Secrecy and Verifiability: An Introduction to Electronic Voting

Paul Keeler[1] and Ben Smyth[2]

[1]School of Mathematics and Statistics, University of Melbourne, Melbourne
[2]Independent Researcher

February 14, 2026

Abstract

Democracies are built upon secure and reliable voting systems. Electronic voting systems seek to replace ballot papers and boxes with computer hardware and software. Proposed electronic election schemes have been subjected to scrutiny, with researchers spotting inherent faults and weaknesses. Inspired by physical voting systems, we argue that any electronic voting system needs two essential properties: ballot secrecy and verifiability. These properties seemingly work against each other. An election scheme that is a complete black box offers ballot secrecy, but verification of the outcome is impossible. This challenge can be tackled using standard tools from modern cryptography, reaching a balance that delivers both properties.

This tutorial makes these ideas accessible to readers outside electronic voting. We introduce fundamental concepts such as asymmetric and homomorphic encryption, which we use to describe a general electronic election scheme while keeping mathematical formalism minimal. We outline game-based cryptography, a standard approach in modern cryptography, and introduce notation for formulating elections as games. We then give precise definitions of ballot secrecy and verifiability in the framework of game-based cryptography. A principal aim is introducing modern research approaches to electronic voting.

1 Introduction to voting systems

Any democratic state needs to be able to offer an election, but what is an election and what properties should it have to help deliver democracy? Several properties emerge from democratic principles. Two of these appear fundamentally at odds. Voters need assurance that their votes are counted accurately, yet revealing how any individual voted would undermine the democratic process itself. Understanding these properties helps us see what is needed in building an *electronic voting system*, which aims to replace the traditional voting setup of ballot papers and boxes with some combination of computer hardware and software.

An election is considered, essentially, a decision-making procedure for choosing representatives [2, 55, 74, 91]. Intergovernmental organizations such as the United Nations recommend voting systems that ensure that voting be done by voters so that each voter has equal influence over the result [81, 82, 109]. In an election, voters must be able to cast votes with what we call *free-choice*, meaning all voters can vote for any election candidates they desire without fear of repercussions. Another crucial property is that nobody can alter cast votes or create false votes, without leaving evidence of such *undue influence*. Electronic voting systems are already emerging,

but unfortunately, these voting systems are routinely broken in ways to reveal that they violate free-choice [21,53,70,106,111,112] or permit undue influence [21,24,61,70,108].

1.1 Ballot secrecy

To give voters in democratic elections free-choice, a voting system needs to offer *ballot secrecy*, preventing the voters suffering from their cast votes. The secret ballot has a long history, being used in ancient Greece and Rome. The modern system is a relatively recent concept, first implemented in the Australian colonies in 1856 during British colonial rule, and subsequently adopted by Britain, the United States, and other democracies. The seemingly simple yet revolutionary change was to make ballots identical. Previously each political party could produce its own ballot paper, whose style and size clearly revealed anybody's voting intentions. A voting system with this lasting innovation is often called the *Australian system*, which demands that votes be marked on uniform ballots in isolated polling booths and then deposited in ballot boxes [22].

By granting ballot secrecy, the Australian system can assure that only voters vote and that they do so with free-choice. In this paper, we employ a definition of ballot secrecy introduced by Smyth [96].

- Ballot secrecy. A voter's vote is not revealed to anyone.

To be clear, this definition of ballot secrecy means that not even the organizers of the election can see the vote of any voter. We will return to this definition later in Section 10.1.

1.2 Verifiability

A voting system also needs *verifiability*, which is a way to prove that no undue influence has occurred. To provide this traditionally, election monitors or observers can check that only one ballot paper is distributed to each eligible voter who then only deposits one paper, without papers coming from other sources. The observers can then discard incomplete or tarnished ballot papers, with the remaining papers accurately corresponding to the election outcome. But any assurance of verifiability is limited by the monitoring ability of the observers [16,64,80].

The property of verifiability can be considered in a more specific manner. In the setting of electronic voting, Smyth, Frink and Clarkson [103] introduced a definition for *universal verifiability*, which is designed for anyone to check that no undue influence has occurred in the election and the election outcome is the true one.

- Universal verifiability. Anyone can check whether an outcome corresponds to votes expressed in collected ballots.

Universal verifiability ensures that the election outcome reflects the collected votes. But it is not enough for a voter to merely cast a vote and to assume it is properly collected, because somebody may discard or modify the vote. We argue that voters must be able to uniquely identify *their* ballots. Consequently, Smyth, Frink and Clarkson [103] also introduced the notion of *individual verifiability*.

- Individual verifiability. A voter can check whether their ballot is collected.

1.3 Voting: a fundamental balance

The two essential properties of ballot secrecy and verifiability seem to compete against each other. It is easy to create a trivial voting system that adheres to one of these properties. For example, if all voters attach their names to their ballot papers (being written on a piece paper or encoded electronically), we can immediately verify that all votes were cast by the proper parties, but then ballot secrecy is gone.

On the surface these two conflicting properties may seem to destroy the chances of any electronic voting scheme possessing both of them in fullness: too much of one renders the other one not possible. We later see how this problem has repeatedly appeared in the research on electronic voting, due to the definitions of properties being too strong. But we argue that it is possible to arrive at a subtle balance, rather than a trade-off, between these two properties, resulting in voting systems that offer both secrecy and verifiability.

1.4 Terminology

We use *election scheme* to refer to the cryptographic protocol, and *voting system* to refer to implementations. Of course a secure election scheme does not guarantee a secure voting system.

The terms *secrecy* and *privacy* occasionally appear as synonyms in the literature; we prefer ballot secrecy to avoid confusion with other privacy notions. We discuss related concepts such as *coercion resistance* and *receipt-freeness* in Section 10.5.

Structure of article

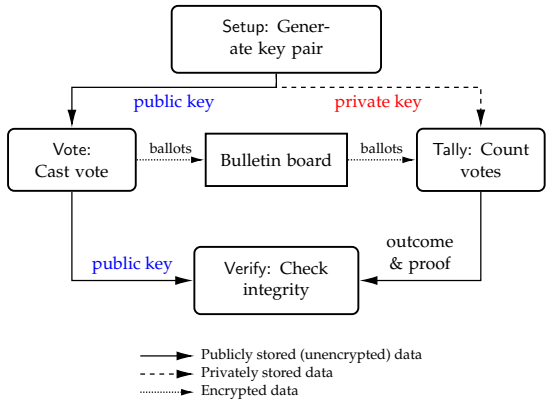
The tutorial is organized as follows. Section 2 previews an election scheme, giving a high-level picture before the technical details. Section 3 covers cryptography basics, including negligible functions and encryption. Section 4 introduces game-based cryptography. Section 5 summarizes the notation used throughout. Section 6 gives an overview of election schemes. Section 7 defines the formal syntax for election schemes. Section 8 surveys attacks on electronic voting and their countermeasures. Section 9 formalizes individual and universal verifiability. Section 10 formalizes ballot secrecy and proves sufficient conditions. Section 11 presents case studies of Helios, Helios Mixnet, and Belenios. Section 12 summarizes lessons learnt and concludes. Readers familiar with cryptographic foundations may wish to skim or skip Sections 3–5 and proceed directly to Section 6.

2 An election scheme at a glance

Before introducing the cryptographic tools that make electronic voting possible, it helps to see what an election scheme actually looks like. Many election schemes have been proposed; as a running example, we use one due to Smyth, Frink and Clarkson [103], which we return to throughout the tutorial. Figure 1 shows its high-level structure, built from four algorithms.

The algorithm Setup generates a pair of keys: a *public key*, which is shared openly, and a *private key*, which must be kept secret. Votes are encrypted using the public key, meaning anyone can create an encrypted vote, but only the holder of the private key can decrypt it. The algorithm Vote takes a voter's choice and the public key, and produces an encrypted ballot that reveals nothing about the vote inside. These encrypted ballots are posted to a *bulletin board*, which is a public, append-only log that anyone can read but nobody can secretly alter. Once voting ends, the algorithm Tally uses the private key to compute the election outcome and produce a proof that the

Figure 1 Overview of an election scheme with four algorithms. Setup generates a key pair; the public key is used by Vote to create encrypted ballots, while the private key is used by Tally to decrypt. Ballots flow through the bulletin board (dotted arrows indicate encrypted data). Verify checks the election outcome using public information. A detailed version appears in Figure 5.



count is correct. Finally, Verify allows anyone to check this proof using only public information: the public key, the ballots on the bulletin board, and the claimed outcome. No private key is needed to verify.

The rest of this tutorial makes all of this precise. We begin with the cryptographic building blocks, such as encryption, key generation, and related concepts, before returning to election schemes with formal definitions and security analysis.

3 Cryptography basics

Delivering a voting system that promises both ballot secrecy and verifiability is a daunting task, but fortunately, the tools from modern cryptography offer a way. We will now cover, in a rather informal fashion, some basic concepts from cryptography under the assumption that the reader has some fundamental knowledge in mathematics, logic, and algorithms. We will link concepts in cryptography to election schemes. Readers familiar with the fundamentals of cryptography can skip this section.

Historically cryptography focused on ways of sending messages between two parties without anyone else being able to read the messages. In the cryptography literature, the two parties are often called Alice and Bob. There are other named characters appearing in cryptography narratives, with a prominent figure being Eve the eavesdropper, who wants to listen into communications between Alice and Bob. If Alice wants to send Bob a secret message, she can, for example, simply mangle or encrypt it in some way, making it seemingly unrecognizable, and then tells Bob how she did that. Then Bob can reverse Alice's encryption step by decrypting the message.

3.1 Symmetric encryption with private keys

Encryption transforms a message, called the *plaintext*, into an unreadable form called the *ciphertext*. To decrypt, one needs a secret *key*. In *symmetric* (or *private-key*) cryptography, the same key is used for both encryption and decryption. If we write $\mathsf{Enc}(k, m)$ for encrypting message m with key k, then decryption satisfies $\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m$. The limitation of symmetric cryptography is that

Alice must somehow share the secret key with Bob while keeping it hidden from Eve, which is a challenge that asymmetric cryptography addresses.

3.2 Asymmetric encryption with private and public keys

To avoid Alice having to give Bob the private key, modern day cryptography uses a *key pair* consisting of one key pk for only encrypting plaintexts and another key sk for only decrypting ciphertexts. If Bob wants to receive messages from Alice, he generates a pair of keys (pk, sk), respectively called the *public key* and the *private key*. He then puts the public key pk somewhere it can be accessed by Alice (and anyone else), while keeping the private key sk secret. Alice uses the public key pk to encrypt her message and sends it to Bob who then uses his private key sk, which only he knows, to decrypt the encrypted message. Alice and everyone else knows the public key pk, but they never know the corresponding private key sk kept secret by Bob. It is computationally hard (or practically impossible in implementation) to infer sk from pk, so only the holder of the private key sk can decrypt Alice's message.

One key for encrypting, another for decrypting. This revolutionary concept is called *asymmetric* or *public-key* cryptography. Mathematically, we can write asymmetric encryption as

$$c = \mathsf{Enc}(pk, m)$$

where c is again a ciphertext and Enc is an algorithm representing asymmetric encryption. Decrypting the ciphertext c with sk can then be written as

$$\mathsf{Dec}(sk, c) = \mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m\,,$$

where Dec is the decryption algorithm that uses the private key sk. But after using the public key for encryption, it cannot be used for decryption, implying $\mathsf{Dec}(pk, c) \neq m$.

More formally, we can interpret the encryption algorithm $\mathsf{Enc}(pk, m)$ as a function that maps from the space of plaintexts $\mathcal{M}$ to the space of ciphertexts $\mathcal{C}$, while the encryption algorithm $\mathsf{Dec}(pk, c)$ maps from $\mathcal{C}$ to $\mathcal{M}$. A more precise description of asymmetric encryption is located in Section C of the appendix.

3.3 Examples of asymmetric cryptography

It would be difficult to overstate the importance of asymmetric cryptography in modern computing. Since the 1970s, researchers have proposed various public-key schemes. The most famous is RSA, named for its inventors Rivest, Shamir, and Adleman [89]. RSA exploits a fundamental asymmetry: it is easy to multiply two large prime numbers together, but incredibly hard to factor the product back into its prime components. This asymmetry, meaning easy in one direction, hard in reverse, is what makes public-key cryptography possible.

Other important schemes include Diffie–Hellman and ElGamal, based on the difficulty of the discrete logarithm problem. We will encounter ElGamal in the Helios voting system. Alternative schemes based on lattices, codes, and other structures are also an active area, particularly for post-quantum security (Section 10.5).

Example 1 (ElGamal encryption)**.** *We illustrate ElGamal with small numbers. Let $p = 23$ be a prime and $g = 5$ a generator of the multiplicative group modulo p.*

Key generation. *The key holder chooses a random private key $sk = 6$ and computes the public key*

$$pk = g^{sk} \bmod p = 5^6 \bmod 23 = 8.$$

The public parameters $(p, g, pk) = (23, 5, 8)$ are published; the private key $sk = 6$ is kept secret.

Encryption. *Suppose a voter wishes to encrypt a vote for candidate* 2*. In the variant of ElGamal used for voting (sometimes called* exponential ElGamal*), the voter encrypts g^m rather than m directly. With vote $m = 2$ and random value $r = 3$, the ciphertext is the pair (c_1, c_2):*

$$c_1 = g^r \bmod p = 5^3 \bmod 23 = 10,$$
$$c_2 = g^m \cdot pk^r \bmod p = 5^2 \cdot 8^3 \bmod 23 = 2 \cdot 6 \bmod 23 = 12.$$

The ciphertext $(10, 12)$ is posted to the bulletin board. A different random r would produce a different ciphertext for the same vote. This is why ElGamal encryption is probabilistic*, and why two ballots for the same candidate look different.*

Decryption. *The key holder recovers g^m using the private key:*

$$g^m = c_2 \cdot (c_1^{sk})^{-1} \bmod p = 12 \cdot (10^6)^{-1} \bmod 23 = 12 \cdot 6^{-1} \bmod 23 = 12 \cdot 4 \bmod 23 = 2.$$

Because the number of candidates is small, the key holder recovers m by checking a short table: $g^1 \bmod 23 = 5$, $g^2 \bmod 23 = 2$, $g^3 \bmod 23 = 10, \ldots$ The decrypted value matches g^2, so $m = 2$, confirming the voter voted for candidate 2*.*

Homomorphic property. *Suppose a second voter encrypts a vote for candidate* 1 *(so $g^{m'} = g^1 = 5$) with randomness $r' = 5$, obtaining ciphertext $(c_1', c_2') = (20, 11)$. Multiplying ciphertexts component-wise:*

$$(c_1 \cdot c_1' \bmod 23,\ c_2 \cdot c_2' \bmod 23) = (10 \cdot 20 \bmod 23,\ 12 \cdot 11 \bmod 23) = (16, 17).$$

Decrypting this combined ciphertext yields $g^{m+m'} = g^3 = 10$, encoding the sum $m + m' = 3$, the total of both votes, without ever decrypting either ballot individually. This is how homomorphic tallying works in Helios.

3.3.1 Decisional Diffie–Hellman (DDH) assumption

The security of both Diffie–Hellman key exchange and ElGamal encryption rests on the *Decisional Diffie–Hellman (DDH) assumption*. Informally, DDH states: given a group generator g and the values g^a and g^b for random secret exponents a and b, no efficient algorithm can distinguish g^{ab} from a random group element. That is, even knowing g^a and g^b, the shared secret g^{ab} looks random to any computationally bounded observer. For ElGamal, DDH ensures that the ciphertext component $g^m \cdot pk^r = g^m \cdot g^{sk \cdot r}$ is indistinguishable from a random group element, hiding the vote m.

3.3.2 Perceived quantum threat

A common misconception is that quantum computers would break all encryption. In fact, the famed quantum algorithm of Shor [94] provides an *exponential* speedup only for schemes whose security relies on number-theoretic problems, or more abstractly, problems with group-theoretic structure [75, Chapter 2], such as integer factorisation (RSA) and the discrete logarithm (Diffie–Hellman, ElGamal). No comparable exponential quantum speedup is known for lattice-based, code-based, or hash-based schemes. Symmetric encryption such as the Advanced Encryption Standard (AES), which secures most stored data in practice, is also unaffected. The best known quantum attack uses Grover's algorithm, which offers only a quadratic speedup, easily countered by doubling key lengths. The election schemes in this tutorial rely on ElGamal and are therefore vulnerable; we discuss post-quantum alternatives in Section 10.5.

