication modulo p has $p-1$ elements; since p is a large prime and therefore is odd there will be a factor 2 in $p-1$. If $(p-1)$ factors as $2 \cdot q$ where q is also prime and we pick an element $g \in \mathbb{Z}_p^*$ of order q then the subgroup $G := \langle g \rangle \subset \mathbb{Z}_p^*$ is itself a cyclic group of order q. Since q is prime, G has no true subgroups, i.e. apart from the identity, there is no extra "structure" to be discovered by examining the traces of individual group elements[5].

The ElGamal encryption scheme lives in such a group G given by parameters (p, q, g). Since G is isomorphic to $\mathbb{Z}_q$, we have an inclusion

$\mathbb{Z}_q \to G, (x \mapsto g^x \pmod p)$. This map is efficient to compute (square-and-multiply and variations) but is considered to be hard to invert on randomly chosen points. Its inverse is known as taking the discrete logarithm of a group element. Further, given two group elements h and k, there are unique integers $a, b \in \mathbb{Z}_q$ such that $h = g^a \pmod p$ and $k = g^b \pmod p$. The group operation sends such (h, k) to $h \cdot k = g^{a+b} \pmod p$. We can define a further operation $\otimes$ that sends such (h, k) to $g^{a \cdot b} \pmod p$. This turns out to be a bilinear map on G called the Diffie-Hellman product and it is considered to be hard to compute in general; computing it for random h, k is the computational Diffie-Hellman problem. For random h, k and another group element z, it is even considered hard to tell whether $z = h \otimes k$ or z is just another random group element, this is called the decisional Diffie-Hellman problem. However, if you are given the integer a (from which you could easily compute $h = g^a$ in G) then you can easily take the Diffie-Hellman product with any k as $h \otimes k = k^a \pmod p$.

Definition 14. *A Diffie-Hellman group is a group $\langle g \rangle \subset \mathbb{Z}_p^*$ of order q for p, q primes with $(p-1)/2 = q$. Such a group is given by parameters (p, q, g) and such parameters can be public and shared among all users of a cryptosystem.*

To generate a Diffie-Hellman keypair, pick parameters if required and pick an x at random from $\mathbb{Z}_q$, then set $y = g^x \pmod p$. Your secret key is x and your public key is y.

Two comments on this scheme are in order. First, the group has order q but is represented as a subgroup of $\mathbb{Z}_p^*$. The rule to remember is, always reduce group elements modulo p and integers (exponents) modulo q. This is why you pick your secret key from $\mathbb{Z}_q$ (it's an integer) and then compute the public key (a group element) modulo p.

Secondly, there are other possible realisations of cryptographically useful prime-order groups in which the Diffie-Hellman product and discrete logarithms are assumed to be hard. The most popular alternative uses a representation on an elliptic curve over a finite field; we will not go into details of the construction in this work but the ElGamal encryption scheme works identically whether you are using a $\mathbb{Z}_p^*$ group or an elliptic curve group.

3.5 ElGamal

The ElGamal encryption scheme [E85] was invented in 1985. It encrypts a message $m \in G$ as a pair (c, d):

Definition 15. *The ElGamal encryption scheme is the encryption scheme given by the algorithms below.*

`KeyGen` Pick or obtain parameters (p, q, g). Pick sk at random from $\mathbb{Z}_q$ and set $pk = g^{sk} \pmod p$, return (pk, sk).

`Encrypt`(pk, m) Pick r at random from $\mathbb{Z}_q$ and set $c = g^r \pmod{p}, d = m \cdot pk^r \pmod{p}$. Return (c, d).
`Decrypt`$(sk, (c, d))$ Compute $m = d/c^{sk} \pmod{p}$.

The message is multiplied with a random group element, resulting in a uniformly distributed group element d. Since r was random in $\mathbb{Z}_q$, so is $pk^r \pmod{p}$ for any group element pk, thus d on its own is independent of m. To allow the key-holder, and her only, to decrypt, an additional element c is provided. Since $m = d/(c \otimes y)$, the decryptor can compute m with her secret key; for anyone else extracting the message given both c and d is equivalent to solving the computational Diffie-Hellman problem. Telling which of two messages was encrypted (the IND-CPA security game) is equivalent[6] to solving the decisional Diffie-Hellman problem.

Exponential ElGamal. ElGamal is homomorphic but the operation is not as useful as we would like: for ciphertexts (c, d) and (c', d') we can set

$$\texttt{Add}((c, d), (c', d')) := (c \cdot c' \pmod{p}, d \cdot d' \pmod{p})$$

such that for messages m, m' in G we get a ciphertext for $m \cdot m' \pmod{p}$. What we would really like for voting is a scheme where messages lie in the additive group $\mathbb{Z}_q$ and we can perform homomorphic addition, rather than multiplication, of ciphertexts. If our messages are restricted to small integers (indeed, in our ballots they will be wither 0 or 1) then we can use a variation called exponential ElGamal: to encrypt an integer m, replace the d-component with $g^m \cdot pk^r \pmod{p}$. For two ciphertexts (c, d) for m and (c', d') for m' the `Add` operation now produces a ciphertext that decrypts to $g^{m+m' \pmod{q}}$ as desired. While getting the exponent back from an arbitrary group element is hard (the discrete logarithm problem), for small enough exponents this can be done just by trying $g^0, g^1, g^2, \ldots$ until we find the correct decryption. This is the approach taken by Helios, which we will replicate in our minivoting scheme as a first step towards constructing Helios.

Other homomorphic schemes. Besides ElGamal, there are numerous other homomorphic encryption schemes. The DLIN scheme of Boneh et al. [BBS04] works on similar principles to ElGamal but uses a different security assumptioni and is thus applicable to different kinds of groups. The Paillier scheme [P99] operates in composite-order RSA-type groups and offers additive instead of multiplicative homomoprhism, but is much less efficient than ElGamal. There are also many extensions of ElGamal such as Signed ElGamal (a.k.a. TDH0) and TDH2 [SG98] from which an ElGamal ciphertext with homomorphic properties can be extracted.

[6] Ignoring some details.

3.6 Minivoting

We will develop the concept of homomorphic voting in several steps, ending up with Helios as an example. The first step is a scheme called "minivoting" by Bernhard et al. from Esorics 2011 [BC+11]. Minivoting is not verifiable and indeed is only secure against passive attackers who cannot send malformed ciphertexts. In a later step we will add further components to minivoting in order to obtain a fully secure scheme.

Definition 16. *Minivoting is the following voting scheme for a yes/no question, based on a homomorphic asymmetric encryption scheme E with a message space $\mathbb{Z}_n$ for some n larger than the number of voters.*

Participants *Minivoting requires one authority, a public bulletin board to which everyone can post authenticated messages and any number of voters smaller than n.*

Setup *The authority creates a key pair $(pk, sk) \leftarrow E.\texttt{KeyGen}$ and posts pk to the bulletin board.*

Voting *Voters read the public key pk off the board. They choose $v = 1$ for "yes" and $v = 0$ for "no" and create a ballot $b \leftarrow E.\texttt{Encrypt}(pk, v)$ which they post on the board.*

Tallying *The authority uses the $E.\texttt{Add}$ operation to add all ballots, creating a final ballot s which she decrypts as $d \leftarrow E.\texttt{Decrypt}(sk, s)$. The authority then counts the number m of ballots submitted and posts the result "d yes, $m - d$ no" to the board.*

4 Vote Privacy

We give a notion of ballot privacy against observers for voting schemes, following the principles set out by the IND-CPA game for encryption. The attacker can choose two votes for each voter and the voters will either cast the first or second vote (all voters make the same choice which of the two to cast). The attacker's aim is to tell which choice the voters made, just like the IND-CPA game asks the attacker to tell which of two messages was encrypted. Since the two results that this game produces may differ, which would immediately tell the attacker what is going on, the game will always report the first result.