3.4 Key generation

The key generation algorithm Gen produces keys for an encryption scheme.

For asymmetric encryption, the key generation algorithm produces a *key pair* (pk, sk) consisting of a public key and a private key. The algorithm takes as input a *security parameter* κ, typically written in unary notation as 1^κ, which determines the size or strength of the generated keys; we write $(pk, sk) \leftarrow \mathsf{Gen}(\kappa)$ to denote this. Larger values of κ produce longer keys that are harder to break but slower to use.

Key generation must incorporate randomness. Otherwise, if keys were predictable, an adversary could compute the same key and break the scheme. We write $(pk, sk) \leftarrow_R \mathsf{Gen}(\kappa)$ to emphasize that generation involves random choices. For election systems, key generation is particularly sensitive: a compromised private key sk would allow an adversary to decrypt all ballots and violate ballot secrecy.

3.5 Homomorphic encryption

A traditional problem with encrypted data, such as messages or votes, is that we need to decrypt it in order to perform operations on it, which then leaves it vulnerable to adversaries like Eve. But this issue is circumvented by the concept of *homomorphic encryption*, which allows certain mathematical operations, such as addition, to be performed on the encrypted data, without ever decrypting it or even using the private key.

More formally, for all possible messages m and m' and public keys pk allowed in the encryption scheme, homomorphic under a binary operation $\oplus$, such as addition, means that

$$\mathsf{Enc}(pk, m) \odot \mathsf{Enc}(pk, m') = \mathsf{Enc}(pk, m \oplus m'),$$

where $\oplus$ and $\odot$ are two binary operations, the first being operation for the plaintexts m and m', while the second is the corresponding operation for the ciphertexts, which is not necessarily the same type of operation. Decrypting the ciphertext with the private key sk yields

$$\mathsf{Dec}(sk, \mathsf{Enc}(pk, m) \odot \mathsf{Enc}(pk, m')) = m \oplus m'.$$

Several homomorphic encryption schemes exist. Paillier [83] introduced a scheme homomorphic under addition (where ciphertext multiplication corresponds to plaintext addition). ElGamal is homomorphic under multiplication but can be adapted for addition. These are *partially* homomorphic, because they support only one operation. Gentry [50] showed that *fully* homomorphic encryption, which allows arbitrary computations on encrypted data, is possible, though this remains too computationally expensive for practical voting systems. The schemes used in electronic voting, including those we study, rely on partially homomorphic encryption.

For a voting scheme, homomorphic encryption is extremely useful, as it is possible to count or tally the votes of the voters without ever decrypting the individual votes or even accessing the private key. For example, in a two-candidate election, voters encrypt either 0 or 1. These encrypted votes are homomorphically combined to derive a ciphertext encoding the total votes for candidate 1, which is then decrypted to reveal the outcome, without ever revealing individual votes.

As discussed by Paillier [83], Park and Rivest [85] and others, homomorphic encryption is important for preserving secrecy in the voting system.

3.6 Malleable and non-malleable encryption

An encryption scheme is *malleable* if, given a ciphertext, one can produce a valid ciphertext for a related plaintext, possibly without learning the original plaintext. A *non-malleable* scheme prevents

this: given a ciphertext, it is infeasible to create another valid ciphertext for a related message.

Homomorphic encryption is inherently malleable (combining ciphertexts produces a valid ciphertext for the combined plaintexts), while non-malleable encryption cannot be homomorphic. For voting, this creates a tension: homomorphic encryption enables efficient tallying, but malleability can enable attacks. We return to this connection when discussing ballot privacy.

3.7 Threshold encryption

In *threshold encryption*, decryption requires collaboration among multiple parties. With n total *shares* (partial decryption keys), at least t shares must cooperate to decrypt, since fewer than t shares reveal nothing. A *perfect threshold* scheme ensures that $t - 1$ shares provide no information about the missing shares.

For voting, threshold encryption is essential because it distributes trust: no single authority can decrypt ballots alone. Consider an election where the private key is split among five trustees with threshold $t = 3$. Even if two trustees are corrupt or compromised, they cannot decrypt any ballot without cooperation from at least one honest trustee. This prevents a single point of failure that could compromise ballot secrecy.

Threshold encryption also enables *distributed key generation*, where trustees jointly generate the key pair without any single party ever knowing the complete private key. The public key is published for voters to encrypt their ballots, but decryption requires the trustees to collaborate, typically after the election closes. Katz and Lindell [63, Section 13.3.3] cover threshold encryption in relation to a simple election scheme.

3.8 Algorithms and primitives

In cryptography, algorithms such as Enc, Dec, and Gen are called *primitives* when treated as building blocks of larger schemes. We use the terms algorithm and primitive interchangeably.

There is some specific language used for describing the running of primitives or algorithms. To start an algorithm is to *invoke* it, and the running of an algorithm is an *invocation*. To provide input parameters to the algorithm is to *instantiate* it, so an *instantiation* of an algorithm is its set of inputs for one invocation.

3.9 Complexity approach to cryptography

Modern cryptography is based on *computational* security rather than information-theoretic security: adversaries could in principle break the system, but doing so would require non-polynomial time. Schemes are considered secure if the probability of an efficient adversary breaking them is negligible. In practical terms, security means the scheme will not be broken for hundreds or thousands of years with foreseeable technology.

3.10 Proofs

A *proof* is an object or interaction that convinces the reader of some mathematical claim. When proofs are interactive, we introduce two more characters from cryptography: Peggy the prover and Victor the verifier. Peggy needs to convince Victor in a series of interactions. At the end Victor outputs a response, either accepting or rejecting the claim by Peggy, based on whether Peggy's actions have managed to convince him of the validity of a certain mathematical statement.

Two fundamental properties characterize the quality of proof systems. *Completeness* requires that if a statement is true, an honest prover can always convince the verifier to accept. *Soundness* requires that if a statement is false, no cheating prover can convince the verifier to accept (except with negligible probability). In other words, completeness ensures that valid proofs succeed, while soundness ensures that invalid proofs fail. These concepts extend beyond proof systems: in the context of election verification, soundness means that the verification algorithm rejects incorrect outcomes, while completeness means it accepts correct ones.

A *zero-knowledge proof* allows Peggy to convince Victor of a statement without revealing *why* it is true. A classic illustration: suppose you have a red ball and a green ball, identical except for colour, and a colour-blind friend. By repeatedly asking whether the friend switched the balls behind their back, you can convince them the colours differ; yet the friend never learns which ball is which. For voting, zero-knowledge proofs are essential: the system must convince everyone that votes were tallied correctly without revealing any individual vote.

3.11 Fiat-Shamir transformation

The *Fiat-Shamir transformation* converts interactive proofs into non-interactive ones.

The key insight of the Fiat-Shamir transformation is to replace the verifier's random challenge with the output of a cryptographic hash function. In an interactive protocol, the prover sends a commitment to the verifier, the verifier responds with a random challenge, and the prover computes a response based on both the commitment and challenge. The verifier then checks the response. In the non-interactive version, instead of receiving a random challenge from the verifier, the prover computes the challenge themselves by hashing the commitment (and typically other public data). Since hash functions behave unpredictably, the prover cannot choose a commitment that will produce a favorable challenge.

More formally, suppose an interactive proof has the prover send commitment a, receive challenge c, and respond with z. The Fiat-Shamir transformation produces a non-interactive proof (a, c, z) where $c = H(a, m)$ for a hash function H and message m being proved. The verifier checks both that $c = H(a, m)$ and that (a, c, z) is a valid transcript.

The security of the Fiat-Shamir transformation is typically analyzed in the *random oracle model*, which treats the hash function as a truly random function. While no real hash function is truly random, this idealization captures the intuition that a good hash function is unpredictable. Proofs in the random oracle model provide strong evidence of security, though they do not guarantee security when the random oracle is instantiated with a concrete hash function.

A subtle but important variant is the *weak Fiat-Shamir transformation*, which hashes only the commitment a rather than including the message m. This seemingly minor difference can introduce serious vulnerabilities in some contexts, as we will see when analyzing the Helios voting system. The weak variant allows an adversary more freedom in constructing proofs, potentially enabling attacks that the standard transformation would prevent.

For electronic voting, the Fiat-Shamir transformation is essential because it allows voters to construct non-interactive proofs that their ballots are well-formed without requiring real-time interaction with a verifier. These proofs can then be posted to a public bulletin board and verified by anyone at any time.

3.12 Simulator

In cryptography, an important concept related to proofs is that of a *simulator*. This is essentially a fast (probabilistic polynomial-time) algorithm that can produce outputs when called upon by

the verifier without interacting with the real prover, and the outputs are indistinguishable from those resulting from interactions with the real prover. In other words, given Victor the verifier and Peggy the prover, the simulator is able to *simulate* Victor's interaction with Peggy.

3.13 Some useful functions

We now introduce functions that appear throughout cryptographic definitions.

3.13.1 Hash functions

A *hash function* maps data of arbitrary size to a fixed-size output. The main requirement is *collision resistance*, meaning it should be computationally infeasible to find two distinct inputs that produce the same output. In the voting context, collisions could allow manipulation of election outcomes by creating duplicate ballots.

3.13.2 Negligible functions

A *negligible function* $\mathsf{negl}(\kappa)$ is one that decreases faster than the inverse of any polynomial as the security parameter κ grows. For example, $2^{-\kappa}$ and $\kappa^{-\log \kappa}$ are negligible. Events occurring with negligible probability are considered infeasible in practice, because the time needed for them to occur would be more than the lifetime of our universe. A cryptographic scheme is secure if the probability of any efficient adversary breaking it is negligible; see Katz and Lindell [63, Section 3.1.2] for numerical examples.

3.14 Importance of randomness

Why does randomness appear everywhere in cryptography? Two principal reasons. First, randomness produces unpredictable data that is secret to all parties. This is essential for key generation: if a key generation algorithm produced predictable keys, an adversary could compute the same key and break the scheme.

Second, there is always some probability that an adversary could guess a key by chance. A good cryptographic system makes this probability negligible, meaning vanishingly small as the security parameter increases. The negligible functions defined above make this notion precise: as the key length grows, the probability of a successful guess decreases faster than any polynomial, rendering brute-force attacks infeasible.

3.15 Probabilistic algorithms

A *probabilistic algorithm* (also called a *randomized algorithm*) takes a deterministic input, performs operations using random choices (such as coin flips), and produces an output influenced by that randomness. Two executions on the same input will likely produce different outputs. For voting, this is crucial: if encryption were deterministic, an adversary could encrypt each candidate's name and compare the results to voters' ballots, breaking secrecy.

3.16 Random variables

We write the output of a probabilistic algorithm as $A(x_1, \ldots, x_n; r)$, where $x_1, \ldots, x_n$ are deterministic inputs and r represents the random choices. The randomness of r typically stems from a sequence of independent random bits, called *coins* in cryptography (or *Bernoulli random variables*

in probability theory). We write $x \leftarrow_R S$ to denote sampling an element x uniformly at random from set S. A *nonce* is a number used only once, often generated randomly for authentication.

3.17 Polynomial-time algorithms

An algorithm is *polynomial-time* if the number of steps it takes is bounded by a polynomial in the input size. A *probabilistic polynomial-time (PPT)* algorithm runs in polynomial time regardless of its random choices. Security definitions in cryptography restrict adversaries to probabilistic polynomial-time algorithms, which captures efficient attackers, while excluding those requiring exponential resources.

3.18 Cryptographic tools for electronic voting

The cryptographic primitives described above, such as encryption, proofs, and hash functions, are general-purpose tools. Electronic voting systems combine these primitives in specific ways to achieve ballot secrecy and verifiability. This section introduces cryptographic constructions that are particularly important for voting applications.

3.18.1 Bulletin boards

The *bulletin board* is a central component of electronic voting systems. It serves as a public ledger where encrypted ballots are posted and stored until tallying. The bulletin board must satisfy several properties:

- *Public readability*: Anyone can read the contents of the bulletin board. This enables universal verifiability, as any observer can check that the announced outcome corresponds to the posted ballots.
- *Append-only*: Once a ballot is posted, it cannot be removed or modified. New ballots can only be added. This ensures that voters can verify their ballot remains on the board throughout the election.
- *Consistency*: All observers see the same bulletin board contents. Different parties cannot be shown different versions of the board.
- *Availability*: The bulletin board remains accessible throughout the election and verification period.

In our formal model, we represent the bulletin board as a set $\mathfrak{bb}$ of ballots. The set representation abstracts away implementation details while capturing the essential property that ballots are collected for tallying. In practice, bulletin boards may be implemented using web servers or distributed ledgers.

We typically assume an honest bulletin board in security definitions: ballots posted by voters appear on the board, and the board accurately reflects all posted ballots. Relaxing this assumption leads to stronger adversary models where the adversary controls ballot collection.

3.18.2 Encoding votes

Election schemes need to represent votes in a form suitable for cryptographic operations. Several encoding strategies exist, each with trade-offs for efficiency and the types of elections they support.

Integer encoding. The simplest approach encodes a vote for candidate $v \in \{1, \ldots, nc\}$ directly as the integer v. This works well with homomorphic encryption schemes that support addition: the sum of encrypted votes yields an encrypted sum, but this sum alone does not reveal individual vote counts per candidate. Additional techniques (such as computing the sum of g^{v_i} values and solving a discrete logarithm for small results) can extract the tally.

Bitstring encoding. A more common approach for homomorphic tallying encodes each vote as a binary vector. For an election with nc candidates, a vote for candidate v becomes a vector of length $nc - 1$:

- If $v < nc$: position v contains 1, all other positions contain 0
- If $v = nc$: all positions contain 0

For example, with $nc = 4$ candidates:

Vote for candidate 1:	$(1, 0, 0)$
Vote for candidate 2:	$(0, 1, 0)$
Vote for candidate 3:	$(0, 0, 1)$
Vote for candidate 4:	$(0, 0, 0)$

Each component is encrypted separately, producing a tuple of ciphertexts. Homomorphically combining ciphertexts column-wise across all ballots yields encrypted sums; decrypting these reveals the vote count for each candidate without decrypting individual ballots.

The Helios voting system uses bitstring encoding. This is why Helios ballots contain tuples of ciphertexts $(c_1, \ldots, c_{nc-1})$ rather than a single ciphertext.

Example 2 (Bitstring encoding and homomorphic tallying)**.** *We trace the full pipeline for a small election with $nc = 3$ candidates and three voters.*

Step 1: Encode. *Each voter encodes their vote as a binary vector of length $nc - 1 = 2$:*

	Vote	*Encoding*
Voter 1:	*candidate 2*	$(0, 1)$
Voter 2:	*candidate 1*	$(1, 0)$
Voter 3:	*candidate 1*	$(1, 0)$

Step 2: Encrypt. *Each voter encrypts the components of their encoding separately and posts the resulting ballot to the bulletin board. Writing $E(x)$ for an encryption of x, the bulletin board contains:*

	Column 1	*Column 2*
Voter 1:	$E(0)$	$E(1)$
Voter 2:	$E(1)$	$E(0)$
Voter 3:	$E(1)$	$E(0)$

Note that each $E(\cdot)$ uses fresh randomness, so no two ciphertexts look alike, even those encrypting the same value.

Step 3: Combine. *The tallier combines ciphertexts* column-wise *using the homomorphic property, without decrypting any individual ballot:*

$$\underbrace{E(0) \otimes E(1) \otimes E(1)}_{\textit{Column 1}} = E(0+1+1) = E(2),$$
$$\underbrace{E(1) \otimes E(0) \otimes E(0)}_{\textit{Column 2}} = E(1+0+0) = E(1).$$

Step 4: Decrypt and read off the result. *The tallier decrypts each column sum exactly once:*

$$\textit{Column 1} \to 2, \qquad \textit{Column 2} \to 1.$$

These are the vote counts for candidates 1 *and* 2*, respectively. The count for candidate* 3 *is obtained by subtraction:* $3 - 2 - 1 = 0$*, where* 3 *is the total number of ballots.*

The election outcome is the vector $(2, 1, 0)$*: candidate* 1 *received two votes, candidate* 2 *received one vote, and candidate* 3 *received none. At no point was any individual ballot decrypted; only the column-wise sums were. This is the mechanism used by Helios.*

3.18.3 Mix networks

A *mix network* (or *mixnet*) is a cryptographic protocol for anonymous communication, introduced by Chaum [30]. Mix networks break the link between voters and their votes while still allowing votes to be tallied.

Basic concept. A set of encrypted messages enters the network, and the same messages exit, re-encrypted and reordered, so that an observer who sees both inputs and outputs cannot determine which input corresponds to which output, provided at least one mix server behaves honestly.

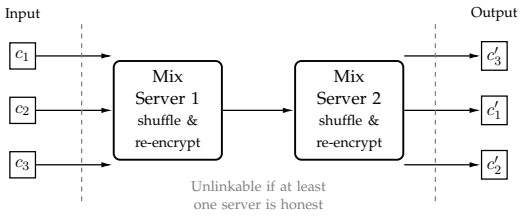
Consider an analogy. Several people each place a sealed envelope containing a message into a box. A trusted party shakes the box thoroughly, opens all envelopes, and reads out the messages in random order. An observer learns all messages but cannot determine who wrote which one. A mix network achieves this electronically, replacing physical shuffling with cryptographic operations.

Structure. A mix network consists of a sequence of *mix servers* $M_1, M_2, \ldots, M_k$. Each server receives a batch of ciphertexts, randomly reorders (shuffles) them, and *re-encrypts* each one with fresh randomness. Re-encryption produces a new ciphertext that looks entirely different but still encrypts the same plaintext; this is possible with encryption schemes like ElGamal that support homomorphic re-encryption. Even if an observer knows some servers' permutations, as long as one server's permutation remains secret, the overall mapping from inputs to outputs is hidden.

Figure 2 illustrates how ciphertexts pass through a sequence of mix servers.

Verifiable shuffles. For voting, mix servers must prove they shuffled correctly without revealing the permutation they used. Each server therefore produces a zero-knowledge proof showing that its output ciphertexts are exactly a re-encryption and permutation of its inputs: nothing added, removed, duplicated, or modified. These proofs allow anyone to verify mixing integrity while learning nothing about which input maps to which output.

Figure 2 Operation of a mix network. Ciphertexts enter and pass through a sequence of mix servers. Each server randomly permutes its inputs and re-encrypts them, producing ciphertexts that encrypt the same plaintexts but appear different and are in an unpredictable order. An observer cannot link inputs to outputs provided at least one server keeps its permutation secret.



Application to voting. In a mixnet-based voting system, voters encrypt their votes and post ciphertexts to the bulletin board. The mix servers sequentially shuffle and re-encrypt the ciphertexts, each providing a proof of correct shuffling. After the final shuffle, the ciphertexts are decrypted (using threshold decryption) to reveal votes, which are then tallied. Mixing breaks the link between voters and encrypted votes, so decryption reveals only shuffled votes rather than who cast each one. The shuffle proofs ensure universal verifiability. The Helios Mixnet system (Section 11.2) uses this approach.

3.18.4 Homomorphic tallying vs. mixnet tallying

Electronic voting systems use two main approaches to compute election outcomes while preserving ballot secrecy.

Homomorphic tallying. With additively homomorphic encryption (Section 3.5), encrypted votes can be combined without decryption:

$$\mathsf{Enc}(v_1) \otimes \mathsf{Enc}(v_2) \otimes \cdots \otimes \mathsf{Enc}(v_n) = \mathsf{Enc}(v_1 + v_2 + \cdots + v_n)$$

Only the final sum is decrypted, revealing the aggregate tally but never individual votes. This is efficient and provides strong privacy, but it is limited to elections where the outcome is a sum of votes. It cannot support complex voting methods like ranked-choice or approval voting. The basic Helios system uses homomorphic tallying.

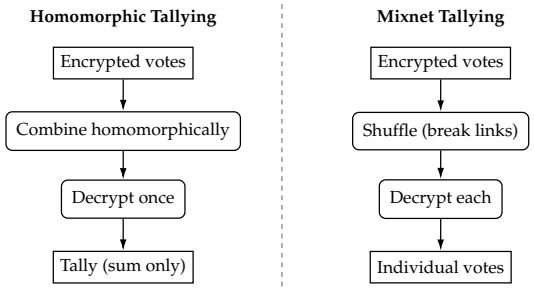
Mixnet tallying. With mix networks, the encrypted votes are first shuffled through the mixnet, then each shuffled ciphertext is decrypted to reveal an individual vote. The revealed votes can then be tallied using any counting method. This supports arbitrary voting methods and provides flexibility for complex elections, but it reveals individual votes (though not who cast them) and requires verification of the shuffle proofs. Helios Mixnet uses this approach to support elections beyond simple plurality voting.

Figure 3 contrasts the two approaches.

3.18.5 Well-definedness

An encryption scheme satisfies *well-definedness* if there is an efficient way to distinguish properly formed ciphertexts from malformed ones. A ciphertext is *well-formed* if it could have been produced by the encryption algorithm on some valid plaintext; otherwise it is *ill-formed*.

Figure 3 Two approaches to tallying encrypted votes. *Left:* Homomorphic tallying combines ciphertexts mathematically, then decrypts once to reveal only the aggregate sum, so the individual votes are never exposed. *Right:* Mixnet tallying shuffles ciphertexts to break the link between voters and ballots, then decrypts each vote individually. Homomorphic tallying reveals less information but only supports summation; mixnet tallying supports arbitrary voting methods but reveals individual votes (though not who cast them).



For many encryption schemes, any bit string of appropriate length could potentially be a ciphertext, making tampering difficult to detect. A well-defined scheme provides a method to check whether a ciphertext lies in the valid ciphertext space.

Well-definedness is important for voting because:

- It allows detection of malformed ballots before tallying.
- It ensures tallying algorithms behave predictably on all inputs.
- It prevents attacks where adversaries submit crafted invalid ciphertexts that cause unexpected behavior during decryption.

Well-definedness is distinct from non-malleability. Non-malleability prevents producing a *related valid* ciphertext from an existing one. Well-definedness allows detecting *invalid* ciphertexts regardless of how they were produced. Voting systems typically need both properties.

3.18.6 Ballot weeding

Even with non-malleable encryption, an adversary might submit a ballot that is *related* to another voter's ballot in a way that leaks information during tallying. For example, if the adversary can copy another voter's encrypted ballot and submit it as their own, the tally would reveal whether that voter's choice matched a particular candidate (by showing two votes instead of one).

Ballot weeding is a countermeasure that removes or rejects ballots deemed to be meaningfully related to other ballots before tallying. The precise definition of *meaningfully related* depends on the voting system, but typically includes duplicate ballots and ballots that can be shown to encrypt related plaintexts. Ballot weeding complements non-malleability: non-malleability makes it hard to create related ballots, while ballot weeding provides a second line of defense by detecting and removing any that slip through.

3.19 Further reading on cryptography

For introductions to the modern theory of cryptography, see, for example, the texts by Goldreich [52] or Katz and Lindell [63]. The classic text by Menezes, Van Oorschot and Vanstone [76] gives a wide-encompassing treatment of applied cryptography. More advanced readers can also

see the recent edited collection of cryptography tutorials [75]. Gentry [51] covers the basics of fully homomorphic encryption in an approachable fashion.

4 Game-based cryptography

A modern approach to cryptography is to detect *breaks* (that is, develop suitable attacks that find holes) in cryptography systems by using the concept of a *game* or an *experiment*. In game-based cryptography, a game formulates a series of interactions between a benign challenger, a malicious adversary $\mathcal{A}$, and a cryptography scheme. The adversary $\mathcal{A}$ wins the game by completing a *task* that captures an execution of the scheme in which security is broken. In other words, the adversary $\mathcal{A}$ wins by doing what should be unachievable. Formally, games in cryptography are interpreted as probabilistic (or randomized) algorithms that output Booleans, that is, answers such as true or false; $\top$ or $\bot$; 0 or 1; and so on. Importantly, adversaries are said to be *stateful*, meaning information persists across invocations of an adversary in a game, so adversaries can access earlier assignments.

In the electronic voting literature, it is standard to analyze voting systems by using methods from game-based cryptography. In the current setting, we consider a benign challenger, a malicious adversary and a voting system. The game is that the adversary seeks to break the security of the voting system by, for example, changing votes or revealing the votes of voters. For example, Smyth, Frink and Clarkson [103] used this approach to study their definitions of universal and individual verifiability.

We will now illustrate the basics of game-based cryptography. Notation used throughout this section and the rest of the tutorial is collected in Section 5.

4.1 Indistinguishability as a game

For a concrete example of a game, we introduce the concept of *indistinguishability*, which is when an adversary observes a single ciphertext but then is incapable of determining which of two message the ciphertext corresponds to. In general, indistinguishability is important in cryptography. For voting systems, ballot secrecy can be expressed as the inability to distinguish between an instance of a voting system in which voters cast some votes, from another instance in which the voters cast a permutation of those votes.

An encryption scheme is said to be *perfectly indistinguishable* if no adversary $\mathcal{A}$ can succeed with probability better than one half, meaning, no attacker can do any better than guessing the correct message half the time. But this concept is an idealization, as the probability will not equal exactly one half in a finite system. The probability should approach or converge exceedingly fast to one half as some security parameter κ, such as the key length, approaches infinity.

Consequently we will define a game that formalize the notion of an encryption scheme being *indistinguishable*. To consider this game formally, we write $\Pi = (\mathsf{Enc}, \mathsf{Dec}, \mathsf{Gen})$ to denote an encryption scheme with the message space $\mathcal{M}$, where we recall the respective primitives Enc, Dec and Gen for encryption, decryption and key generation. We now define a game called *indistinguishability*, which we denote by Ind.

Game $\mathsf{Ind}(\Pi, \mathcal{A}, \kappa) =$

1 The adversary $\mathcal{A}$ outputs two messages m_0 and m_1, which both belong to the space of all possible messages $\mathcal{M}$, meaning $m_0, m_1 \in \mathcal{M}$;
2 A key pair pk and sk is generated using the key generation primitive $\mathsf{Gen}(\kappa)$;
3 A random uniform bit is also generated β, meaning a random variable $\beta \in \{0, 1\}$, which is used to randomly choose a message m_β;
4 Applying the encryption primitive Enc to the (randomly chosen) message m_β, a ciphertext c is computed, written as $c \leftarrow \mathsf{Enc}(pk, m_\beta)$;
5 Ciphertext c is given to the adversary $\mathcal{A}$;
6 The adversary $\mathcal{A}$ outputs a bit β';
7 The output of the game is defined to be $\top$ (true) if the bit $\beta' = \beta$, and $\bot$ (false) otherwise;

We say that the adversary $\mathcal{A}$ succeeds if the outcome of the above game is $\top$, which we write as $\top \leftarrow \mathsf{Ind}(\cdot)$. We write $\mathsf{Succ}(\mathsf{Ind}(\cdot)) = \Pr[\top \leftarrow \mathsf{Ind}(\cdot)]$ to denote the probability that the adversary succeeds in the game Ind. We can now formally define the concept of an encryption scheme being indistinguishable.

Definition 1 (Indistinguishable)**.** *Given the game* Ind*, an encryption scheme* $\Pi = (\mathsf{Enc}, \mathsf{Dec}, \mathsf{Gen})$ *with the message space* $\mathcal{M}$*, and a security parameter* κ*, is indistinguishable if for every adversary* $\mathcal{A}$ *the following inequality*

$$\mathsf{Succ}(\mathsf{Ind}(\cdot)) \leq \frac{1}{2} + \mathsf{negl}(\kappa)\,.$$

holds, where $\mathsf{negl}(\kappa)$ *is a negligible function in* κ*.*

The probability of the adversary succeeding is at most one half plus a negligible amount. We see that as the (finite) security parameter κ approaches infinity, the probability of the adversary succeeding approaches one half (overwhelmingly fast). This can be expressed as $\mathsf{Succ}(\mathsf{Ind}(\cdot)) \to 1/2$ as $\kappa \to \infty$, which in the limit gives a perfectly indistinguishable encryption scheme.

How to read a security game

A security game is *not* a procedure to execute; it is a *probability experiment* used to define what security means. When reading a game:

1. **Identify the secret.** There is usually a random bit β (or similar hidden value) that the adversary must guess.
2. **Identify the adversary's view.** What information does the adversary receive? This is typically limited to public keys, ciphertexts, and oracle responses, and never the private key or the secret bit directly.
3. **Read the Return line as the winning condition.** The game outputs $\top$ (true) when the adversary wins. The conditions in the Return line define *what counts as a win*; they are not steps the adversary performs.
4. **Think probabilistically.** Every $\leftarrow_R$ introduces randomness. The question is: over all these random choices, how often does the adversary win? Security means the adversary cannot win much more often than by random guessing.

Each game defines a specific threat model. Different games for the same scheme capture different attack scenarios (for example, passive eavesdropping vs. active manipulation).

4.2 Hierarchy of security notions

The indistinguishability game from Section 4.1 captures a basic notion of security. Cryptographers have defined several variants, forming a hierarchy from weaker to stronger guarantees. Understanding this hierarchy helps in selecting appropriate security assumptions for proofs. To orient the reader, the key notions and their logical relationships are

$$\text{IND-CCA} \rightarrow \text{NM-CPA} \leftrightarrow \text{IND-PA0} \rightarrow \text{IND-CPA}$$

where arrows indicate implication. The weakest notion, IND-CPA, captures security against passive eavesdroppers. The strongest, IND-CCA, implies non-malleability (NM-CPA). For election schemes, the intermediate notion IND-PA0 turns out to be the most natural, because its parallel decryption oracle models the tallying phase where all submitted ballots are decrypted together. We now define each notion.

IND-CPA: Indistinguishability under Chosen Plaintext Attack. This is the basic notion of semantic security that we introduced above. An adversary who can obtain encryptions of chosen messages still cannot distinguish encryptions of two messages of their choice. This captures security against passive eavesdroppers who observe ciphertexts but cannot interfere with the system.

IND-CCA: Indistinguishability under Chosen Ciphertext Attack. A stronger notion where the adversary additionally has access to a decryption oracle and can obtain decryptions of chosen ciphertexts (except the challenge ciphertext). This models active attackers who might trick honest parties into decrypting messages. IND-CCA security implies non-malleability.

IND-PA0: Indistinguishability under Parallel Attack. An intermediate notion, introduced by Bellare and Sahai [9], where the adversary receives a challenge ciphertext and can then submit a vector of ciphertexts for decryption (all at once, not adaptively), excluding the challenge itself. This notion is particularly relevant for voting because tallying is fundamentally a decryption step: the election authority decrypts all submitted ballots together to produce the outcome. The parallel decryption oracle in IND-PA0 models exactly this situation, where an adversary constructs ballots that will all be decrypted in a single batch during tallying. The formal definition appears in Section D of the appendix.

NM-CPA: Non-Malleability under Chosen Plaintext Attack. Rather than an indistinguishability requirement, this captures a qualitatively different property: given a ciphertext, the adversary cannot produce a ciphertext of a *meaningfully related* plaintext. For example, given an encryption of m, the adversary should not be able to produce an encryption of $m + 1$ without knowing m.

Relationships. The equivalence NM-CPA $\leftrightarrow$ IND-PA0 is a theorem of Bellare and Sahai [9], showing that non-malleability and this form of indistinguishability are the same notion expressed differently. This equivalence is useful because IND-PA0 is often easier to work with in proofs than the more complex definition of non-malleability.

For election schemes, we typically require IND-PA0 (equivalently, NM-CPA) because ballot secrecy requires indistinguishability, verifiability requires non-malleability, and the parallel attack model matches the voting scenario where multiple ballots are decrypted together during tallying.

4.3 Game formulation

Using our notation, we can formulate a game, denoted by $\mathsf{Exp}(H, \mathcal{S}, \mathcal{A})$, which gives an adversary $\mathcal{A}$ the task of distinguishing between a function H and a simulator $\mathcal{S}$. (The idea of a simulator is important in game-based cryptography, as it gives a means for the adversary to generate false information with the aim of winning the game.)