Definition 17. *A voting scheme has ballot privacy against observers if no attacker can do better in the following game than guess at random (with probability* 1/2*).*

Setup *The game picks a bit b at random and keeps it secret. The game then sets up the voting scheme and plays the voters, authorities and bulletin board.*

Moves *Once for each voter, the attacker may choose two votes v_0 and v_1. The game writes down both votes. If $b = 0$, the game lets the voter vote for v_0; if $b = 1$ the game lets the voter vote for v_1.*

The attacker may ask to look at the bulletin board at any point in the game. When all voters have voted, the game gives the attacker the result computed as if everyone had cast their first (v_0) vote.

Winning conditions *At any point in the game, the attacker may submit a guess for b. This ends the game immediately. The attacker wins if her guess is correct.*

Although we do not prove it here, we could show that if there is an attacker with a better than random chance of winning this game for the minivoting scheme (based on some homomorphic encryption scheme E) then we can build an attacker who wins the IND-CPA game for the same encryption scheme E with better than one half probability too. The rough idea is that any attacker guessing better than at random for the ballot privacy game must have selected at least one voter and given her different votes v_0 and v_1, so we could run the IND-CPA game with messages v_0 and v_1 and use the attacker's guess to make our guess at which one was encrypted. The crux of the proof is that the IND-CPA game allows only one challenge move whereas the ballot privacy game allows many voters. This gives us the following proposition.

Proposition 18. *For any IND-CPA secure homomorphic asymmetric encryption scheme, the derived minivoting scheme has ballot privacy against observers.*

In particular this holds for the minivoting scheme based on ElGamal.

4.1 🔒 Threshold encryption

Minivoting used a single authority which is bad for two reasons. First, a dishonest authority could decrypt individual ballots. Secondly, if the authority loses her key, the election cannot be tallied. (We will deal with the authority trying to claim a false result in a later section.)

Threshold schemes aim to mitigate these risks. In a k-out-of-n threshold scheme, there are n authorities and any subset of at least k can tally the election. In this way, a coalition of up to $k - 1$ dishonest authorities cannot decrypt individual ballots (or obtain early results) whereas up to $n - k$ of the authorities can drop out and the election can still be tallied.

In our definition of threshold schemes, the authorites run an interactive protocol to generate keys, as a result of which each authority obtains a public key share and a secret key share. A user of the scheme can run a key combination algorithm to obtain a single public key and encrypt messages with this. To decrypt, each authority that takes part in the decryption process produces a decryption share with her secret key and anyone can combine at least k decryption shares to recover the message.

Definition 19. *A (k, n) threshold encryption scheme consists of a key generation protocol* `KeyGen` *for n authorities and four algorithms*

$$(\texttt{CombineKey}, \texttt{Encrypt}, \texttt{DecryptShare}, \texttt{Combine})$$

The key generation protocol results in all participants obtaining a public key share pk_i and a secret key share sk_i. The key combination algorithm takes a list *of n public key shares and returns a public key $pk \leftarrow \texttt{CombineKey}(pk_1, \ldots, pk_n)$ or the special symbol $\perp$ to indicate invalid shares. The encryption algorithm works just like non-threshold encryption: $c \leftarrow \texttt{Encrypt}(pk, m)$. The decryption share algorithm takes a secret key share sk_i and a ciphertext c and outputs a decryption share $d_i \leftarrow \texttt{DecryptShare}(sk_i, c)$. The recombination algorithm takes a ciphertext c, a set $D = \{d_i\}_{i \in I}$ of at least k decryption shares and outputs either a message m or the symbol $\perp$ to indicate failure.*

The correctness condition is that for any message m and any set I of at least k authorities, the following yields $d = m$:

$$\begin{aligned}
&((pk_1, \ldots, pk_n), (sk_1, \ldots, sk_n)) \leftarrow \texttt{KeyGen}();\\
&pk \leftarrow \texttt{CombineKey}(pk_1, \ldots, pk_n);\ c \leftarrow \texttt{Encrypt}(pk, m);\\
&\textit{for } i \in I:\ d_i \leftarrow \texttt{DecryptShare}(sk_i, c);\ d \leftarrow \texttt{Combine}(c, \{d_i\}_{i \in I});
\end{aligned}$$

Threshold ElGamal for $k = n$. Here is an implementation of threshold encryption for $k = n$, i.e. all authorities must be present to decrypt. ElGamal can also be used for arbitrary (k, n) thresholds but the construction is more complex. The definition below is secure against up to $n - 1$ authorities as long as they follow the protocol, i.e. they may compute and communicate freely "on the side" but can not deviate from the key generation protocol. We will adapt the system to be secure against misbehaving authorities in a later section once we have introduced the necessary tools.

`KeyGen` All authorities agree on or obtain common parameters (p, q, g). Each authority then simply generates an ElGamal keypair under these parameters.

`CombineKey` Take all n shares $pk_1, \ldots, pk_n$ and multiply them together: $pk \leftarrow \prod_{i=1}^{n} pk_i \pmod p$.

`Encrypt` This is standard ElGamal encryption with the public key.

`DecryptShare`(sk_i, c) An ElGamal ciphertext is a pair $c = (a, b)$. Return the share $d_i := a^{sk_i} \pmod p$.

`Combine` On input a ciphertext $c = (a, b)$ and a set of n decryption shares $\{d_i\}_{i=1}^{n}$ set $d := b / \prod_{i=1}^{n} d_i \pmod p$.

This works because

$$d = b / \prod_{i=1}^{n} d_i = b / \prod_{i=1}^{n} a^{sk_i} = b / a^{\sum_{i=1}^{n} sk_i \pmod q} = b / a^{sk} \pmod p$$

where $sk := \sum_{i=1}^{n} sk_i \pmod q$ is the secret key corresponding to the public key pk, so this is just a normal ElGamal decryption. We draw the reader's attention to the correct use of ps and qs: the group elements are taken modulo p whereas the integers in the exponent are taken modulo $q = (p - 1)/2$.

4.2 Problems with Minivoting

Minivoting (even with threshold encryption) is not a secure scheme if some of the participants misbehave. For example,

1. A voter may encrypt g^2 to get an unfair advantage. For more complex ballots than yes/no questions, voters have even more ways to cheat.
2. A voter can stall the election by submitting a ballot for g^r for some random r — no-one will be able to decrypt the result anymore.
3. You have to trust the authorities that they have announced the correct result.

Luckily, cryptographers have found solutions to all these problems. They are:

1. Zero-knowledge proofs.
2. Zero-knowledge proofs.
3. Zero-knowledge proofs.

Zero-knowledge proofs are a technique to turn any protocol secure against observers into a protocol secure against misbehaving participants. The idea is that whenever a participant submits some information (say, a ballot) they must submit two things: first, the ballot and secondly, a proof that they have made a correct ballot. “Zero-knowledge” means that these proofs reveal nothing beyond that the ballot is correct. In particular, a proof that your ballot is correct does not leak your vote.

4.3 🔒 Zero-knowledge proofs

Zero-knowledge proofs[7]are tools that allow you to prove that you have done a certain operation correctly, without revealing more than that fact. In this section we develop the mathematical theory of zero-knowledge proofs based on so-called Σ-protocols and then give the protocols used in Helios. Our development and presentation of Σ-protocol theory follows the work of Bernhard [B14]; we present the Schnorr [S91], Chaum-Pedersen [CP92] and disjunctive Chaum-Pedersen (DCP) protocols. The term Σ-protocol is originally due to Cramer.

☼

Proofs in Helios. Helios uses zero-knowledge proofs in three ways:

- Each voter proves that she has cast a ballot for a valid vote, without revealing her vote.
- The authorities prove that they know their secret keys (that match the election public keys), without revealing their secret keys.
- At the end of the election, the authorities prove that they have tallied correctly (decrypted the result correctly), again without revealing their secret keys.

[7] Formally, one can distinguish zero-knowledge “proofs” from “arguments” and “proofs of knowledge” from “proofs of facts”. We ignore these distinctions here.