The game $\mathsf{Exp}(H, \mathcal{S}, \mathcal{A})$ is as follows.

Game $\mathsf{Exp}(H, \mathcal{S}, \mathcal{A}) =$

1 $m \leftarrow \mathcal{A}(); \beta \leftarrow_R \{0, 1\}$;
2 **if** $\beta = 0$ **then** $x \leftarrow H(m)$; **else** $x \leftarrow S(m)$; ;
3 $g \leftarrow \mathcal{A}(x)$;
4 **return** $(g = \beta)$;

In words, the adversary $\mathcal{A}$ first creates a message m and then a random bit β is generated. If this random bit is equal zero (so $\beta = 0$), a function H is applied to the message; otherwise, the simulator S is applied. The adversary receives the output x and must guess the value of β. The adversary wins if their guess g matches β.

An adversary *wins* a game by causing it to output $\top$, and *loses* the game if the output is $\bot$. Since adversaries are stateful, information persists across invocations of an adversary in a game, meaning adversaries can access earlier assignments.

We say that the adversary's *success* in a game $\mathsf{Exp}(\cdot)$, denoted $\mathsf{Succ}(\mathsf{Exp}(\cdot))$, is the probability that the adversary wins. That is, adversary's success is

$$\mathsf{Succ}(\mathsf{Exp}(\cdot)) = \Pr[x \leftarrow \mathsf{Exp}(\cdot) : x = \top] = \Pr[\top \leftarrow \mathsf{Exp}(\cdot)] \,.$$

Interpreting the game $\mathsf{Exp}(\cdot)$ as a random variable, which takes the value $\top$ or $\bot$, we can use standard probability notation $\mathsf{Succ}(\mathsf{Exp}(\cdot)) = \Pr[\mathsf{Exp}(\cdot) = \top]$. We have used the probability to formulate game success, because we focus on computational security, rather than information-theoretic security. This means we tolerate breaks by adversaries in non-polynomial time and breaks with negligible success, since such breaks are infeasible in practice.

4.4 Oracles

An important notion in cryptography is that of an oracle, which formalizes the idea of accessing certain information to help break a cryptographic scheme. The oracle is treated as a blackbox that can be given certain questions or task to aid the adversary, which can be answered or done quickly. (The notion of an oracle can be made formal by modifying a Turing machine, but such details are not important here.)

To give an example of an oracle at work, an adversary $\mathcal{A}$ in a game could use an oracle to encrypt messages by using a key that is unknown to the adversary. Using specifically chosen messages, the adversary $\mathcal{A}$ could infer the key by examining the ciphertexts produced by the oracle. The adversary $\mathcal{A}$ can interact with the oracle as many times as needed and retain information from previous interactions. For another example, an oracle may access game parameters such as the bit β used in the game Exp.

4.5 Games with many interactions

The aforementioned game Exp captures a single interaction between the challenger and the adversary. But if we couple games with oracles, we can extend the games so that they capture arbitrarily many interactions. For instance, we can formulate a strengthening of Exp as follows.

Game $\mathsf{Exp}^{\mathcal{O}}(H, \mathcal{S}, \mathcal{A}) =$

1 $\beta \leftarrow_R \{0,1\}$;
2 $g \leftarrow \mathcal{A}^{\mathcal{O}}()$;
3 **return** $(g = \beta)$;

where $\mathcal{A}^{\mathcal{O}}$ denotes $\mathcal{A}$'s access to oracle $\mathcal{O}$, and the oracle $\mathcal{O}$ is defined as follows:

- $\mathcal{O}(m)$ computes **if** $\beta = 0$ **then** $x \leftarrow H(m)$; **else** $x \leftarrow S(m)$; and outputs x.

In words, the game $\mathsf{Exp}^{\mathcal{O}}$ first generates a random bit β, then allows the adversary to interact with the oracle $\mathcal{O}$ as many times as desired. Each time the adversary submits a message m to the oracle, it receives either $H(m)$ or $S(m)$ depending on the hidden bit β. After gathering information from these interactions, the adversary outputs a guess g, and wins if this guess matches β.

4.6 Anatomy of a security game

Security definitions in cryptography are typically expressed as games between a *challenger* (who runs the cryptographic scheme honestly) and an *adversary* (who tries to break the scheme). Understanding how to read these game definitions is essential for the technical sections that follow.

A security game has several standard components:

Setup phase. The challenger initializes the cryptographic scheme, typically by generating keys. For example, $(pk, sk, mb, mc) \leftarrow \mathsf{Setup}(\kappa)$ generates a key pair and system parameters. The adversary usually receives the public key but not the private key.

Adversary queries. The adversary interacts with the challenger, often through an *oracle* that provides specific capabilities. For instance, an encryption oracle allows the adversary to obtain encryptions of chosen messages. We write $\mathcal{A}^{\mathcal{O}}$ to denote adversary $\mathcal{A}$ with access to oracle $\mathcal{O}$.

Challenge phase. The adversary attempts to accomplish a task that should be infeasible if the scheme is secure. This often involves distinguishing between two possibilities (such as which of two messages was encrypted) or forging something (such as a valid signature).

Output and winning condition. The game returns a Boolean value: $\top$ (true) if the adversary wins, $\bot$ (false) otherwise. The winning condition is specified as a predicate that must hold for the adversary to win.

Example: Reading the IND game. Consider the indistinguishability game from Section 4.1. Let us walk through it line by line:

1. *"The adversary $\mathcal{A}$ outputs two messages m_0 and m_1"*: The adversary chooses which two messages to be challenged on. This models a chosen-plaintext attack.
2. *"A key pair pk and sk is generated"*: The challenger runs key generation honestly.
3. *"A random uniform bit β is generated"*: This is the secret the adversary must guess. The adversary wins by determining β.
4. *"Ciphertext c is computed as $c \leftarrow \mathsf{Enc}(pk, m_\beta)$"*: The challenger encrypts one of the two messages, chosen by the random bit.

5. *"Ciphertext c is given to the adversary"*: The adversary sees only the ciphertext, not β.
6. *"The adversary outputs a bit β'"*: The adversary's guess for which message was encrypted.
7. *"Output $\top$ if $\beta' = \beta$"*: The adversary wins if they guessed correctly.

The scheme is secure if no efficient adversary can win with probability significantly better than $1/2$ (random guessing).

Bounded winning probability. Security definitions typically require that the adversary's success probability is bounded:

$$\mathsf{Succ}(\mathsf{Game}(\cdot)) \leq \frac{1}{2} + \mathsf{negl}(\kappa)$$

The term $\frac{1}{2}$ represents random guessing (for bit-guessing games). The term $\mathsf{negl}(\kappa)$ is a negligible function, meaning the adversary's advantage over guessing vanishes as the security parameter grows. For games where random success is not $1/2$ (such as games where the adversary wins only by achieving something specific), the bound might instead be $\mathsf{Succ}(\mathsf{Game}(\cdot)) \leq \mathsf{negl}(\kappa)$.

4.7 Security proofs by reduction

The standard technique for proving that a cryptographic construction is secure is *proof by reduction*. The idea is to show that if an adversary $\mathcal{A}$ can break the construction, then we can use $\mathcal{A}$ as a subroutine to build another adversary $\mathcal{B}$ that breaks some underlying assumption believed to be hard.

Structure of a reduction proof. A reduction proof typically proceeds as follows:

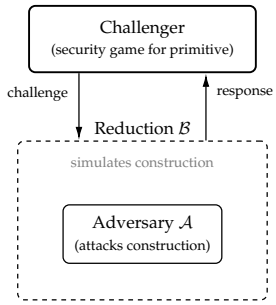
1. *Assume the contrary*: Suppose the construction is insecure, meaning there exists an efficient adversary $\mathcal{A}$ that wins the security game with non-negligible advantage.
2. *Construct a new adversary*: Build adversary $\mathcal{B}$ that uses $\mathcal{A}$ as a black box. Adversary $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$ while itself being a challenger in a different game.
3. *Simulation*: Show that $\mathcal{B}$ can simulate the environment that $\mathcal{A}$ expects. When $\mathcal{A}$ makes oracle queries, $\mathcal{B}$ must respond in a way that is indistinguishable from the real game.
4. *Extraction*: When $\mathcal{A}$ succeeds in its task, show how $\mathcal{B}$ can extract a solution to the underlying hard problem.
5. *Conclude*: Since breaking the underlying assumption is believed to be hard, and $\mathcal{B}$'s success probability is related to $\mathcal{A}$'s, we conclude that $\mathcal{A}$ cannot have non-negligible advantage.

Example sketch. To prove that election scheme $\mathsf{Enc2Vote}(\Pi)$ satisfies ballot secrecy (Proposition 3), we show: if adversary $\mathcal{A}$ breaks ballot secrecy, then we can construct adversary $\mathcal{B}$ that breaks IND-PA0 of the encryption scheme Π.

Adversary $\mathcal{B}$ works as follows:

- $\mathcal{B}$ receives a public key pk from its own challenger and forwards it to $\mathcal{A}$.
- When $\mathcal{A}$ outputs votes (v_0, v_1), adversary $\mathcal{B}$ forwards these to its challenger as the two messages.

Figure 4 Structure of a security reduction. To prove a construction secure, we assume an adversary $\mathcal{A}$ can break it and build a reduction $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine. The reduction $\mathcal{B}$ plays the security game for the underlying primitive while simulating the construction for $\mathcal{A}$. If $\mathcal{A}$ succeeds, $\mathcal{B}$ uses this to win its own game. Since the primitive is assumed secure, no efficient $\mathcal{A}$ can exist.



- $\mathcal{B}$ receives a challenge ciphertext c (an encryption of either v_0 or v_1) and gives it to $\mathcal{A}$ as the ballot.
- $\mathcal{B}$ simulates the rest of the ballot secrecy game for $\mathcal{A}$.
- When $\mathcal{A}$ outputs its guess g, adversary $\mathcal{B}$ outputs g as its own guess.

If $\mathcal{A}$ can distinguish which vote was encrypted, then $\mathcal{B}$ can distinguish which message was encrypted, contradicting the security of Π. The key insight is that $\mathcal{B}$ perfectly simulates the ballot secrecy game for $\mathcal{A}$, so $\mathcal{A}$'s success probability transfers to $\mathcal{B}$.

Why reductions matter. Reduction proofs provide *provable security*: the security of a complex construction (like an election scheme) is reduced to the security of simpler, well-studied primitives (like encryption schemes). Rather than trying to prove absolute security (which is generally impossible), we show that breaking the construction is *at least as hard as* breaking the underlying primitive.

Figure 4 illustrates the structure of a reduction proof.

4.8 Further reading on game-based cryptography

Bellare and Rogaway have made foundational contributions to game-based security definitions. Their work on relations between cryptographic definitions [7] clarifies how different security notions (such as IND-CPA, IND-CCA, and non-malleability) relate to one another. Their treatment of the random oracle model [8] provides an important idealization used in many security proofs.

For readers interested in the application of game-based techniques to voting systems specifically, Smyth, Frink, and Clarkson [103] provide game-based definitions of verifiability, while Smyth [96] develops game-based definitions of ballot secrecy that we use in later sections.

The edited volume by Lindell [75] contains accessible tutorials on various aspects of modern cryptography, including both game-based and simulation-based approaches. This collection is particularly useful for understanding how these two paradigms complement each other.

4.9 Games for voting security

In subsequent sections, we use different styles of security games to capture different properties of election schemes. *Verifiability* (Section 9) uses games that task an adversary to reach a bad

state—producing a fraudulent tally that passes verification, or causing ballots to collide. These are *reachability* games: the scheme is secure if no efficient adversary can reach the forbidden state. *Ballot secrecy* (Section 10) uses *indistinguishability* games, where the adversary must distinguish between two scenarios (such as which of two votes was encrypted). The scheme is secure if the adversary cannot do better than guessing. Recognizing which style of game applies to which property helps when reading the formal definitions that follow.

5 Notation

Having introduced cryptographic primitives and security games, we now consolidate the notation used throughout this tutorial. This section serves as a reference for describing cryptographic games, election schemes, and security proofs; readers familiar with this material may skip ahead and return here as needed.

5.1 Algorithms and randomness

Formally, games and cryptographic constructions are probabilistic (or randomized) algorithms that may output Booleans. We write $A(x_1, \ldots, x_n; r)$ to denote the output of a probabilistic algorithm A, where $x_1, \ldots, x_n$ are the inputs and r is a collection of independent coins that are each chosen uniformly at random from a suitable space of coin values. We often adopt the shorthand $A(x_1, \ldots, x_n)$ to denote $A(x_1, \ldots, x_n; r)$, leaving the randomness implicit.

5.2 Assignments

To denote an assignment of T to x, we write $x \leftarrow T$, meaning x takes on the value of T. For a random assignment, we write $x \leftarrow_R S$ to express that x is chosen uniformly at random from a finite set S. We write $x, x' \leftarrow_R S$ for the independent random assignments $x \leftarrow_R S$ and $x' \leftarrow_R S$.

We write $\Pr(B)$ to denote the probability of a random event B happening.

5.3 Vectors and indexing

Let $x[i]$ denote the i-th component of vector x, while $|x|$ denotes the length of vector x. To express an assignment with vectors, we write $(x_1, \ldots, x_{|T|}) \leftarrow T$ for $x \leftarrow T; x_1 \leftarrow x[1]; \ldots; x_{|T|} \leftarrow x[|T|]$, when T is a vector.

5.4 Logic notation

When defining games and algorithms, we use standard logic notation to write predicates (or Boolean-valued functions), which return true or false. A predicate is a mathematical statement that tests whether something is true or false. You can think of it as a yes-or-no question applied to particular values. For example, the predicate $(n \leq m)$ returns true if the inequality is satisfied for n and m.

We combine predicates with the standard logical operators: *conjunction* $p \wedge q$ (true when both hold), *disjunction* $p \vee q$ (true when at least one holds), *negation* $\neg p$ (true when p is false), and *implication* $p \Rightarrow q$ (if p then q). For example, we might write $(a \neq b) \wedge (n \leq m)$ or $(a \neq b) \vee (n \leq m)$ to combine conditions.

Predicates are also used with quantifiers to express counting conditions. For instance, the notation $(\exists^{=\ell} x : P(x))$ means that exactly ℓ elements satisfy the property P. In other words, there

are precisely ℓ values of x (no more, no fewer) for which the test $P(x)$ returns true. This counting notation appears throughout our formal definitions of election properties.

5.5 Quantifiers

The *universal quantifier* $\forall$ ("for all") and the *existential quantifier* $\exists$ ("there exists") have their standard meanings: $\forall x \in S : P(x)$ asserts that property P holds for every element x in set S, while $\exists x \in S : P(x)$ asserts that P holds for at least one such element.

We also use a *counting quantifier* $\exists^{=\ell} x \in S : P(x)$ to express that *exactly* ℓ distinct elements x in S satisfy property $P(x)$. (More specifically, the notation $\exists^{=\ell}$ denotes a generalized quantifier; for more details on first-order logic, see, for example, the introduction by Schweikardt [93]. Variable x is bounded by the predicate, while the integer ℓ is free.) This quantifier appears in our definition of correct election outcomes; for example, $\exists^{=\ell} b \in \mathbb{bb} \setminus \{\perp\} : \exists r : b = \mathsf{Vote}(pk, v, nc, \kappa; r)$ expresses that exactly ℓ ballots on the bulletin board are valid encryptions of vote v.

5.6 Set notation

We use standard set-theoretic notation: $x \in S$ denotes membership; $S \subseteq T$ denotes that S is a subset of T; $S \cup T$, $S \cap T$, and $S \setminus T$ denote union, intersection, and set difference respectively; $|S|$ denotes cardinality; and $\emptyset$ denotes the empty set. Set-builder notation $\{x \in S \mid P(x)\}$ denotes the set of all elements x in S satisfying property P.

5.7 Tuples and parsing

A *tuple* is an ordered collection of elements, written $(x_1, x_2, \ldots, x_n)$. Unlike sets, tuples preserve order and allow duplicates. We say we *parse* an object as a tuple when we interpret it as having component parts—for example, parsing pk' as a pair $(pk, \mathcal{M})$ means interpreting pk' as consisting of a public key pk and a message space $\mathcal{M}$. Parsing may fail if the object does not have the expected structure, in which case algorithms typically output the error symbol $\perp$.