Consider an ElGamal keypair $(sk, pk = g^{sk} \pmod{p})$ for a group defined by parameters (p, q, g). Suppose you want to prove that you know the secret key matching the public key. One paper-based protocol, following the ideas in earlier sections, could work like this: prepare 100 keypairs. Write the secret keys on pieces of paper and place them in opaque envelopes; write the matching public keys on the outside of the envelopes. Let someone pick 99 of the envelopes, open them and check that the keys inside match those outside (i.e. that $g^{sk_i} = pk_i \pmod{p}$ for each pair (pk_i, sk_i) opened). If this holds for randomly chosen envelopes, then with probability at least 99/100 the last envelope is also correct and you can use the key written on the outside as your public key. This protocol has two slight drawbacks. First, it is wasteful with envelopes — especially if you want a really high security margin like $1 - 2^{100}$ — which is not good for the environment. Secondly, you only convince one person that your key is correct: an observing third party cannot know if you have not agreed in advance which envelope your accomplice will not pick, in which case you could easily cheat and claim someone else's public key as your own, for which you do not know the secret key.

To address the first problem we note that the map turning secret keys into public keys, $(sk \mapsto g^{sk} \pmod{p})$, is linear if you look at it the right way. If you have two key pairs (sk, pk) and (sk', pk') then $sk + sk' \pmod{q}$ is the secret key matching the public key $pk \cdot pk' \pmod{p}$. Note that the operation on public keys is written as a multiplication instead of an addition but the secret and public keys live in isomorphic groups so they are really both just group operations. The map taking a secret key to a public key is an isomorphism, its inverse is the discrete logarithm operation which is (hopefully) hard to compute but a well-defined map nonetheless. Further, if you rerandomise a secret key sk with an integer r to get $sk' = r \cdot sk \pmod{q}$ then the corresponding public key is $pk' = pk^r \pmod{p}$ where pk was the public key corresponding to sk.

On linearity. To speak of a linear map we actually need a vector space over a field; since we are working in a prime-order cyclic group $\mathbb{Z}_q$ for the exponents we may embed this into the finite field $\mathbb{F}_q$ by adjoining the obvious field multiplication structure to get our field. Any field is a one-dimensional vector space over itself so the secret key space can be interpreted as a $\mathbb{F}_q$–vector space. The public key space is isomorphic to the secret key space, so we can really speak of the isomorphism $(sk \mapsto g^{sk} \pmod{p})$ as an $\mathbb{F}_q$–linear map. The statement that you can add secret keys is saying that our map commutes with vector addition; rerandomising a key is field multiplication and together these two properties show linearity. Why we do all this should become clear later when we construct Σ-protocols on statements involving vectors which are tuples of group elements.

With these foundations in place, here is a protocol to prove that you know a secret key sk matching a given public key pk. Pick a second keypair (sk', pk') and reveal pk'. Let someone pick a number c between 0 and $n-1$ (for $n \leq q$, to be exact) and compute the linear combination $pk'' = pk' \cdot pk^c \pmod{p}$. You then reveal $sk'' := sk' + c \cdot sk \pmod{q}$ and your challenger checks that $g^{sk''} = pk''$ (mod p). This protocol does the same (and some more) than the one above with n envelopes. To see why, consider the point of view of the challenger who knows pk and pk' and has just picked c. Unless you know the correct sk, or are able to compute such a sk, there is only one single value of c for which you have any hope of providing the correct answer that will convince your challenger. The probability of cheating is bounded by $1/n$ and n can be chosen as large as you like (up to $q-1$).

Suppose that there are two distinct values of c, namely c_1 and c_2, for which you have some nonzero probability of finding a correct answer and call these answers sk''_1 and sk''_2. By a bit of linear algebra, since both answers are correct we must have

$$g^{sk''_1} = pk' \cdot pk^{c_1} \pmod{p} \tag{1}$$

$$g^{sk''_2} = pk' \cdot pk^{c_2} \pmod{p} \tag{2}$$

which suggests that we divide the two, cancelling pk':

$$g^{sk''_1}/g^{sk''_2} = pk^{c_1}/pk^{c_2} \pmod{p} \tag{3}$$

$$g^{sk''_1 - sk''_2 \pmod{q}} = pk^{c_1 - c_2 \pmod{q}} \pmod{p} \tag{4}$$

but the exponent space is a field and c_1, c_2 are distinct so we can rearrange to get

$$g^{\frac{sk''_1 - sk''_2}{c_1 - c_2} \pmod{q}} = pk \pmod{p} \tag{5}$$

and this exponent is exactly the secret key sk such that $g^{sk} = pk \pmod{p}$. In other words, if you can find the answers sk''_1, sk''_2 to two different challenges c_1, c_2 then from this information you can compute a secret key $sk = \frac{sk''_1 - sk''_2}{c_1 - c_2}$ (mod q) yourself. This property is called “special soundness”. Conversely, if you do not know the secret key sk then you cannot hope to answer any two different challenges c_1, c_2 in the same protocol so your probability of cheating is at most $1/n$. This inability for Alice to cheat (except with a tiny probability) is called “soundness” of the protocol. Special soundness implies soundness.

This protocol is Schnorr’s protocol [S91] for proof of knowledge of a discrete logarithm. An additional advantage of this protocol over the envelope-based one is that you can pick one keypair (pk, sk) and re-run the protocol to convince many different people that you know your secret key, picking a new (pk', sk') keypair for each person you run the protocol with but keeping the same public key pk all the time. (If you ever re-use a pk', the two people that you did the protocol with using the same pk' can get together and compute your secret key exactly as described above, unless they both happened to pick the same challenge which is very unlikely for large n.) A cryptographer would say that Schnorr's protocol is a "proof of knowledge" of a value sk such that $g^{sk} = pk \pmod p$ because the protocol satisfies the following condition:

Proposition 20. *Any person that can convince a challenger with more than $1/n$ probability in Schnorr's protocol (for challenges from $\{0, \dots, n-1\}$ and public key pk) can also compute a secret key sk that matches pk.*

Suppose that Alice is using Schnorr's protocol to prove to Bob that she knows her secret key. We have just established that Alice cannot cheat Bob (except with probability at most $1/n$). Can Bob cheat? That is, can Bob use the Schnorr protocol to gain more information about Alice's secret key than he could if he only got her public key pk in the first place?

The answer is of course "no" — at least if Bob picks his challenge randomly. What Bob gets to see in this protocol is a new public key pk' and a secret key sk'' for a c of his choice; we argue that Bob could just as well create these elements himself if Alice didn't want to run the protocol with him.

1. Bob picks a value c at random from $\{0, 1, \dots, n-1\}$.
2. Bob picks a value sk'' at random from $\{0, 1, \dots, q-1\}$ and computes $pk'' := g^{sk''} \pmod p$.
3. Bob sets $pk' = pk''/pk^c \pmod p$ where pk is Alice's public key.

The triple (pk', c, sk'') looks exactly like one that would be generated if Bob did Schnorr's protocol with Alice; in particular the verification equation holds: $g^{sk''} = pk' \cdot pk^c \pmod p$ from the definition of pk' and c, pk', sk'' are uniformly random subject to this equation holding. So Schnorr's protocol gives Bob no more information about Alice's key than he could already compute by himself, if Bob chooses his challenge randomly. This property of the protocol is called "honest verifier zero-knowledge".

Proposition 21. *If Bob picks his challenge c randomly, he gains no information from a run of Schnorr's protocol with Alice.*

Another way of phrasing this argument is that if Alice does the protocol with Bob and Carol observes this, the protocol convinces Bob but it cannot convince Carol: Alice and Bob could be working together to cheat Carol. To do this, Alice could pick any public key for which she does not know the secret key, Bob could create values as above and agree them with Alice beforehand and they could run the protocol together on these values.

The bit about honest verifiers — Bob picking his challenge randomly — is not just a technicality. For sure, it cannot help Bob to choose his challenge in a way that Alice could predict (this just allows Alice to cheat, but not Bob). Bob can however throw a spanner in the works by choosing his challenge as the value of a pseudorandom or hash function on input the values he has seen so far, pk and pk'. This breaks the "simulation" argument above because Bob had to choose his c before he picked pk'. This is exactly why Alice needs to know sk to convince Bob but Bob does not need to know sk to simulate the protocol: when

Alice is talking to Bob, she has to send him pk' before he chooses c but on his own, Bob can do it "backwards".