With these notational conventions established, we now turn to the formal description of voting systems and their security properties.

6 Overview of an election scheme

Now that we have the cryptographic building blocks in hand, let us revisit the election scheme introduced in Section 2 in more detail. We will soon introduce suitable properties for any election scheme at a more technical level. For now, we expand on one example of an election scheme, which we then use to introduce more technical concepts and definitions. Ultimately, we want to see if this and other proposed election schemes satisfies certain properties, with a focus on ballot secrecy and verifiability.

Recall that the election scheme proposed by Smyth, Frink and Clarkson [103] uses four algorithms or primitives, denoted by Setup, Vote, Tally, and Verify, as illustrated in Figure 1. The *bulletin board* is where encrypted votes are stored until the election ends. We now describe the function of each primitive in more detail and then in the next section give more specific details.

- Setup randomly creates a public and private key pair, where the voters will use the public key and the election authority will use the private key. For illustration purposes, we only consider one key pair, but we can easily have two or more key pairs, and each candidate

(such as a political party) in the election keeps their own private key secret. Setup is simply an asymmetric encryption scheme that is suitably homomorphic.

- Vote allows the voters to vote using the public key. Each voter's vote is encrypted and the Vote primitive must use homomorphic asymmetric encryption with vanishingly small probability of collisions, meaning each voter has a unique but secret encrypted vote. If more than one key pair was created by the primitive Setup, then each voter must use the corresponding public keys to vote.
- Tally adds up the encrypted votes, which is possible with a homomorphic asymmetric encryption scheme. This primitive outputs the election outcome as well as proof for verifying that the election outcome is the true outcome based on the encrypted votes produced by the voters using Vote. Tally does not need the private key, originally generated by Setup, to tally the votes, but it *does* need the private key to produce a proof.
- Verify uses the public key or keys from the primitive Setup and a proof generated by the primitive Tally to verify the election outcome, also from Tally, is correct. Verify outputs *yes* or *no* to answer the question whether election integrity has been kept.

Fortunately these four primitives can be represented as four probabilistic polynomial-time algorithms. We have omitted a couple of system parameters that flow between the primitives, such as the maximum number of ballots, which can easily be made large enough to eliminate all limitations, but these are not crucial for security purposes. We will soon formally detail the specifics of these four primitives, but first we introduce some notation.

7 Election scheme syntax

In Section 6 we gave an informal overview of a proposed election scheme, illustrated in Figure 1. The *tallier* is the entity (or group of entities) responsible for generating election keys and computing the final tally from collected ballots. In the schemes we consider, the tallier runs the Setup and Tally algorithms and possesses the secret key needed for decryption. More sophisticated schemes distribute the tallier's role among multiple parties using threshold encryption (Section 3.7), but we defer such extensions to future work.

First, the tallier generates a key pair using Setup. Secondly, each voter constructs and casts a ballot for their vote using Vote. All the cast ballots are collected and recorded on a bulletin board. Third, the tallier tallies the collected ballots and announces an outcome as a tally of the votes by using Tally. The elected candidate is derived from the vote count by using some voting system, such as the candidate with the most votes. (We note for the first-past-the-post voting systems, Smyth [97] has shown that the syntax can model ranked-choice voting systems too.) Finally, voters and other interested parties check that the election outcome corresponds to the votes expressed in collected ballots by using Verify. We now give specific syntax that captures the four steps of this voting system.

Definition 2 (Election scheme [103])**.** *An* election scheme *is a tuple of probabilistic polynomial-time algorithms* $(\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$ *such that:*

Setup*, denoted* $(pk, sk, mb, mc) \leftarrow \mathsf{Setup}(\kappa)$*, is run by the tallier. The algorithm takes a security parameter* κ *as input and outputs a key pair* (pk, sk)*, a maximum number of ballots* mb*, and a maximum number of candidates* mc*.*

Vote, *denoted* $b \leftarrow \mathsf{Vote}(pk, v, nc, \kappa)$*, is run by voters. The algorithm takes as input a public key* pk*, a voter's vote* v*, some number of candidates* nc*, and a security parameter* κ*. Vote* v *should be selected from a sequence* $1, \ldots, nc$ *of candidates. The algorithm outputs an encrypted ballot* b *or error symbol* $\perp$*.*

Tally, *denoted* $(\mathfrak{v}, pf) \leftarrow \mathsf{Tally}(sk, \mathfrak{bb}, nc, \kappa)$*, is run by the tallier. The algorithm takes as input a private key* sk*, a bulletin board* $\mathfrak{bb}$ *(of encrypted ballots), some number of candidates* nc*, and a security parameter* κ*. The algorithm outputs an election outcome* $\mathfrak{v}$ *and a non-interactive tallying proof* pf*. The election outcome must be a vector of length* nc *and each index* v *of that vector should indicate the number of votes for candidate* v*. Moreover, the tallying proof should demonstrate that the outcome corresponds to votes expressed in ballots on the bulletin board.*

Verify, *denoted* $s \leftarrow \mathsf{Verify}(pk, \mathfrak{bb}, nc, \mathfrak{v}, pf, \kappa)$*, is run to audit an election. The algorithm takes as input a public key* pk*, a bulletin board* $\mathfrak{bb}$*, some number of candidates* nc*, an election outcome* $\mathfrak{v}$*, a tallying proof* pf*, and a security parameter* κ*. The algorithm outputs a bit* s*, which is* 1 *if the outcome should be accepted and* 0 *otherwise. We require the algorithm to be deterministic.*

Beyond having proper syntax, election schemes must satisfy *correctness*: the probability of an incorrect election outcome should be negligible. We now state this notion formally.

Definition 3 (Election correctness)**.** *An election scheme satisfies correctness if there exists a negligible function* negl*, such that for all security parameters* κ*, integers* nb *and* nc*, and votes* $v_1, \ldots, v_{nb} \in \{1, \ldots, nc\}$*, it holds that, given a zero-filled vector* $\mathfrak{v}$ *of length* nc*, such that:*
$\Pr[(pk, sk, mb, mc) \leftarrow \mathsf{Setup}(\kappa);$

for $1 \leq i \leq nb$ **do**
  $b_i \leftarrow \mathsf{Vote}(pk, v_i, nc, \kappa);$
  $\mathfrak{v}[v_i] \leftarrow \mathfrak{v}[v_i] + 1;$
$(\mathfrak{v}', pf) \leftarrow \mathsf{Tally}(sk, \{b_1, \ldots, b_{nb}\}, nc, \kappa) : ((nb \leq mb) \wedge (nc \leq mc)) \Rightarrow (\mathfrak{v} = \mathfrak{v}')] > 1 - \mathsf{negl}(\kappa).$

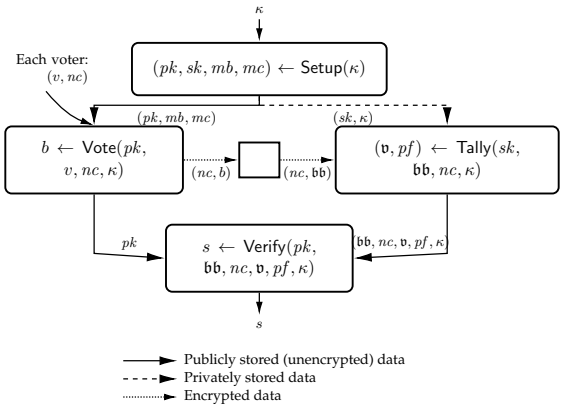
The above definition of correctness means that, given an election setup created by $\mathsf{Setup}(\kappa)$, for all nb voters, each voter i casts their vote v_i by using the $\mathsf{Vote}(pk, v, nc, \kappa)$ primitive, and then the votes are summed up, giving an outcome $\mathfrak{v}$. Then the $\mathsf{Tally}(sk, \mathfrak{bb}, nc, \kappa)$ primitive also returns a vote count $\mathfrak{v}'$ and a proof pf. Provided the number of ballots nb is less than the ballot maximum mb and the number of candidates nc is less than the candidate maximum mc, we then say the election scheme is correct if the probability of two outcomes $\mathfrak{v}$ and $\mathfrak{v}'$ being the same is greater than one minus a negligible function of the security parameter, meaning the probability of the two outcomes being different is negligible.

The syntax bounds the number of ballots mb and candidates mc to broaden the scope of the correctness definition. For example, the voting scheme Helios requires mb and mc to be less than or equal to the size of the underlying encryption scheme's message space, and represents votes as integers, rather than alphanumeric strings, for brevity.

This syntax allows us to model voting systems, while correctness ensures that election outcomes correspond to collected votes when constructed and tallied as prescribed. We will use our syntax to express verifiability and secrecy properties of election schemes. Then we will model and study implemented electronic voting systems.

Figure 5 illustrates the data flow between the four algorithms of Definition 2.

Figure 5 Detailed view of the election scheme. The four algorithms Setup, Vote, Tally, and Verify are defined in Definition 2. Dashed arrows indicate secret data (the private key sk); dotted arrows indicate encrypted ballots.



8 Voting

Now that we have formally defined election schemes and their syntax, we can examine the threats these systems face. Electronic voting systems are vulnerable to a range of security attacks that can undermine either ballot secrecy or election integrity. Understanding these attacks is essential for designing robust systems and for appreciating why the formal security properties we define in subsequent sections matter. We categorize attacks by their primary target and mechanism.

8.1 Attacks

8.1.1 Coercion attacks

In a *coercion attack* [62], an adversary forces a voter to cast a particular vote against their will. The coercer may be present during voting, may demand proof of how the voter voted, or may threaten consequences for non-compliance. Coercion attacks are particularly concerning because they can occur outside the voting system itself, making them difficult to prevent through technical means alone.

Several variants of coercion attacks exist:

- *Forced abstention*: The coercer prevents a voter from participating in the election entirely, for example by confiscating credentials or physically preventing access to voting.
- *Randomization attack*: The coercer instructs the voter to use a specific randomization method when choosing their vote, such as flipping a coin. This allows the coercer to manipulate election outcomes statistically without knowing individual votes.
- *Simulation attack*: The coercer obtains the voter's credentials (such as private keys) after registration but before voting, then casts a vote on behalf of the voter.

8.1.2 Vote-selling

Closely related to coercion is *vote-selling*, where a voter willingly sells their vote to a buyer. Unlike coercion, vote-selling involves a voluntary transaction, but it similarly undermines the principle

that votes should reflect genuine preferences. Vote-selling requires a mechanism for the buyer to verify how the seller voted, which creates tension with ballot secrecy: strong secrecy makes verification difficult, while receipts that enable verification also enable vote-selling.

8.1.3 Ballot manipulation attacks

Several attacks target the ballots themselves:

- *Ballot copying attack*: An adversary copies another voter's encrypted ballot and submits it as their own. If successful, this duplicates the copied voter's choice without the adversary knowing what that choice was. This attack exploits malleable encryption schemes.
- *Modification attack*: An adversary intercepts a ballot in transit and modifies it, changing the encoded vote. This requires the ability to meaningfully alter ciphertexts, which non-malleable encryption prevents.
- *Clash attack* [73]: Two or more voters' ballots produce the same identifier or hash value, making it impossible to distinguish them on the bulletin board. An adversary can exploit collisions to substitute ballots or cause confusion during verification.

8.1.4 Infrastructure attacks

- *Man-in-the-middle attack*: An adversary positions themselves between the voter and the voting server, intercepting and potentially modifying communications in both directions. The adversary can alter votes, steal credentials, or present false information to voters.
- *Brute-force attack*: When the space of possible votes is small (as in most elections with few candidates), an adversary who obtains encrypted ballots may attempt to decrypt them by encrypting all possible votes and comparing the results. Probabilistic encryption defends against this by ensuring identical plaintexts produce different ciphertexts.

8.2 Countermeasures

Various countermeasures have been developed to defend against the attacks described above.

8.2.1 Ballot weeding

Weeding is a technique where duplicate or malformed ballots are removed from the bulletin board before tallying. This defends against ballot copying attacks by ensuring that each valid ballot appears exactly once. Weeding can be performed by checking that each voter credential is used at most once, or by detecting duplicate ciphertexts.

8.2.2 Non-malleable encryption

Using *non-malleable encryption* [63] prevents adversaries from meaningfully modifying ciphertexts. If an adversary cannot transform a ciphertext encrypting vote v into a ciphertext encrypting a related vote v', then modification and ballot copying attacks become infeasible.

Figure 6 Taxonomy of attacks on electronic voting systems, the security properties they violate, and corresponding countermeasures. Properties in the middle column correspond to formal definitions: Ballot secrecy (Section 10.1), Universal and Individual verifiability (Section 9), and Non-malleability (Section 4.2). Receipt-freeness and coercion-resistance are discussed in Section 10.5.

Attack	Property Violated	Countermeasure
Coercion attacks		
Coercion	Coercion-resistance	Coercion-resistant protocols
Forced abstention	Coercion-resistance	Credential recovery; deniable voting
Randomization	Coercion-resistance	Coercion-resistant protocols
Simulation	Coercion-resistance	Credential binding
Vote-selling	Receipt-freeness	Receipt-free protocols
Ballot manipulation attacks		
Ballot copying	Ballot secrecy; Non-malleability	Ballot weeding; NM-CPA encryption
Modification	Universal verifiability	Non-malleable encryption; ZK proofs
Clash attack	Individual verifiability	Collision-resistant hashing
Infrastructure attacks		
Man-in-the-middle	Ballot secrecy; Verifiability	Authenticated channels; ZK proofs
Brute-force	Ballot secrecy	Probabilistic encryption

8.2.3 Zero-knowledge proofs

Requiring voters to provide *zero-knowledge proofs* that their ballots are well-formed (encrypting a valid vote) prevents submission of malformed ballots that might exploit vulnerabilities in the tallying process.

8.2.4 Receipt-freeness

A voting system is *receipt-free* [12] if voters cannot prove to a third party how they voted. Receipt-freeness defends against both vote-selling (the buyer cannot verify the seller's compliance) and coercion (the voter can claim to have voted as instructed while actually voting freely). Achieving receipt-freeness while maintaining verifiability is challenging, as both properties involve what voters can prove.

8.2.5 Coercion-resistance

Stronger than receipt-freeness, *coercion-resistance* [62] ensures that even if a coercer actively participates in the voting process (for example, by providing credentials or observing the voter), the voter can still cast their intended vote. This typically requires mechanisms for voters to cast "fake" votes that satisfy the coercer but are later replaced or ignored.

8.3 Attacks and countermeasures summary

Figure 6 summarizes the relationships between attacks, the security properties they violate, and their countermeasures. This taxonomy illustrates why election schemes require multiple secu-

rity properties working in concert: defending against one class of attacks may leave the system vulnerable to others.

A recurring suggestion is to use blockchain technology to secure elections, for example by recording ballots on a distributed ledger to prevent tampering. However, blockchains do not address the fundamental challenges of ballot secrecy and software independence. As Rivest remarked at the RSA Conference in 2020:

> "Blockchain is the wrong security technology for voting. I like to think of it as bringing a combination lock to a kitchen fire."
>
> —Ronald Rivest [88]

Park, Specter, Narula and Rivest [86] argue that blockchain-based voting would greatly increase the risk of undetectable, nation-scale election failures, and that any convenience gains would come at the cost of losing meaningful assurance that votes have been counted as cast.

9 Verifiability

To ensure that election outcomes correctly correspond to the votes expressed in collected ballots, the Australian election system relies upon monitoring. The mere depositing of ballots into ballot boxes is enough to ensure that they are collected. (Though several years ago the state of Western Australia lost more than a thousand ballot papers in a federal election.)

This approach is in direct contrast with electronic election schemes, which compute election outcomes in a manner that should not be monitored. Otherwise such monitoring would reveal the tallier's private key, compromising the ballot secrecy. Furthermore, just casting a ballot is insufficient to ensure it is collected, because an adversary may discard or modify ballots. Nevertheless, election schemes generate tallying proofs to provide evidence that the election outcomes are correctly computed, as well as allowing voters to check whether their ballot are properly collected. These two concepts are formalized, respectively, by universal verifiability and individual verifiability.