Before we go on to fix this problem, we slightly abstract Schnorr's protocol which will come in useful when we discuss other protocols along similar lines such as Chaum-Pedersen. The flow of messages in Schnorr's protocol can be drawn to look like the Greek letter Σ which prompted Cramer [C96] to call such protocols "Σ-protocols".

Definition 22. *A Σ-protocol is a protocol after the following template for Alice to prove knowledge of a preimage $x : y = \phi(x)$ of a value y to Bob.*

1. *Alice samples a random a from the domain of ϕ, sets $b := \phi(a)$ and sends b to Bob. We assume that Bob already knows y, alternatively Alice could send y to Bob too.*
2. *Bob picks a challenge c randomly from the set $\{0, 1, \ldots, n-1\}$ which must form part of the field $\mathbb{F}$. Bob sends c to Alice.*
3. *Alice computes $d := a + cx$ in the field $\mathbb{F}$ and sends d to Bob.*
4. *Bob checks that $\phi(d) = b + c \cdot y$ where this calculation is done in the $\mathbb{F}$–vector space of which y is an element ($c \cdot y$ is scalar multiplication; c is a field element).*

A Σ-protocol gives Alice no more than a $1/n$ probability of cheating and Bob, if he chooses his challenge randomly, no advantage in finding Alice's preimage.

Proposition 23. *A Σ-protocol derived from the template above is a honest verifier zero-knowledge proof of knowledge of a preimage of a linear map ϕ (over some field $\mathbb{F}$).*

We have a second problem beyond Bob not choosing a random c. If Alice wants to use a zero-knowledge proof to convince everyone that her ballot is valid, using an interactive proof like the one above would mean that everyone must be able to challenge Alice to run a protocol with them, even after the election has closed. This is clearly impractical. What we want is a non-interactive proof, where Alice can once and for all time convince every possible Bob that her ballot is valid. And, almost paradoxically, the way Alice can do this is by doing exactly what we just argued that no Bob can be allowed to do: choose the challenge c herself as a hash value on pk and pk'. This technique is usually attributed to and named after Fiat and Shamir[8][FS86].

Definition 24. *The Fiat-Shamir transformation of a Σ-protocol is the protocol in which Bob's choice of a challenge c is replaced by Alice computing the challenge as $c := H(y, b)$ where y is the value of which she is proving a preimage and b is her "commitment", the message that she would send to Bob immediately before getting his challenge. H is a cryptographic hash function with range $\{0, 1, \ldots, n-1\}$. This is a non-interactive proof of knowledge.*

[8] The attribution is not uncontested: others prefer to credit Blum with the technique [BR93].

To verify a proof $\pi = (y, b, c, d)$ you first check that $c = H(y, b)$. If this holds, check that $\phi(d) = b + c \cdot y$ for the function ϕ in question and accept the proof if this is the case.

Key generation in Helios. These are the exact steps that a Helios authority performs to generate her key share. Helios uses n-out-of-n threshold keys as we described eariler.

1. Obtain parameters (p, q, g) or agree these with the other authorities.
2. Generate a key pair by picking sk_i randomly from $\mathbb{Z}_q$ and setting $pk_i := g^{sk_i r} \pmod{p}$. Here i is some identifier.
3. Pick another keypair $(a, b = g^a \pmod{p})$ for the proof.
4. Compute $c := H(pk_i, a) \pmod{q}$ where H is a hash function (Helios uses SHA-256).
5. Compute $d := a + c \cdot sk_i \pmod{q}$.
6. Your public key component is pk_i and its proof of correctness is $\pi_i = (c, d)$. Your public key share is (pk_i, π_i).

The value a is omitted from the proof as the verifier can recompute it using the verification equation as $a = g^d/(pk_i)^c \pmod{p}$ and then check that $c = H(pk_i, a)$. This variation is equivalent to the one we gave above, i.e. zero-knowledge and a proof of knowledge, but saves one group element per proof.

4.4 Chaum-Pedersen: proving correct decryption

To prove that you have decrypted a ciphertext correctly, you need to show that you have produced a value d such that $d = a^{sk_i} \pmod{p}$ where sk_i is your secret key share satisfying $g^{sk_i} = pk_i \pmod{p}$, the value a is the first component of the ciphertext and pk_i is the key component of your public key share. Put another way, you have to show knowledge of an sk_i satisfying

$$a^{sk_i} = d \pmod{p} \wedge g^{sk_i} = pk_i \pmod{p}$$

All constants appearing in this formula (a, d, g, pk_i, p) are public. In the notation that we have just introduced, you have to show knowledge of

$$\text{a preimage } x \text{ of } (d, g) \text{ under } \phi(x) = (a^x \pmod{p}, g^x \pmod{p})$$

this function is conveniently also linear. Here we start to see why linearity and vector spaces are the correct way to understand Σ-protocols abstractly: for the finite field $\mathbb{F}_q$, our function signature is $\phi : \mathbb{Z}_q \to G^2$, mapping integers (1-dimensional vectors) into 2-dimensional vectors over the group G.

The protocol for this particular ϕ-function was invented by Chaum and Pedersen [CP92]. We give the exact steps to prove a decryption share correct:

1. Inputs: ciphertext (a, b), public key share pk_i, secret key share sk_i.
2. Pick a random r from $\mathbb{Z}_q$. Compute

$$(u, v) := \phi(r) = (a^r \pmod p), g^r \pmod p))$$

3. Compute a challenge as $c := H(pk_i, a, b, u, v)$.
4. Let $s := r + c \cdot sk_i \pmod q$.
5. Compute the decryption factor $d := a^{sk_i} \pmod p$.
6. Reveal d and the proof $\pi := ((u, v), s)$.

To check such a proof, your inputs are a, b, pk_i, u, v and s. Compute the hash value $c := H(pk_i, a, b, u, v)$ and check that

$$a^s = u \cdot d^c \pmod p \quad \wedge \quad g^s = v \cdot (pk_i)^c \pmod p)$$

There is an important difference between Schnorr's protocol and that of Chaum and Pedersen. In the former, Bob already knows that whatever Alice claims as her public key has a discrete logarithm — all group elements do, by definition. Alice is only trying to convince Bob that she knows the discrete logarithm of her public key. By contrast, in the Chaum-Pedersen protocol the focus is less on convincing Bob that you know how to decrypt but that you have done so correctly. Indeed, if a ciphertext decrypts to d but you claim some $d' \neq d$ instead, the pair (pk, d') will not lie in the image of ϕ so there will be no x with which you can convincingly run the protocol. For the interactive Chaum-Pedersen protocol, someone who has decrypted incorrectly cannot cheat (with more than $1/n$ probability) even if they have unlimited resources and can even take discrete logarithms. For the non-interactive protocol, the security analysis depends on the hash function but we still get the property that no realistic attacker can produce a proof of a false decryption. This property is called "soundness".

Proposition 25. *In a Σ-protocol following our construction, it is infeasible to produce a proof (whether interactive or non-interactive) for a value not in the image of the ϕ-function.*

4.5 DCP: proving that a vote is valid

We come to our third and final Σ-protocol. This one is for the voter to prove that she encrypted a valid vote in her ballot (namely 0 or 1), without revealing the vote. ElGamal encryption of a message m with random value r, in the exponential version used by Helios, can be expressed by the formula

$$c = \phi(m, r) := (g^r \pmod p), g^m pk^r \pmod p))$$

which is linear in m and r as a function with signature $\phi : (\mathbb{Z}_q)^2 \to G^2$ where g, p, q and pk are taken to be constants. This immediately yields a Σ-protocol to prove knowledge of your vote and randomness but does not prove that your vote m lies in a particular range. Let us consider how Alice would prove that she had voted for a particular value of m. If her ciphertext is $(a, b) = (g^r \pmod p), g^m pk^r \pmod p))$ then she could divide out g^m again to get $(a, b') = (a, b/g^m \pmod p))$ which is the image of the linear function

$$\phi' : \mathbb{Z}_q \to G^2, r \mapsto (g^r \pmod p), pk^r \pmod p))$$

In other words, to prove that (a, b) is a ciphertext for m Alice can prove that she knows a preimage r of $(a, b/g^m \pmod p))$ under the function ϕ'. This is of course exactly the protocol of Chaum and Pedersen with pk playing the role of the second basis (instead of a in our last discussion).

There is a general construction for Alice to prove that she knows a preimage of (at least) one of two linear functions for given images, without revealing which. This technique, also known as an OR or disjunctive proof, was first given by Cramer, Damgård and Schoenmakers in 1994 [CDS94]; like the AND construction underlying the Chaum-Pedersen proof it can be seen as a generic construction on Σ-protocols.

Given linear functions ϕ_0, ϕ_1 and values y_0, y_1 in their respective domains, to prove that she knows $x_0 : \phi_0(x_0) = y_0$ or $x_1 : \phi_1(x_1) = y_1$ Alice runs the following protocol.

- Start running the Σ-protocols for both functions.
- Get Bob to pick a single challenge c from $\{0, 1, \ldots, n-1\}$.
- For each of the two protocols, produce a new challenge c_i and a final value x''_i such that both protocols are correct individually and $c = c_1 + c_2 \pmod n$.

The trick is that Alice cheats and picks c and x'' first for the function where she does not have a preimage. The condition $c = c_1 + c_2$ where c is chosen by the challenger lets Alice cheat in one of the two protocols but not both. In more detail, here is the general construction.

1. For the value i where you know a preimage $x_i : \phi_i(x_i) = y_i$, pick a new pair $(x'_i, y'_i = \phi_i(x'_i))$ as you would to start the Σ -protocol for this function.
2. For the value j where you do not know a preimage, run the cheating protocol: pick c_j at random from $\{0, 1, \ldots, n\}$ and x''_j at random from the domain of ϕ_j. Then set $y'_j := \phi(x''_j) - c_j \cdot y_j$, where these operations are done in the vector space that contains the range of ϕ_j, i.e. $c_j \cdot y_j$ is scalar multiplication with the scalar c_j.
3. Send y'_0 and y'_1 to the challenger and obtain a c in return.
4. Set $c_i := c - c_j \pmod n$ and complete the protocol for ϕ_i by setting $x''_i := x'_i + c_i \cdot x_i$. These operations are done in the vector space that contains the range of ϕ_i.
5. Send c_0, c_1, x''_0, x''_1 to the challenger to complete the protocol.

From Bob's point of view, there are two Σ-protocols running in parallel:

1. Bob knows functions ϕ_0, ϕ_1 and claimed images y_0, y_1. He gets a pair of further values y'_0, y'_1 from Alice.
2. Bob chooses a single value c at random from $\{0, 1, \ldots, n-1\}$.

3. Alice sends Bob values c_0, c_1, x_0'', x_1''. Bob checks the following equations. The first two check the individual Σ protocols and the final one ensures that Alice can cheat on at most one of the protocols:

$$\begin{aligned} \phi_0(x_0'') &= y_0' + c_0 \cdot y_0 && (6) \\ \phi_1(x_1'') &= y_1' + c_1 \cdot y_1 && (7) \\ c &= c_0 + c_1 \pmod{n} && (8) \end{aligned}$$

The argument that Alice cannot cheat in both protocols is as follows. Suppose Alice knows neither a preimage of y_0 nor of y_1. If there are two values of c, c' for which she could both convince Bob then there must be some values c_0, c_1, c_0', c_1' that she could use to convince Bob, i.e. $c_0 + c_1 = c$ and $c_0' + c_1' = c'$ (all modulo n). But $c \neq c'$ so at least one of $c_0 \neq c_0'$ or $c_1 \neq c_1'$ must hold, which implies that Alice can already cheat in one of the two individual Σ-protocols on its own.

Of course this protocol can be made non-interactive with a hash function just like any Σ-protocol. The items that need to be hashed here are y_0, y_1, y_0', y_1' and any other constants appearing in the two protocols. Similarly, the same technique can be used for three or more functions — what Alice is proving in each case is that she knows at least one preimage, without revealing which. Applied to Chaum-Pedersen proofs, the resulting protocol is called disjunctive Chaum-Pedersen or DCP.

☼ **Proofs in Helios ballots.** In Helios, a voter uses this technique to prove that she either knows a random value r_0 with which she can do a Chaum-Pedersen proof that her ballot is an encryption of 0, or she knows a value r_1 with which she can do a Chaum-Pedersen proof that she has encrypted 1. The voter must produce one such proof for each ciphertext in her ballot.

If the election format demands that the voter choose a certain minimum/maximum number of options in a question (e.g. vote for at most one candidate) then the voter additionally takes the homomorphic sum of all her ciphertexts for the question and performs an additional DCP proof on the sum, showing that it is in the allowed range. This proof is known as the overall proof for the question.

Definition 26. *The following is the voter's protocol for proving that a ciphertext is an encryption of* 0 *or* 1*. The voter's inputs are the parameters* (p, q, g)*, the election public key* pk*, the voter's ciphertext* $(a, b) = (g^r, g^v \cdot pk^r)$*, her vote* $v \in \{0, 1\}$ *and the random value* r *that she used to encrypt her vote.*

If your vote is $v = 0$:

1. Simulate the protocol for proving $v = 1$. Pick c_1 randomly from $\{0, 1, \ldots, n\}$ and r''_1 from $\mathbb{Z}_q$ at random. Set

$$\begin{aligned} b' &:= b/g^1 \pmod p \\ a'_1 &:= g^{r''_1}/a^{c_1} \pmod p \\ b'_1 &:= pk^{r''_1}/(b')^{c_1} \pmod p \end{aligned}$$

2. Set up the proof that $v = 0$. Create a value r'_0 from $\mathbb{Z}_q$ at random and set

$$\begin{aligned} a'_0 &:= g^{r'_0} \pmod p \\ b'_0 &:= pk^{r'_0} \pmod p \end{aligned}$$

3. Get the challenge for the $v = 0$ proof. Compute

$$\begin{aligned} c &:= H(pk, a, b, a'_0, b'_0, a'_1, b'_1) \\ c_0 &:= c_1 - c \pmod n \end{aligned}$$

4. Complete the $v = 0$ proof. Compute

$$r''_0 := r'_0 + c_0 \cdot r \pmod q$$

5. Your proof π is the tuple

$$(a'_0, a'_1, b'_0, b'_1, c_0, c_1, r''_0, r''_1)$$

If your vote is $v = 1$:

1. Simulate the protocol for proving $v = 0$. Pick c_0 randomly from $\{0, 1, \ldots, n\}$ and r''_0 from $\mathbb{Z}_q$ at random. Set

$$\begin{aligned} a'_0 &:= g^{r''_0}/a^{c_0} \pmod p \\ b'_0 &:= pk^{r''_0}/b^{c_0} \pmod p \end{aligned}$$

2. Set up the proof that $v = 1$. Create a value r'_1 from $\mathbb{Z}_q$ at random and set

$$\begin{aligned} a'_1 &:= g^{r'_1} \pmod p \\ b'_1 &:= pk^{r'_1} \pmod p \end{aligned}$$

3. Get the challenge for the $v = 1$ proof. Compute

$$\begin{aligned} c &:= H(pk, a, b, a'_0, b'_0, a'_1, b'_1) \\ c_1 &:= c_0 - c \pmod n \end{aligned}$$

4. Complete the $v = 1$ proof. Compute

$$r''_1 := r'_1 + c_1 \cdot r \pmod q$$

5. Your proof π is the tuple

$$(a'_0, a'_1, b'_0, b'_1, c_0, c_1, r''_0, r''_1)$$

To verify such a proof, the following equations need to be checked.

$$g^{r''_0} = a'_0 \cdot a^{c_0} \pmod p \tag{9}$$

$$g^{r''_1} = a'_1 \cdot a^{c_1} \pmod p \tag{10}$$

$$pk^{r''_0} = b'_0 \cdot b^{c_0} \pmod p \tag{11}$$

$$pk^{r''_1} = b'_1 \cdot (b/g^1)^{c_1} \pmod p \tag{12}$$

$$c_0 + c_1 = H(pk, a, b, a'_0, b'_0, a'_1, b'_1) \pmod n \tag{13}$$

The Helios ballot format. Homomorphic ballots are possible not just for yes/no questions but for a number of voting/election setups including first-past-the-post, approval voting and top-k-of-n elections. All these formats have in common that a voter answers a question by ticking some (or all, or none) of a predefined set of checkboxes and the election result is essentially a list, for each box, of how many voters ticked this box. Homomorphic voting in the above sense cannot handle write-in votes or ranked (Instant runoff etc.) counts.