Before formalizing these concepts, we briefly survey the broader landscape of verifiability concepts. Researchers have proposed different definitions of verifiability. For example, voting systems may satisfy a stronger notion of universal verifiability: anyone can check whether an outcome corresponds to votes expressed in collected ballots that are authorized, except votes cast by the same voter. A related concept is *unforgeability*: only voters can construct authorized ballots. (Previously Smyth [103] used the term *eligibility verifiability* as a synonym for *unforgeability*, but we prefer the latter.)

Verifiability is often decomposed into finer-grained properties [34], including *cast-as-intended*, *stored-as-cast*, and *tallied-as-stored* verifiability, which together comprise *end-to-end verifiability*. Beyond verifying that votes are counted correctly, *eligibility verifiability* ensures only authorized voters can cast ballots. In this tutorial, we focus on formalizing individual and universal verifiability as defined below.

9.1 Universal verifiability

Universal verifiability asserts that anyone must be able to check whether an election outcome corresponds to the votes expressed in collected ballots. This can be expressed in terms of *reachability*: whether a system, like a game, equipped with a set of rules, can reach a certain state.

Since checks can be performed by algorithm Verify, it is sufficient that the algorithm accepts if and only if the outcome corresponds to votes expressed in the collected ballots. The *only if* requirement is captured by *soundness*, which requires algorithm Verify to only accept correct outcomes, and the *if* requirement is captured by completeness, which requires election outcomes produced by algorithm Tally to be accepted by algorithm Verify.

Soundness. Soundness captures the requirement that verification should reject incorrect outcomes. An adversary attempting to announce a fraudulent tally should be unable to produce a proof that passes verification. The formal definition tasks an adversary to produce any inputs, such as public key, bulletin board, outcome, and proof, that cause verification to accept an outcome that does not match the votes actually present in the ballots.

We formalize correct outcomes by using the function $\mathit{correct\text{-}outcome}$. Recall that the counting quantifier $(\exists^{=\ell} x : P(x))$ expresses that exactly ℓ distinct values of x satisfy property $P(x)$.

Using this predicate, function $\mathit{correct\text{-}outcome}$ is defined by

$$\mathit{correct\text{-}outcome}(pk, nc, \mathbb{bb}, \kappa)[v] = \ell \iff \exists^{=\ell} b \in \mathbb{bb} \setminus \{\bot\} : \exists r : b = \mathsf{Vote}(pk, v, nc, \kappa; r),$$

where $\mathit{correct\text{-}outcome}(pk, nc, \mathbb{bb}, \kappa)$ is a vector of length nc and $1 \leq v \leq nc$, and where r denotes the random coins used by the probabilistic algorithm $\mathsf{Vote}(pk, v, nc, \kappa)$. We see that component v of vector $\mathit{correct\text{-}outcome}(pk, nc, \mathbb{bb}, \kappa)$ equals ℓ if and only if there exist ℓ ballots for vote v in the bulletin board. The function requires that ballots be interpreted for only one candidate, which we can ensure with the concept of *injectivity*.

Definition 4 (Injectivity [98, 103])**.** *An election scheme* (Setup, Vote, Tally, Verify) *satisfies* Injectivity*, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, security parameters κ and computations $(pk, nc, v, v') \leftarrow \mathcal{A}(\kappa); b \leftarrow \mathsf{Vote}(pk, v, nc, \kappa); b' \leftarrow \mathsf{Vote}(pk, v', nc, \kappa)$ such that $(v \neq v') \wedge (b \neq \bot) \wedge (b' \neq \bot)$, we have $b \neq b'$.*

We use a definition of injectivity that ensures that a ballot for vote v can never be interpreted for another vote v', meaning the votes expressed in ballots correspond to unique outcomes.

Equipped with a notion of correct outcomes, we can formalize *soundness*.

Definition 5 (Soundness [103])**.** *Let $\Gamma = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$ be an election scheme, $\mathcal{A}$ be an adversary, κ be a security parameter, and $\mathsf{Soundness}(\Gamma, \mathcal{A}, \kappa)$ be the following game.*

$\mathsf{Soundness}(\Gamma, \mathcal{A}, \kappa) =$

1 $(pk, \mathbb{bb}, nc, \mathfrak{v}, pf) \leftarrow \mathcal{A}(\kappa)$;
2 **return** $(\mathsf{Verify}(pk, \mathbb{bb}, nc, \mathfrak{v}, pf, \kappa) = 1) \wedge (\mathfrak{v} \neq \mathit{correct\text{-}outcome}(pk, nc, \mathbb{bb}, \kappa))$;

We say election scheme Γ satisfies Soundness*, if Γ satisfies injectivity and for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function* negl*, such that for all security parameters κ, we have* $\mathsf{Succ}(\mathsf{Soundness}(\Gamma, \mathcal{A}, \kappa)) \leq \mathsf{negl}(\kappa)$.

In Definition 5, soundness is defined as a game that tasks the adversary to compute inputs to algorithm Verify (Line 1), including an election outcome and some ballots, that cause the algorithm to accept when the outcome does not correspond to the votes expressed in those ballots (Line 2). When algorithm Verify only accepts election outcomes that correspond to votes expressed in collected ballots, we say that an election scheme satisfies Soundness.

Design guideline 1. *Verification must only accept outcomes that correspond to votes expressed in collected ballots.*

Completeness. Completeness is the dual of soundness: it ensures that honestly-computed outcomes are always accepted. Without completeness, a correct election could be rejected, allowing a malicious party to void legitimate results. The formal definition tasks an adversary to produce a bulletin board for which the honestly-computed tally fails verification.

Definition 6 (Completeness [103]). *Let* $\Gamma = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$ *be an election scheme,* $\mathcal{A}$ *be an adversary,* κ *be a security parameter, and* $\mathsf{Completeness}(\Gamma, \mathcal{A}, \kappa)$ *be the following game.*

$\mathsf{Completeness}(\Gamma, \mathcal{A}, \kappa) =$

1 $(pk, sk, mb, mc) \leftarrow \mathsf{Setup}(\kappa)$;
2 $(\mathfrak{bb}, nc) \leftarrow \mathcal{A}(pk, \kappa)$;
3 $(\mathfrak{v}, pf) \leftarrow \mathsf{Tally}(sk, \mathfrak{bb}, nc, \kappa)$;
4 **return** $(\mathsf{Verify}(pk, \mathfrak{bb}, nc, \mathfrak{v}, pf, \kappa) \neq 1) \wedge (|\mathfrak{bb}| \leq mb) \wedge (nc \leq mc)$;

We say election scheme Γ *satisfies* Completeness*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a negligible function* negl*, such that for all security parameters* κ*, we have* $\mathsf{Succ}(\mathsf{Completeness}(\Gamma, \mathcal{A}, \kappa)) \leq \mathsf{negl}(\kappa)$.

In Definition 6 we formalize completeness as a game that, given the key pair is computed by algorithm Setup (Line 1), tasks the adversary to compute a bulletin board and some number of candidates (Line 2) such that the corresponding election outcome computed by algorithm Tally (Line 3) is rejected by algorithm Verify (Line 4).

An election scheme satisfies Completeness when algorithm Verify accepts outcomes computed by algorithm Tally, for key pairs computed by algorithm Setup. It follows that completeness implies an aspect of accountability. Indeed, if verification fails, then the tallier is responsible for that failure, in particular, they must have incorrectly computed their key pair or the election outcome.

Design guideline 2. *Tallying must produce outcomes that will be accepted during verification.*

We formalize universal verifiability by combining the above notions.

Definition 7 (Universal-Verifiability [98, 103]). *An election scheme* Γ *satisfies* Universal-Verifiability*, if* Soundness *and* Completeness *are satisfied.*

9.2 Individual verifiability

Individual verifiability ensures that voters can confirm their ballot appears in the election record. Without this property, a malicious system could silently discard votes. The key challenge is that merely seeing a ballot on the bulletin board is insufficient. Rather, a voter must be confident it is *their* ballot, not a coincidentally identical one constructed by someone else. This requires ballots to be uniquely identifiable.

Individual verifiability asserts that voter Alice must be able to check whether her ballot is among those that have been collected. Given that ballots should be collected and recorded on a bulletin board, which must be available to everyone, then it is enough for voters to be able to check that their ballots (which they constructed) are on the bulletin board. This means that voters must be able to check that their ballots have not been omitted from the bulletin board. Nevertheless, this alone is not sufficient, because the existence of a ballot identical to an individual voter's ballot does not imply that the ballot constructed by the voter exists. In other words, if a ballot identical to voter Alice's exists, it does not mean Alice's vote exists, because Alice's ballot might have been constructed by another voter.

Consequently, individual verifiability requires that voters must be able to uniquely identify their ballots, so that the ballots have not collided. We now give a definition of individual verifiability that captures this requirement.

Definition 8 (Individual verifiability [103]). *Let* $\Gamma = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$ *be an election scheme,* $\mathcal{A}$ *be an adversary,* κ *be a security parameter, and* $\mathsf{Individual\text{-}Verifiability}(\Gamma, \mathcal{A}, \kappa)$ *be the following game.*

$\mathsf{Individual\text{-}Verifiability}(\Gamma, \mathcal{A}, \kappa) =$

1 $(pk, nc, v, v') \leftarrow \mathcal{A}(\kappa)$;
2 $b \leftarrow \mathsf{Vote}(pk, nc, v, \kappa)$;
3 $b' \leftarrow \mathsf{Vote}(pk, nc, v', \kappa)$;
4 **return** $(b = b') \wedge (b \neq \bot) \wedge (b' \neq \bot)$;

We say election scheme Γ *satisfies* Individual-Verifiability, *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a negligible function* negl, *such that for all security parameters* κ*, the inequality*

$$\mathsf{Succ}(\mathsf{Individual\text{-}Verifiability}(\Gamma, \mathcal{A}, \kappa)) \leq \mathsf{negl}(\kappa)\,,$$

holds.

In Definition 8, we formalize individual verifiability as a game that tasks the adversary to compute inputs to algorithm Vote (Line 1) that cause the algorithm to output ballots (Lines 2 & 3) that collide (Line 4).

When algorithm Vote generates uniquely identifiable ballots, meaning ballots that do not collide, then we say an election scheme satisfies Individual-Verifiability. Elaborating further, the respective concepts of correctness, individual verifiability and injectivity all hinge upon ballots not colliding, but they do so under different assumptions. Correctness requires that ballots do not collide, with overwhelming probability, for public keys computed by algorithm Setup; Injectivity requires that ballots for distinct votes never collide; and Individual-Verifiability requires that ballots do not collide with overwhelming probability. Consequently, Individual-Verifiability implies that ballots do not collide in the context of correctness. But Individual-Verifiability and Injectivity are in a sense orthogonal to each other. Specifically, Individual-Verifiability allows collisions with negligible probability, whereas Injectivity allows collisions between ballots for the same vote.

The recurring requirement of no ballot collisions motivates another design guideline.

Design guideline 3. *Ballots must be distinct.*

10 Ballot secrecy

As noted in Section 1, we prefer the term *ballot secrecy* over *privacy* to avoid confusion with other privacy notions.

The Australian system relies upon uniform ballots to ensure voters' votes are not revealed. This uniformity ensures that ballots are indistinguishable during distribution, whereas the isolated polling booths ensures votes are not revealed while marking. Folded ballots are indistinguishable during collection and indistinguishability of markings can ensure votes are not revealed while tallying. In short, the Australian system derives ballot secrecy from physical characteristics of the world. By comparison, electronic election schemes cannot rely on such physical characteristics, so they must rely on cryptography to ensure voters' votes are not revealed. (Admittedly, some electronic voting systems do rely upon physical characteristics. For instance, MarkPledge [79],

Pret à Voter [29], Remotegrity [113], Scantegrity II [27] and Three Ballot [90] all use features implemented with paper, such as scratch-off surfaces and detachable columns. But these systems fall outside the scope of our election scheme syntax.)

Some scenarios inevitably reveal voters' votes: Unanimous election outcomes reveal how everyone voted and, more generally, outcomes can be coupled with partial knowledge of voters' votes to deduce voters' votes. For example, suppose Alice, Bob and Mallory participate in a referendum and the outcome has frequency two for 'yes' and one for 'no.' Mallory and Alice can deduce Bob's vote by pooling knowledge of their own votes. Similarly, Mallory and Bob can deduce Alice's vote. Furthermore, Mallory can deduce that Alice and Bob both voted yes, if she voted no. For simplicity, our informal definition of ballot secrecy (§1) deliberately omitted side-conditions that exclude these inevitable revelations and that are necessary for satisfiability. We now refine that definition as follows:

> A voter's vote is not revealed to anyone, except when the vote can be deduced from the election outcome and any partial knowledge of voters' votes.

This refinement ensures the aforementioned examples are not violations of ballot secrecy. By comparison, if Mallory votes yes and she can deduce the vote of Alice, without knowledge of Bob's vote, then ballot secrecy is violated.

10.1 Security definition

The core idea behind ballot secrecy is an *indistinguishability test*. Imagine two parallel elections that are identical in every way, except that two voters, say Alice and Bob, swap their votes. In one election Alice votes for candidate 1 and Bob for candidate 2; in the other, Alice votes for candidate 2 and Bob for candidate 1. Every other voter casts the same vote in both elections. Because the same collection of votes is cast in total, the election outcome is identical in both cases.

Then we ask, can anyone, by examining the encrypted ballots, the tally, and the proof, determine which of the two elections actually took place? If not, then the scheme keeps ballots secret: no one can link a specific vote to a specific voter, even with access to all public election data. The formal game below captures exactly this: a secret coin flip determines which election is run, an adversary sees everything that is made public, and ballot secrecy holds if no efficient adversary (meaning a probabilistic polynomial-time one) can guess the coin with probability meaningfully better than $\frac{1}{2}$.

We formalize ballot secrecy (Definition 9) as a game that tasks the adversary to: select two lists of votes; construct a bulletin board from ballots for votes in one of those lists, which list is decided by a coin flip; and (non-trivially) determine the result of the coin flip from the resulting election outcome and tallying proof. That is, the game tasks the adversary to distinguish between an instance of the voting system for one list of votes, from another instance with the other list of votes, when the votes cast from each list are permutations of each other (hence, the distinction is non-trivial). The game proceeds as follows: The challenger generates a key pair (Line 1), the adversary chooses some number of candidates (Line 2), and the challenger flips a coin (Line 3) and initialises a set to record lists of votes (Line 4). The adversary computes a bulletin board from ballots for votes in one of two possible lists (Line 5), where the lists are chosen by the adversary, the choice between lists is determined by the coin flip, and the ballots (for votes in one of the lists) are constructed by an oracle (further ballots may be constructed by the adversary). The challenger tallies the bulletin board to derive the election outcome and tallying proof (Line 6), which are given to the adversary and the adversary is tasked with determining the result of the coin flip (Line 7 & 8).

The formal definition includes a *balanced condition* that prevents trivial attacks. An adversary querying the oracle with input $(1, 2)$ receives a ballot b that encrypts either vote 1 or vote 2, depending on the secret bit β. Placing only this ballot on the bulletin board would let the adversary win by simply observing whether candidate 1 or 2 received a vote. The balanced condition excludes such trivial wins by requiring that the votes for $\beta = 0$ be a permutation of the votes for $\beta = 1$, ensuring both cases produce the same election outcome.

As an example, an adversary making queries $(1, 2)$, $(2, 1)$, and $(3, 3)$ receives ballots b_1, b_2, b_3. A bulletin board containing all three is balanced: when $\beta = 0$ the votes are $1, 2, 3$, and when $\beta = 1$ they are $2, 1, 3$—both yield one vote per candidate. The adversary must exploit some weakness in the voting scheme itself to win, not merely observe the outcome. Omitting b_2 and b_3 would make the board unbalanced (one vote for candidate 1 versus one for candidate 2), and such configurations are excluded.