For example, in a first-past-the-post election for three candidates A, B and C, voters will be presented with three boxes — obviously labelled A, B and C — and must tick exactly one box each (or possibly none, to cast a blank vote). The election result is the number of votes that A, B and C each got, from which one can form the sum and determine the turnout and the percentage of votes that each candidate got.

Helios supports such elections: a ballot contains one ciphertext for each checkbox. These ciphertexts are encryptions of either 0 or 1. In addition, each ciphertext is accompanied by a proof that it really contains 0 or 1; these proofs are known as individual proofs. If the election specification sets limits on the numbers of boxes you can/must check, there is one further proof per ballot attesting to this known as the overall proof. A ballot can be composed of several structures as just described, allowing for multiple questions in a poll or multiple races in an election.

Attacks against the ballot format and ballot weeding. Cortier and Smyth [CS13] pointed out the following problem with bulletin board based elections. Suppose there are three voters, Alice, Bob and Eve. Alice and Bob both cast ballots. Next, Eve reads Alice's ballot off the board and submits a copy of it as her own ballot. The election result is now announced as 2 yes, 1 no: Eve knows that Alice must have voted yes and Bob no, since the two copied ballots must encrypt the same vote. However, in a truly private election, Eve should never be able to tell whether Alice votes yes and Bob no or the other way round, since these two scenarios both make the same contribution to the result. If the result is that everybody voted yes then Eve can deduce that Alice voted yes too, which is unavoidable — the problem with ballot copying is that Eve can find out more than she could by just observing the result.

A first reaction to this problem could be to introduce ballot weeding in the following sense: we reject any ballot that is an exact copy of a ballot already on the board. If ballots are non-malleable ciphertexts, this is actually sufficient — however, homomorphic ciphertexts can never be non-malleable as Eve can always `Add` an encryption of g^0 to an existing ciphertext. Eve will still know that the two ciphertexts encrypt the same vote but no-one else, even the decryptor, can tell such a "rerandomised" ciphertext from a genuine ballot by a voter who just happened to vote for the same choice as Alice.

We can solve this problem and the problem of Eve voting for g^2 in one go by adding a non-malleable zero-knowledge proof to each ciphertext. Ballot weeding will now reject any ballot that shares a proof with a previous ballot.

In fact we even fix one more problem that Cortier and Smyth identified with the original ballot format. Consider a poll for choices A, B and C so ballots take the form

$$(c_A, \pi_A, c_B, \pi_B, c_C, \pi_C, \pi_O)$$

where c_A is the ciphertext for choice A, π_A is the individual proof that c_A is well-formed and π_O is the overall proof (that at most one of c_A, c_B, c_C encrypts a 1). If the above is Alice's ballot, Eve can submit the following modified ballot:

$$(c_B, \pi_B, c_A, \pi_A, c_C, \pi_C, \pi_O)$$

The result is that Eve has swapped the A- and B-components of Alice's ballot around but she knows exactly what the relations between the original and modified ballot are and can use this knowledge to attack Alice's privacy. In Helios version 3, after you submitted a ballot, the hash of your ballot was sent to you as a confirmation value and the hashes of all ballots were displayed on a "short board", with the "full board" of all ballots also available. The point here is that Eve's ballot will have a completely different hash value to Alice's and while Helios prevented Eve from submitting a ballot with the same hash value as Alice's (i.e. making an exact copy), this modified ballot was accepted by Helios without complaint. It could be detected by auditing the full board but code for this was not available in Helios at the time.

4.6 Ballot privacy

With zero-knowledge proofs in ballots, everyone can be assured that voters are casting ballots for valid votes. To model what security level this yields, we give the full ballot privacy game that protects against dishonest voters. It differs from the previous notion of privacy against observers in that the attacker can declare any voters she likes to be dishonest and provide them with arbitrary ballots. This accounts for both attempts at making "bad ballots" and ballot-copying or modifying existing ballots and resubmitting them as your own.

Definition 27. *A voting scheme has ballot privacy if no attacker can do better in the following game than guess at random (with probability* 1/2*).*

Setup *The game picks a bit b at random and keeps it secret. The game then sets up the voting scheme and plays the voters, authorities and bulletin board.*

Moves *Once for each voter, the attacker may perform one of two moves.*

- *The attacker declares this voter to be honest. She may then choose two votes v_0 and v_1. The game writes down both votes. If $b = 0$, the game lets the voter vote for v_0; if $b = 1$ the game lets the voter vote for v_1.*
- *The attacker declares this voter to be dishonest and may provide an arbitrary ballot b for the voter, which the game processes.*

The attacker may ask to look at the bulletin board at any point in the game. When all voters have voted, the game individually decrypts all dishonest voters' ballots. It then gives the attacker the result computed as follows: for each honest voter, it takes the first (v_0) vote and for each dishonest voter, it takes the vote obtained by decrypting the submitted ballot.

Winning conditions *At any point in the game, the attacker may submit a guess for b. This ends the game immediately. The attacker wins if her guess is correct.*

4.7 Achieving ballot privacy with non-malleability

We sketch how one would show that minivoting with zero-knowledge proofs added to key shares, ballots and decryption shares achieves ballot privacy. The main issue is that the attacker may derive one of her own ballots from that of an honest voter. Bernhard et al. [BPW12a,BS13] have explored the connection between ballot privacy and ballot independence — the property that there are no "unexpected" relations between the votes in ballots of different voters — and concluded that the two are essentially the same, i.e. to achieve privacy one must ensure that ballots are independent.

The model that we use to capture independence is that of non-malleable encryption. This can be expressed by taking the IND-CPA game and letting the attacker, once in the game, produce any number of ciphertexts she likes and ask the game to decrypt them. If the attacker has already obtained a challenge ciphertext, she cannot ask for the challenge ciphertext to be decrypted however.

Definition 28. *An asymmetric encryption scheme E is non-malleable if no attacker can win the following game with better probability than* $1/2$*, the probability of guessing at random.*

Setup *The game creates a keypair* $(pk, sk) \leftarrow \texttt{KeyGen}()$ *and gives the attacker the public key* pk*. The game also picks a bit b randomly from* $\{0, 1\}$ *and keeps this secret.*

Moves *Once in the game, the attacker may pick a pair of messages* m_0 *and* m_1 *of the same length and send them to the game. The game encrypts* $c \leftarrow \texttt{Encrypt}(pk, m_b)$ *and returns this to the attacker.*
Once in the game, the attacker may send the game a list of any number of ciphertexts $(c_1, \ldots, c_n)$*. If the attacker has already obtained a challenge ciphertext c, she must not include this ciphertext in her list. The game decrypts each ciphertext in the list and sends the attacker back the decrypted messages.*

Winning conditions. *The attacker may make a guess at b which ends the game. The attacker wins if she guesses correctly.*

This notion captures non-malleability in the following sense. Suppose that a scheme is malleable in that the attacker can take a ciphertext c and somehow turn it into a different ciphertext c' for a message that has some relation to the message in c — for example, the two ciphertexts encrypt the same message. Then the attacker can win the non-malleability game as follows:

– Pick any messages m_0, m_1 and ask for a challenge ciphertext c.
– Turn c into c' and ask for c' to be decrypted.
– If the decrypted message is m_0, guess $b = 0$, otherwise guess $b = 1$.

The exact form of the game has been shown be Bellare and Sahai [BS99] to imply that the attacker is unable to construct any number of ciphertexts that have an "unexpected" relation with the challenge message. By "unexpected", we mean that we are glossing over the following problem: the attacker can always make two fresh ciphertexts c_0 and c_1 for m_0 and m_1 herself, this pair (c_0, c_1) will then have the relation "one of the two encrypted messages matches that in the challenge ciphertext". Informally, what we want is that the attacker cannot construct any relation that helps her decide what is in the challenge ciphertext. The formal argument can be found in the cited paper [BS99].

ElGamal on its own is homomorphic and therefore not non-malleable: an attacker can always add an encryption of 0 to a challenge ciphertext to get a new ciphertext for the same message. ElGamal with a Σ-protocol based zero-knowledge proof however is non-malleable. As proven by Bernhard, Pereira and Warinschi [BPW12b]:

Proposition 29. *The encryption scheme obtained by combining ElGamal with a DCP proof that the encrypted message lies in a certain range is non-malleable.*

Side note — CCA security. Non-malleability is a strictly stronger notion of security than IND-CCA. There is a further, even stronger notion that is often cited as "the correct notion" for security of asymmetric encryption called CCA, security against "chosen-ciphertext attacks". In this notion, the attacker can use the decryption move as many times as she likes, as long as she never asks to decrypt the challenge ciphertext. The game is usually presented in a form where the attacker asks one decryption at a time instead of a list at once which makes no difference (whereas the fact that the attacker only gets one decryption move is a central part of the non-malleability notion). For the purposes of building ballot private voting schemes, non-malleability is sufficient.

Ballot privacy of minivoting. To obtain ballot privacy we need two ingredients. The first is non-malleability which prevents Eve from submitting a modified version of Alice's ballot as her own. The second ingredient is ballot weeding, a way to catch Eve if she tries to submit an exact copy of Alice's ballot. In a scheme with ballot weeding, whenever anyone submits a ballot, the bulletin board checks the ballot against all previous ballots and rejects it if it finds a relation. If ballots are non-malleable, this relation can be "the ballot is an exact copy of a previous one". This leads to the following proposition attesting to ballot privacy of minivoting with proofs.

Proposition 30. *Minivoting with non-malleable ballots and ballot weeding for exact copies has ballot privacy.*

The non-malleability of ElGamal with DCP gives non-malleable ballots if each ballot contains only a single ciphertext. Otherwise, as explained in the section on the Helios ballot format, Cortier/Smyth attacks [CS13] in which the ciphertexts in a ballot get permuted might still be possible. The correct way to weed ballots containing multiple ElGamal+DCP ciphertexts is to check for repeated proofs. This extends to show ballot privacy of the version of Helios that we present in this work.

Proposition 31. *Minivoting with ElGamal+DCP ciphertexts/proofs has ballot privacy if the ballot weeding rejects any ballot re-using a proof from an earlier ballot.*

☼ **On ballot privacy in Helios.** The original Helios security result by Bernhard et al. [BC+11] showed that Helios would have ballot privacy if it employed CCA secure encryption. The combination of ElGamal and a DCP proof (or any other Σ-protocol with special soundness w.r.t the encryption randomness r and the Fiat-Shamir transformation for the challenge) has not been shown CCA secure in a widely accepted model and indeed there is evidence suggesting that it is not CCA secure, although no proof of this has been published to date.

CCA security is not necessary for ballot privacy however: the latest proofs [BPW12a,BPW12b] achieve privacy from non-malleability alone and ElGamal + DCP definitely is non-malleable.

However, Bernhard, Pereira and Warinschi [BPW12b] have also shown that the currently available version 3 of Helios does not perform the zero-knowledge proofs of knowledge correctly, as a result of which Helios currently does not satisfy our notion of ballot privacy and can even be attacked in practice. The Helios described in this paper is a fixed version; the Helios authors have assured us that the upcoming Helios version 4 will contain fixed proofs.

4.8 ☼ Helios

We now have all the building blocks to describe the Helios electronic voting scheme. The idea to create a voting scheme from homomorphic encryption and zero-knowledge proofs comes from a 1997 paper by Cramer, Gennaro and Schoenmakers [CGS97]; Helios is an implementation of this idea.

To generate an election or poll, the authorities agree on the questions and options and generate parameters (p, q, g). They each generate an ElGamal keypair with a Schnorr proof to obtain their threshold keys. One of the authorities then combines the public key and publishes the election specification, parameters, public key shares and the public key itself on a bulletin board.

To vote, voters obtain and check the public key. By check, we mean that they check the authorities' Schnorr proofs and re-run the key combination step.

For each option “checkbox”, the voter encrypts g^0 to leave the box empty and g^1 to check it. She accompanies each ciphertext with a DCP proof that she has indeed encrypted either g^0 or g^1. If the election specification restricts the maximum or minimum number of boxes that a voter must check for a question, she also makes an overall proof that the number of boxes checked lies in the allowed range. The voter’s ballot contains her ciphertexts, individual proofs and if required and overall proof for each question; she posts this ballot to the bulletin board. Typically, a board will require voters to authenticate themselves before accepting any ballots. The board never sees the actual votes however.

To tally an election, the authorities check all proofs in the ballots and discard any ballots with invalid proofs. Further, they reject any ballot that has copied a proof from an earlier ballot[9]. These checks can also be done by the board itself when ballots are submitted so invalid ones never end up on the board but for security reasons, the authorities always need to repeat these checks to protect against a dishonest board collaborating with a dishonest voter. One authority sums all ciphertexts for each individual question and posts the sum-ciphertexts back on the board. Each authority then produces a decryption share for each sum-ciphertext and posts this to the board. One authority completes the tally by combining the decryption shares for each sum and posting this on the board and computing the result in the correct format, for example each option’s count as a percentage of the total number of ballots cast.

5 Verifiability

Anyone can verify a Helios election. Taking the board of a completed election, they should perform the following steps.

1. Check that the Schnorr proofs on the public key shares are correct and that the public key was combined correctly.
2. Check that each ballot meets the election format (correct number of ciphertexts and proofs) and that all proofs in the ballots verify, or that all invalid ballots have been marked as such and excluded from the tally.
3. If ballots contain voter information, check that this is consistent, i.e. only eligible voters have voted and no-one has cast more votes than allowed.
4. Check that no ballot re-uses a proof from an earlier ballot, or that all ballots that do so have been marked as invalid.
5. Recompute the sum-ciphertexts and check that they are correct. In particular, the sums should be over only those ballots not marked as invalid.
6. Check the zero-knowledge proofs on all decryption shares.
7. Recombine all decryptions and check that they are correct.
8. Check that the announced result matches the decryptions.

From this procedure, we can check whether Helios meets the following verifiability criteria.

[9] At least, this is the way Helios should check ballots and will do in a future version. The current version (v3) is still susceptible to some ballot-copying attacks.

Individual verifiability Each voter can save a copy of her ballot and check that it is included in the final bulletin board. This property is satisfied.

Eligibility verifiability This depends on the election setting as eligibility information must be available to check this (such as a list of all eligible voters) and a method is required to verify that ballots really come from who they claim to come from. If voters are all equipped with digital signature keypairs and the public keys are available in a public directory, voters could be asked to sign their ballots as part of the authentication process.

Universal verifiability This was the key design aim of Helios and is satisfied. All the steps in the protocol that use secret information are protected by zero-knowledge proofs: key generation, ballot creation (the vote is secret) and decryption. These proofs are available on the bulletin board for anyone to audit.

Ballot verifiability Ballots are protected by zero-knowledge proofs attesting to the fact that they contain correct votes. These votes are available on the bulletin board to audit. This property is satisfied[10].

As the reader will have noted, this too is an informal analysis — no formal model for verifiability of Helios-type schemes[11]exists yet, in contrast to ballot privacy. Creating a suitable model and proving Helios secure in such a model (or finding an attack) is one of the open challenges of cryptographic voting.