Definition 9 (Ballot-Secrecy [96])**.** *Let* $\Gamma = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$ *be an election scheme,* $\mathcal{A}$ *be an adversary,* κ *be a security parameter, and* $\mathsf{Ballot\text{-}Secrecy}(\Gamma, \mathcal{A}, \kappa)$ *be the following game.*

$\mathsf{Ballot\text{-}Secrecy}(\Gamma, \mathcal{A}, \kappa) =$

1 $(pk, sk, mb, mc) \leftarrow \mathsf{Setup}(\kappa)$;
2 $nc \leftarrow \mathcal{A}(pk, \kappa)$;
3 $\beta \leftarrow_R \{0, 1\}$;
4 $L \leftarrow \emptyset$;
5 $\mathfrak{bb} \leftarrow \mathcal{A}^{\mathcal{O}}()$;
6 $(\mathfrak{v}, pf) \leftarrow \mathsf{Tally}(sk, \mathfrak{bb}, nc, \kappa)$;
7 $g \leftarrow \mathcal{A}(\mathfrak{v}, pf)$;
8 **return** $(g = \beta) \wedge \mathit{balanced}(\mathfrak{bb}, nc, L) \wedge (1 \leq nc \leq mc) \wedge (|\mathfrak{bb}| \leq mb)$;

Predicate $\mathit{balanced}(\mathfrak{bb}, nc, L)$ *holds when: for all votes* $v \in \{1, \ldots, nc\}$ *we have*

$$|\{b \mid b \in \mathfrak{bb} \wedge \exists v_1 \,.\, (b, v, v_1) \in L\}| = |\{b \mid b \in \mathfrak{bb} \wedge \exists v_0 \,.\, (b, v_0, v) \in L\}| \,.$$

Oracle $\mathcal{O}$ *is defined as follows:*

- $\mathcal{O}(v_0, v_1)$ *computes* $b \leftarrow \mathsf{Vote}(pk, v_\beta, nc, \kappa); L \leftarrow L \cup \{(b, v_0, v_1)\}$ *and outputs* b*, where* $v_0, v_1 \in \{1, ..., nc\}$.

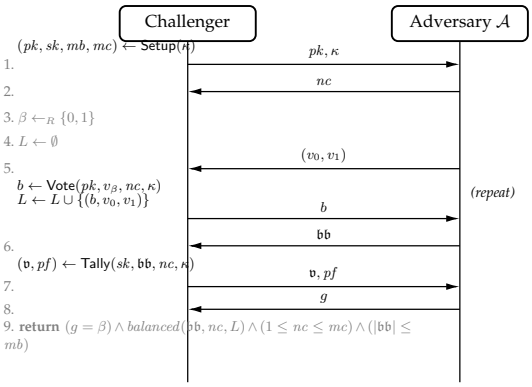
We say election scheme Γ *satisfies* Ballot-Secrecy*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a negligible function* negl*, such that for all security parameters* κ*, the inequality*

$$\mathsf{Succ}(\mathsf{Ballot\text{-}Secrecy}(\Gamma, \mathcal{A}, \kappa)) \leq \frac{1}{2} + \mathsf{negl}(\kappa) \,,$$

holds.

Figure 7 illustrates the interactions between challenger and adversary in the ballot secrecy game. An election scheme delivers ballot secrecy when the voting algorithm Vote produces ballots that conceal voters' choices, and when the tallying algorithm Tally generates outcomes and proofs without exposing how individual ballots map to the final result. In the game Ballot-Secrecy, the adversary must construct a bulletin board using ballots that an oracle creates for votes from one of two possible lists. The adversary then attempts to identify which list was chosen, based solely on the election outcome and tallying proof derived from that board. The choice between lists is determined by the result β of a coin flip, and the oracle outputs a ballot for vote v_β on input of a pair of votes v_0, v_1. Hence, the oracle constructs ballots for votes in one of two possible lists, where

Figure 7 The ballot secrecy game (Definition 9). The challenger generates keys and samples a secret bit β. The adversary repeatedly queries an oracle with vote pairs (v_0, v_1) and receives a ballot encrypting v_β; the set L records these queries. After submitting a bulletin board $\mathfrak{bb}$, the adversary receives the tally and proof, then guesses β. The adversary wins if the guess is correct and the bulletin board is balanced (the multiset of left votes equals the multiset of right votes).



the lists are chosen by the adversary, and the bulletin board may contain those ballots in addition to ballots constructed by the adversary.

Election schemes reveal the number of votes for each candidate, meaning the election outcome). To avoid trivial distinctions in game Ballot-Secrecy, we require that runs of the game are *balanced*: inputs to the oracle for $\beta = 0$ are equivalent to inputs for $\beta = 1$, when the corresponding outputs appear on the bulletin board. For example, suppose the inputs to the oracle are $(v_{1,0}, v_{1,1}), \ldots, (v_{n,0}, v_{n,1})$ and the corresponding outputs are $b_1, \ldots, b_n$, further suppose the bulletin board is $\{b_1, \ldots, b_\ell\}$ such that $\ell \leq n$. That game is balanced if the inputs for $\beta = 0$, namely $v_{1,0}, \ldots, v_{\ell,0}$, are a permutation of the inputs for $\beta = 1$, namely $v_{1,1}, \ldots, v_{\ell,1}$. To avoid trivial wins, we require the balanced condition. Consider an adversary who constructs a bulletin board with only the ballot from oracle query $(1, 2)$. Although such an adversary could easily identify β by observing whether candidate 1 or 2 received a vote, this configuration violates the balanced condition and therefore doesn't count as a valid win. (We leave formally defining a winning adversary as an exercise.)

We can easily see the intuition behind the game Ballot-Secrecy. A winning adversary possesses a method for distinguishing ballots. Such an adversary can tell apart a voting system instance where voters submit certain votes from another instance where they submit a permutation of those same votes—thereby exposing voters' choices. Otherwise, the adversary is unable to distinguish between a voter casting a ballot for vote v_0 and another voter casting a ballot for vote v_1, hence, voters' votes cannot be revealed.

10.2 Example

We outlined asymmetric encryption schemes in Section 3.2, while more precise details are given in Section 3.2 of the appendix. Using an asymmetric scheme, we now model our Enc2Vote voting system (§1) as the following election scheme.

Definition 10 (Enc2Vote [96]). *Given an asymmetric encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, *we define* $\mathsf{Enc2Vote}(\Pi) = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$ *such that:*

- $\mathsf{Setup}(\kappa)$ *computes* $(pk, sk, \mathcal{M}) \leftarrow \mathsf{Gen}(\kappa); pk' \leftarrow (pk, \mathcal{M}); sk' \leftarrow (pk, sk)$, *derives* mc *as the*

largest integer such that $\{0,\ldots,mc\} \subseteq \{0\} \cup \mathcal{M}$ *and for all* $m_0, m_1 \in \{1,\ldots,mc\}$ *we have* $|m_0| = |m_1|$*, and outputs* $(pk', sk', p(\kappa), mc)$*, where* p *is a polynomial function.*

- $\mathsf{Vote}(pk', v, nc, \kappa)$ *parses* pk' *as pair* $(pk, \mathcal{M})$*, outputting* $\perp$ *if parsing fails or* $(v \notin \{1,\ldots,nc\}) \vee (\{1,\ldots,nc\} \not\subseteq \mathcal{M})$*, computes* $b \leftarrow \mathsf{Enc}(pk, v)$*, and outputs* b*.*
- $\mathsf{Tally}(sk', \mathfrak{bb}, nc, \kappa)$ *initialises* $\mathfrak{v}$ *as a zero-filled vector of length* nc*, parses* sk' *as pair* (pk, sk)*, outputting* $(\mathfrak{v}, \perp)$ *if parsing fails, computes* **for** $b \in \mathfrak{bb}$ **do** $v \leftarrow \mathsf{Dec}(sk, b)$; **if** $1 \leq v \leq nc$ **then** $\mathfrak{v}[v] \leftarrow \mathfrak{v}[v] + 1$*, and outputs* $(\mathfrak{v}, \epsilon)$*, where* ϵ *is a constant symbol.*
- $\mathsf{Verify}(pk, \mathfrak{bb}, nc, \mathfrak{v}, pf, \kappa)$ *outputs 1.*

Note that pk' and sk' have special structure in this construction: the election's public key $pk' = (pk, \mathcal{M})$ bundles the encryption public key with the message space, while the election's private key $sk' = (pk, sk)$ bundles both the encryption public key and private key.
To ensure $\mathsf{Enc2Vote}(\Pi)$ is an election scheme, we require asymmetric encryption scheme Π to produce distinct ciphertexts with overwhelming probability, otherwise correctness cannot be satisfied, as the following lemma demonstrates.

Lemma 1. *There exists an asymmetric encryption scheme* Π *such that* $\mathsf{Enc2Vote}(\Pi)$ *is not an election scheme.*

To prove our lemma, we show that colliding ciphertexts suffice to ensure that Enc2Vote cannot satisfy the correctness property of Definition 2.

Proof. Let $\mathsf{Enc2Vote}(\Pi) = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$. Suppose (pk', sk', mb, mc) is an output of $\mathsf{Setup}(\kappa)$ and b and b' are outputs of $\mathsf{Vote}(pk', v, nc, \kappa)$ such that $(2 \leq mb) \wedge (1 \leq v \leq nc \leq mc)$, where κ is a security parameter. Further suppose $\mathfrak{v}$ is a zero-filled vector of length nc, except for index v which contains the value 2. Moreover, suppose $(\mathfrak{v}', pf)$ is an output of $\mathsf{Tally}(sk', \{b, b'\}, nc, \kappa)$. If b and b' collide, then outcome $\mathfrak{v}'$ is computed from the set $\{b, b'\} = \{b\}$, therefore, the correct outcome cannot have been computed, implying $\mathfrak{v} \neq \mathfrak{v}'$ with a non-negligible probability, so correctness is not satisfied. By definition of algorithm Setup, pk' is a pair, and, by definition of algorithm Vote, b and b' are ciphertexts on plaintext v. Consequently, we now need to show that asymmetric encryption schemes can produce ciphertexts that collide. Indeed, they can: Consider an encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that $\mathsf{Enc}(pk, m)$ outputs m and $\mathsf{Dec}(sk, c)$ outputs c. Although Π is clearly not secure, it is straightforward to see that Π satisfies correctness, because $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m; r)) = m$ for all key pairs (pk, sk), plaintexts m, and coins r. □

It follows from Lemma 1 that we must restrict the class of asymmetric encryption schemes used to instantiate Enc2Vote. We could consider a broad class of schemes that produce distinct ciphertexts with overwhelming probability, but we favour the narrower class of non-malleable schemes, since we require non-malleability for ballot secrecy. For non-malleability in general, the definitions are complex and proofs are relatively difficult. This motivates us to adopt the definition of indistinguishability under parallel attack (IND-PA0) by Bellare & Sahai [9], which is simpler, yet equivalent to their definition of comparison based non-malleability (CNM-CPA). We recall the definition of IND-PA0 in Section D of the appendix.

Lemma 2. *If asymmetric encryption scheme* Π *satisfies* IND-PA0*, then* $\mathsf{Enc2Vote}(\Pi)$ *is an election scheme.*

To prove our lemma, we show that Enc2Vote satisfies the correctness property of Definition 2 when ciphertexts do not collide.

Proof. Let $\mathsf{Enc2Vote}(\Pi) = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$ and $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Moreover, let κ be a security parameter, nb and nc be integers, $v_1, \ldots, v_{nb} \in \{1, \ldots, nc\}$ be votes, and $\mathfrak{v}$ be a zero-filled vector of length nc. Suppose (pk', sk', mb, mc) is an output of $\mathsf{Setup}(\kappa)$ such that $(nb \leq mb) \wedge (nc \leq mc)$. Further suppose we compute **for** $1 \leq i \leq nb$ **do** $b_i \leftarrow \mathsf{Vote}(pk, v_i, nc, \kappa)$; $\mathfrak{v}[v_i] \leftarrow \mathfrak{v}[v_i] + 1$. Moreover, suppose $(\mathfrak{v}', pf)$ is an output of $\mathsf{Tally}(sk, \{b_1, \ldots, b_{nb}\}, nc, \kappa)$. To prove correctness, it suffices to prove $\mathfrak{v} = \mathfrak{v}'$, with overwhelming probability.

By definition of algorithm Setup, we have pk' is a pair $(pk, \mathcal{M})$ and sk' is a pair (pk, sk) such that $(pk, sk, \mathcal{M})$ was output by $\mathsf{Gen}(\kappa)$. Moreover, mc is the largest integer such that $\{0, \ldots, mc\} \subseteq \{0\} \cup \mathcal{M}$, hence, $\{1, \ldots, nc\} \subseteq \mathcal{M}$. It follows by definition of algorithm Vote that for each $i \in \{1, \ldots, nb\}$ we have b_i is an output of $\mathsf{Enc}(pk, v_i)$. By definition of algorithm Tally, outcome $\mathfrak{v}'$ is initialized as a zero-filled vector of length nc and computed as follows:

for $b \in \{b_1, \ldots, b_{nb}\}$ **do** $v \leftarrow \mathsf{Dec}(sk, b)$; **if** $1 \leq v \leq nc$ **then** $\mathfrak{v}'[v] \leftarrow \mathfrak{v}'[v] + 1$.

Since Π satisfies IND-PA0, ciphertexts $b_1, \ldots, b_{nb}$ are distinct with overwhelming probability, hence, that computation is equivalent to the following:

for $1 \leq i \leq nb$ **do** $v \leftarrow \mathsf{Dec}(sk, b_i)$; **if** $1 \leq v \leq nc$ **then** $\mathfrak{v}'[v] \leftarrow \mathfrak{v}'[v] + 1$.

Moreover, by correctness of Π, we have $\mathsf{Dec}(sk, b_i) = v_i$ for all $i \in \{1, \ldots, nb\}$, with overwhelming probability. Thus, the above computation is equivalent to computing

for $1 \leq i \leq nb$ **do** $\mathfrak{v}'[v_i] \leftarrow \mathfrak{v}'[v_i] + 1$,

with overwhelming probability. It follows that outcomes $\mathfrak{v}$ and $\mathfrak{v}'$ are computed identically, hence, $\mathfrak{v} = \mathfrak{v}'$, as required, with overwhelming probability. □

Intuitively, election scheme $\mathsf{Enc2Vote}(\Pi)$ satisfies ballot secrecy until tallying, because asymmetric encryption scheme Π can ensure that voters' votes are not revealed, and tallying maintains ballot secrecy by revealing only the election outcome.

Proposition 3. *If an asymmetric encryption scheme* Π *satisfies* IND-PA0*, then election scheme* $\mathsf{Enc2Vote}(\Pi)$ *satisfies* Ballot-Secrecy*.*

Proving this proposition and other ballot secrecy results is time consuming. Indeed, Quaglia & Smyth's ballot-secrecy proof for our simple Enc2Vote voting system fills over six and a half pages [87, Appendix C.6] and Cortier *et al.* devoted one person-year to their ballot-secrecy proof for Helios [39]. To reduce the expense of ballot-secrecy proofs, the following section introduces sufficient conditions that enable the simplification of game Ballot-Secrecy, which gives way to simpler proofs. Indeed, we prove Proposition 3 in just over a page.

10.3 Simplifying proofs

Tallying proofs may reveal voters' votes. For example, a variant of Enc2Vote might define tallying proofs that map ballots to votes. Hence, such proofs are rightly provided to the adversary in game Ballot-Secrecy (Line 7). Nevertheless, if tallying proofs reveal nothing about the votes expressed in ballots on the bulletin board, then they can be omitted from the game. This precondition is ensured by election schemes that use zero-knowledge tallying proofs. Thus, the adversary need not be provided with such proofs in game Ballot-Secrecy when analysing such schemes, which achieves our first reduction in the expense of ballot-secrecy proofs. Our second reduction involves modifying the computation of election outcomes.