6 Mix-nets

In this section we give a brief overview of mix-nets, the other main technique (beside blind signatures and homomorphic encryption) to achieve private and verifiable cryptographic voting schemes. An advantage of mix-nets over homomorphic voting is that they can handle arbitrary ballot formats including write-in votes.

Suppose that every voter encrypts their vote with normal ElGamal (not the exponential variant) and posts the ciphertext on the board, along with some identification information (or even a digital signature) to ensure eligibility. Normal ElGamal can handle arbitrary bitstrings (of a fixed length) as messages as long as the basic group is chosen cleverly[12]. Since we can no longer do homomorphic tallying, we need another way to anonymize ballots. Here is one: a trusted authority takes all ballots, randomly shuffles them and re-encrypts each

[10] This analysis refers to the version of Helios described in this work — the current (v3) Helios does not satisfy ballot verifiability due to a bug in the implementation of the proofs.

[11] There is a model for verifiability by Kuesters et al. [KTV11] but it coniders voting schemes from a highly abstract point of view and has, to our knowledge, never been successfully applied to a fully cryptographic scheme such as Helios.

[12] The kind of groups we presented in this work are not suitable for this kind of scheme, since our messages have to start out as group elements. ElGamal in groups defined over elliptic curves does work and is typically faster (for the same key-size) too.

one, that is for a ciphertext (a, b) the authority generates a random r from $\mathbb{Z}_q$ and sets $(a', b') := (a \cdot g^r \pmod p, b \cdot pk^r \pmod p)$. These shuffled ciphertexts contain the same set of votes as the originals but the link between voter and ballot is hidden, so the shuffled ciphertexts can be decrypted individually.

This also removes the need for zero-knowledge proofs to assert correctness of ballots: if someone has encrypted a 2 in an 0/1 question, since ballots are decrypted individually such invalid ballots can be discarded individually too. This technique does not prevent the need for non- malleable encryption to combat ballot-copying however so ElGamal will still need some kind of proof protecting the ciphertexts.

If one does not have a trustworthy authority, once can take several authorities who each shuffle and re-encrypt all ballots in turn. As long as any one of the authorities is honest, this protects voters' privacy from all other authorities. The system is also resilient to mixers failing: a mixer who does not complete a mix can be simply replaced by another.

Unfortunately, the scheme as described is not verifiable and in fact completely insecure against a cheating mixer substituting ballots of her own instead of returning a shuffled, re-encrypted version of her inputs. In this way, a dishonest mixer can arbitrarily manipulate the election results. The solution to this problem is clear: zero-knowledge proofs!

In a real mix-net, each mixer takes a list of ciphertexts $(c_1, \ldots, c_n)$ as input and outputs a mix $(c'_1, \ldots, c'_n)$ together with a proof π that the outputs are a mix of the inputs. Different mix-nets differ in the kind, size and efficiency of the proof: proofs of correct mixing are typically quite expensive to compute. Some mix-nets offer an "online/offline" mode where most of the work in computing a mix and a proof can be done "offline" before or during the election. This work involves choosing random values $r_1, \ldots, r_n$ for some upper bound n of the number of ballots expected and a permutation p on the set $\{1, \ldots, n\}$, then pre-computing as much of the proof as possible without the actual ciphertexts. At the end of the election, the pre-computed values can then be applied more efficiently to the ciphertexts forming the ballots in an "online" phase, returning the permuted and rerandomised ballots and the proof of correct mixing.

7 Conclusion

We have presented several cryptographic schemes and techniques for voting, introduced the building blocks from which they are constructed and given an overview of how cryptographers work with security properties and models. The state of the art is that we have (albeit imperfect) models for ballot privacy and proofs for that the (fixed) Helios scheme satisfies these models; while these models and proofs were developed, their authors discovered subtle problems with the existing Helios which shows the importance of a detailed and formal approach to security. While Helios is widely believed to be verifiable, which is one of its design goals if not the key selling point, a formal model in which this claim can be justified has not been published at the time of writing.

References

[DH76] W. Diffie and M. Hellman. New Directions in Cryptography. In: IEEE Transactions on Information Theory, vol. 22, no. 6, pages 644–654, 1976.

[RSA78] R. Rivest, A. Shamir and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. In: Communications of the ACM vol. 21.2, pages 120–126, 1978.

[C85] D. Chaum. Security without Identification: Transaction Systems to make Big Brother obsolete. In: Communications of the ACM vol. 28.10, October 1985.

[E85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In: IEEE transactions on information theory, Pages 469-472, Volume 31, 1985.

[FS86] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology — CRYPTO '86, pages 186–194, 1986.

[S91] C. P. Schnorr. Efficient signature generation for smart cards. In: Journal of Cryptology, Volume 4, Pages 161-174, 1991.

[CP92] D. Chaum and T. P. Pedersen. Wallet Databases with Observers. In: Advances in Cryptology — CRYPTO' 92, LNCS 740, pages 89–105, 1992.

[FOO92] A. Fujioka, T. Okamoto and K. Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In: Advances in Cryptology – AUSCRYPT' 92, LNCS 718, pages 244–251, 1992.

[BR93] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: Proceedings of the 1st ACM conference on Computer and communications security (CCS '93), pages 62–73, 1993.

[CDS94] R. Cramer, I. Damgård and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: Advances in Cryptology — CRYPTO '94, LNCS 839, pages 174–187, 1994.

[C96] R. Cramer. Modular Design of Secure yet Practical Cryptographic Protocols. PhD thesis, University of Amsterdam, 1996.

[CGS97] R. Cramer, R. Gennaro and B. Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In: Advances in Cryptology — Eurocrypt '97, LNCS 1223, pages 103–118, 1997.

[SG98] V. Shoup and R. Gennaro. Securing Threshold Cryptosystems Agains Chosen-Ciphertext Attack. In: Advances in Cryptology (Eurocrypt '98), LNCS 1403, pages 1–16, 1998.

[P99] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Advances in Cryptology (Eurocrypt '99), LNCS 1592, pages 223–238, 1999.

[BS99] M. Bellare and A. Sahai. Non-Malleable Encryption: Equivalence between Two Notions, and an Indisinguishability-Based Characterization. In: Advances in Cryptology — CRYPTO '99, LNCS 1666, pages 519–536, 1999.

[BNPS03] . Bellare, C. Namprempre, D. Pointcheval and M. Semanko. The One-More-RSA-Inversion Problem and the ASecurity of Chaum's Blind Signature Scheme. In: J. of Cryptology, vol. 16, no. 3, pages 185–215, 2003.

[BBS04] D. Boneh, X. Boyen and H. Shacham. Short Group Signatures. In: Advances in Cryptology (Crypto '04), LNCS 3152, pages 41–55, 2004.

[BC+11] D. Bernhard, V. Cortier, O. Pereira, B. Smyth and B. Warinschi. Adapting Helios for Provable Ballot Secrecy. In: Proceedings of ESORICS '11, LNCS 6879, pages 335–354, 2011.

[US11] D. Schröder and D. Unruh. Security of Blind Signatures Revisited. eprint, report 2011/316.

[KTV11] R. Ksters, T. Truderung and A. Vogt. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. IEEE Symposium on Security and Privacy (S&P '11), IEEE Computer Society, 2011.

[BPW12a] D. Bernhard, O. Pereira and B. Warinschi. On Necessary and Sufficient Conditions for Private Ballot Submission. Eprint, eprint.iacr.org/2012/236

[BPW12b] D. Bernhard, O. Pereira and B. Warinschi. How Not to Prove Yourself: Pitfalls of Fiat-Shamir and Applications to Helios. In: Advances in Cryptology — Asiacrypt '12, LNCS 7658, pages 626–643, 2012.

[BS13] D. Bernhard and B. Smyth. Ballot secrecy and ballot independence coincide. In: Computer Security — ESORICS '13, LNCS 8134, pages 463–480, 2013.

[CS13] V. Cortier and B. Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. In: Journal of Computer Security, volume 21(1), pages 89–148, 2013.

[B14] D. Bernhard. Zero-Knowledge Proofs in Theory and Practice. PhD thesis, University of Bristol, 2014.