Game Ballot-Secrecy computes the election outcome from ballots constructed by the oracle and ballots constructed by the adversary (Line 6). Intuitively, such an outcome can be equivalently computed as follows:

$(\mathfrak{v}, pf) \leftarrow \mathsf{Tally}(sk, \mathfrak{bb} \setminus \{b \mid (b, v_0, v_1) \in L\}, nc, \kappa)$;
$(\mathfrak{v}', pf') \leftarrow \mathsf{Tally}(sk, \mathfrak{bb} \cap \{b \mid (b, v_0, v_1) \in L\}, nc, \kappa)$;
$\mathfrak{v} \leftarrow \mathfrak{v} + \mathfrak{v}'$;

Yet, a poorly designed tallying algorithm might not ensure equivalence. In particular, ballots constructed by the adversary can cause the algorithm to behave unexpectedly. (Such algorithms are nonetheless compatible with our correctness requirement, because correctness does not consider an adversary.) Nevertheless, the equivalence holds when individual ballots are tallied correctly. Moreover, the above computation is equivalent to the following:

$(\mathfrak{v}, pf) \leftarrow \mathsf{Tally}(sk, \mathfrak{bb} \setminus \{b \mid (b, v_0, v_1) \in L\}, nc, \kappa)$;
for $(b \in \mathfrak{bb}) \wedge ((b, v_0, v_1) \in L)$ **do**
  $(\mathfrak{v}', pf') \leftarrow \mathsf{Tally}(sk, \{b\}, nc, \kappa)$;
  $\mathfrak{v} \leftarrow \mathfrak{v} + \mathfrak{v}'$;

Furthermore, by correctness of the election scheme, the above for-loop can be equivalently computed as follows:

for $(b \in \mathfrak{bb}) \wedge ((b, v_0, v_1) \in L)$ **do**
  $\mathfrak{v}[v_\beta] \leftarrow \mathfrak{v}[v_\beta] + 1$;

Indeed, for each $(b \in \mathfrak{bb}) \wedge ((b, v_0, v_1) \in L)$, we have b is an output of $\mathsf{Vote}(pk, v_\beta, nc, \kappa)$, hence, $\mathsf{Tally}(sk, \{b\}, nc, \kappa)$ outputs $(\mathfrak{v}, pf)$ such that $\mathfrak{v}$ is a zero-filled vector, except for index v_β which contains one, and this suffices to ensure equivalence. In addition, for any adversary that wins game Ballot-Secrecy, we are assured that $balanced(\mathfrak{bb}, nc, L)$ holds, hence, the above for-loop can be computed as

for $(b \in \mathfrak{bb}) \wedge ((b, v_0, v_1) \in L)$ **do**
  $\mathfrak{v}[v_0] \leftarrow \mathfrak{v}[v_0] + 1$;

or

for $(b \in \mathfrak{bb}) \wedge ((b, v_0, v_1) \in L)$ **do**
  $\mathfrak{v}[v_1] \leftarrow \mathfrak{v}[v_1] + 1$;

without weakening the game. Thus, perhaps surprisingly, tallying ballots constructed by the oracle does not provide the adversary with an advantage (in determining whether $\beta = 0$ or $\beta = 1$) and we can omit such ballots from tallying in game Ballot-Secrecy. That is, we need only consider the game derived from Ballot-Secrecy by replacing $\mathcal{A}(\mathfrak{v}, pf)$ with $\mathcal{A}(\mathfrak{v})$ and $balanced(\mathfrak{bb}, nc, L)$ with $\{b \mid (b, v_0, v_1) \in L\} \cap \mathfrak{bb} = \emptyset$, where the former modification captures our first reduction in the expense of ballot-secrecy proofs and the latter captures our second.

Smyth [96] further reduces the expense of ballot-secrecy proofs by removing the oracle in favour of a single challenge ballot:

Definition 11 (IND-CVA [96]). *Let* $\Gamma = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$ *be an election scheme,* $\mathcal{A}$ *be an adversary,* κ *be the security parameter, and* $\mathsf{IND\text{-}CVA}(\Gamma, \mathcal{A}, \kappa)$ *be the following game.*

$\mathsf{IND\text{-}CVA}(\Gamma, \mathcal{A}, \kappa) =$

1 $(pk, sk, mb, mc) \leftarrow \mathsf{Setup}(\kappa)$;
2 $(v_0, v_1, nc) \leftarrow \mathcal{A}(pk, \kappa)$;
3 $\beta \leftarrow_R \{0, 1\}$;
4 $b \leftarrow \mathsf{Vote}(pk, v_\beta, nc, \kappa)$;
5 $\mathfrak{bb} \leftarrow \mathcal{A}(b)$;
6 $(\mathfrak{v}, pf) \leftarrow \mathsf{Tally}(sk, \mathfrak{bb}, nc, \kappa)$;
7 $g \leftarrow \mathcal{A}(\mathfrak{v})$;
8 **return** $(g = \beta) \wedge (b \notin \mathfrak{bb}) \wedge (1 \leq v_0, v_1 \leq nc \leq mc\ ) \wedge (|\mathfrak{bb}| \leq mb)$;

We say election scheme Γ *satisfies* IND-CVA*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a negligible function* negl*, such that for all security parameters* κ*, we have* $\mathsf{Succ}(\mathsf{IND\text{-}CVA}(\Gamma, \mathcal{A}, \kappa)) \leq \frac{1}{2} + \mathsf{negl}(\kappa)$.

An election scheme satisfies IND-CVA when algorithm Vote outputs non-malleable ballots.

The relationship between Ballot-Secrecy and IND-CVA (also called IND-CVA, for *indistinguishability under chosen-vote attack*) is central to simplifying ballot-secrecy proofs. In Ballot-Secrecy, the adversary can query an oracle many times to obtain ballots for vote pairs of their choice, then must distinguish which votes were actually encrypted. In IND-CVA, the adversary receives only a *single* challenge ballot and must determine which of two votes it contains—but may construct additional ballots themselves and place them on the bulletin board.

Why does this simplification work? The key insight is that if ballots are non-malleable, the adversary gains nothing from oracle access beyond what they could construct themselves. Malleable ballots would allow an adversary to transform an oracle-provided ballot into a related ballot, potentially leaking information about the encrypted vote. Non-malleability blocks this attack vector.

Smyth proves that game Ballot-Secrecy is strictly stronger than game IND-CVA, moreover, he proves that the games coincide for election schemes with zero-knowledge tallying proofs that tally individual ballots correctly [96, Theorems 1 & 4]. Here strictly stronger means any scheme satisfying Ballot-Secrecy also satisfies IND-CVA, but not necessarily vice versa. However, the games coincide (yield equivalent security guarantees) when two conditions hold: (1) tallying proofs reveal nothing about individual votes (zero-knowledge), and (2) the tally correctly counts each ballot. Under these conditions, proving the simpler IND-CVA game suffices to establish Ballot-Secrecy.

Furthermore, Smyth proves that universally-verifiable election schemes tally individual ballots correctly [96, Lemmata 8 & 24].

Design guideline 4. *Ballots must be non-malleable.*

These results significantly reduce the expense of ballot-secrecy proofs and we now use them to prove Proposition 3.

Proof of Proposition 3. Suppose to the contrary that election scheme $\mathsf{Enc2Vote}(\Pi)$ does not satisfy Ballot-Secrecy. Since Ballot-Secrecy is strictly stronger than IND-CVA [96, Theorem 1], scheme $\mathsf{Enc2Vote}(\Pi)$ does not satisfy IND-CVA either, hence, there exists a probabilistic polynomial-time adversary $\mathcal{A}$ that wins $\mathsf{IND\text{-}CVA}(\mathsf{Enc2Vote}(\Pi), \mathcal{A}, \kappa)$ with success greater than negligibly better than guessing, for some security parameter κ. From $\mathcal{A}$, we construct the following adversary $\mathcal{B}$ that wins IND-PA0.

- $\mathcal{B}(pk, \mathcal{M}, \kappa)$ computes $pk' \leftarrow (pk, \mathcal{M})$; $(v_0, v_1, nc) \leftarrow \mathcal{A}(pk', \kappa)$ and outputs (v_0, v_1).
- $\mathcal{B}(b)$ computes $\mathfrak{bb} \leftarrow \mathcal{A}(b)$, parses $\mathfrak{bb}$ as a set $\{b_1, \ldots, b_{|\mathfrak{bb}|}\}$, and outputs vector $(b_1, \ldots, b_{|\mathfrak{bb}|})$.

- $\mathcal{B}(\mathbf{m})$ initialises $\mathfrak{v}$ as a zero-filled vector of length nc, computes **for** $1 \leq i \leq |\mathbf{m}|$ **do** $v \leftarrow \mathbf{m}[i]$; **if** $1 \leq v \leq nc$ **then** $\mathfrak{v}[v] \leftarrow \mathfrak{v}[v] + 1$; $g \leftarrow \mathcal{A}(\mathfrak{v})$ and outputs g.

We prove that the success of adversary $\mathcal{B}$ is equivalent to the success of adversary $\mathcal{A}$, which contradicts our assumption that Π satisfies IND-PA0.

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and $\mathsf{Enc2Vote}(\Pi) = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$. Suppose $(pk, sk, \mathcal{M})$ is an output of $\mathsf{Gen}(\kappa)$, (v_0, v_1) is an output of $\mathcal{B}(pk, \mathcal{M}, \kappa)$, and b is an output of $\mathsf{Enc}(pk, v_\beta)$, for some bit β chosen uniformly at random. By inspection of algorithms Setup and Vote, and of adversary $\mathcal{B}$, it is straightforward to see that $\mathcal{B}$ simulates the challenger in IND-CVA to adversary $\mathcal{A}$. Indeed, adversary $\mathcal{B}$ couples public key pk with message space $\mathcal{M}$, and inputs the resulting pair pk' to $\mathcal{A}$, which corresponds to the public key computed by algorithm Setup, hence, the public key input to $\mathcal{A}$ by the challenger in IND-CVA. Thus, adversary $\mathcal{A}$ behaves as if playing game IND-CVA and output (v_0, v_1) is indistinguishable from outputs that would be observed while playing that game. Moreover, since $\mathcal{A}$ wins with success greater than negligibly better than guessing, we have $1 \leq v_0, v_1 \leq nc \leq mc$, furthermore, $\{1, \ldots, nc\} \subseteq \mathcal{M}$, where mc is the largest integer such that $\{0, \ldots, mc\} \subseteq \{0\} \cup \mathcal{M}$ and for all $m_0, m_1 \in \{1, \ldots, mc\}$ we have $|m_0| = |m_1|$ (hence, $|v_0| = |v_1|$, which is required to win IND-PA0), with the same probability. It follows that outputs of $\mathsf{Enc}(pk, v_\beta)$ and $\mathsf{Vote}(pk, v_\beta, nc, \kappa)$ are indistinguishable. Suppose $(b_1, \ldots, b_\ell)$ is an output of $\mathcal{B}(b)$. It is trivial to see that $\mathcal{B}$ simulates the challenger in IND-CVA to adversary $\mathcal{A}$, because the aforementioned outputs of Enc and Vote are indistinguishable. Since $\mathcal{A}$ wins, we have $b \notin \{b_1, \ldots, b_\ell\}$, hence, $\bigwedge_{1 \leq i \leq \ell}(b \neq b_i)$, again with the same probability.

Let $\mathbf{m} = (\mathsf{Dec}(sk, b_1), \ldots, \mathsf{Dec}(sk, b_\ell))$ and suppose g is an output of $\mathcal{B}(\mathbf{m})$. By inspection of algorithm Tally and of adversary $\mathcal{B}$, we can see that $\mathcal{B}$ simulates the challenger in IND-CVA to adversary $\mathcal{A}$. Indeed, both the algorithm and adversary initialise $\mathfrak{v}$ as a zero-filled vector of length nc, then the adversary $\mathcal{B}$ computes

for $1 \leq i \leq |\mathbf{m}|$ **do** $v \leftarrow \mathbf{m}[i]$; **if** $1 \leq v \leq nc$ **then** $\mathfrak{v}[v] \leftarrow \mathfrak{v}[v] + 1$;

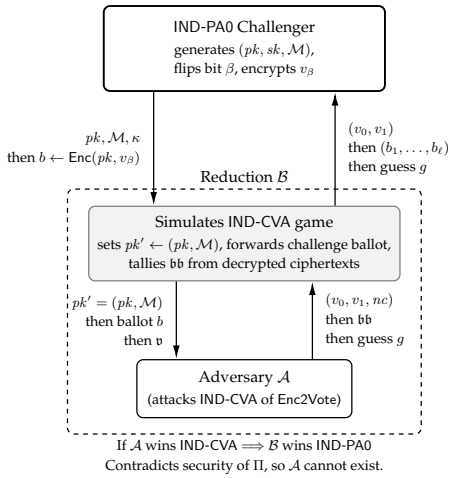
which is equivalent to algorithm Tally computing

for $b \in \{b_1, \ldots, b_\ell\}$ **do** $v \leftarrow \mathsf{Dec}(sk, b)$; **if** $1 \leq v \leq nc$ **then** $\mathfrak{v}[v] \leftarrow \mathfrak{v}[v] + 1$;

because algorithm Dec is deterministic. Thus, output g is indistinguishable from outputs that would be observed in game IND-CVA. It follows that the success of adversary $\mathcal{B}$ is equivalent to the success of $\mathcal{A}$, and we conclude our proof by [96, Theorem 4], since Smyth proves that games IND-CVA and Ballot-Secrecy coincide for election schemes with zero-knowledge tallying proofs that tally individual ballots correctly. (We omit proving that election scheme $\mathsf{Enc2Vote}(\Pi)$ has zero-knowledge tallying proofs and tallies individual ballots correctly to avoid recalling formal definitions of those properties. Proving the former is trivial, because pf is a constant, hence, it reveals nothing about the votes expressed in ballots $b_1, \ldots, b_\ell$. The latter follows from the definition of algorithm Tally, by correctness of Π, and since Π satisfies IND-PA0, which is required only to ensure ciphertexts do not collide. Formally proving these details is left as an exercise for the reader.) □

We can exploit Proposition 3 to achieve our fourth and final reduction in the expense of ballot-secrecy proofs. Indeed, if an election scheme tallies sets of ballots correctly (rather than individual ballots, as previously required), then we can compute the election outcome using function *correct-outcome* in game IND-CVA, rather than the tallying algorithm, that is, by replacing $(\mathfrak{v}, pf) \leftarrow \mathsf{Tally}(sk, \mathfrak{bb}, nc, \kappa)$ with $\mathfrak{v} \leftarrow \mathit{correct\text{-}outcome}(pk, nc, \mathfrak{bb}, \kappa)$. It follows that election scheme $(\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$ satisfies Ballot-Secrecy if and only if $(\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}', \mathsf{Verify}')$ does, assuming algorithms Tally and Tally′ both tally sets of ballots correctly. Proposition 3 proves

Figure 8 Reduction for Proposition 3. Adversary $\mathcal{B}$ receives an IND-PA0 challenge (an encryption of either v_0 or v_1) and uses it as the challenge ballot in a simulated IND-CVA game for $\mathcal{A}$. When $\mathcal{A}$ returns a bulletin board $\mathfrak{bb} = \{b_1, \ldots, b_\ell\}$, adversary $\mathcal{B}$ submits these ciphertexts to its own challenger for decryption and tallies the result. Since $\mathcal{B}$ perfectly simulates the IND-CVA game, any advantage $\mathcal{A}$ has transfers directly to $\mathcal{B}$, contradicting the IND-PA0 security of Π.



that election scheme Enc2Vote(Π) satisfies Ballot-Secrecy, assuming Π is an asymmetric encryption scheme satisfying IND-PA0, and Smyth [96, Lemma 12] proves that Enc2Vote(Π) tallies sets of ballots correctly, under the additional assumption that Π satisfies *well-definedness*, that is, ill-formed ciphertexts are distinguishable from well-formed ciphertexts. Thus, Ballot-Secrecy is satisfied by any election scheme derived from Enc2Vote(Π) by replacing its tallying and verification algorithms, assuming the replacement tallying algorithm tallies sets of ballots correctly and uses zero-knowledge tallying proofs [96, Theorem 13]. Moreover, Smyth proves that universally-verifiable election schemes tally sets of ballots correctly [96, Lemma 24]. It follows that proofs of ballot secrecy are trivial for a class of universally-verifiable, encryption-based voting systems: Any universally-verifiable election scheme derived from Enc2Vote(Π) satisfies Ballot-Secrecy if Π satisfies IND-PA0 and well-definedness, and tallying proofs are zero-knowledge.

We will use our third simplification to prove that a variant of Helios satisfies Ballot-Secrecy and the fourth to prove that a variant of Helios Mixtnet does too (the original schemes have known vulnerabilities and they do not satisfy Ballot-Secrecy), thereby demonstrating the application of these results.

10.4 Related definitions of ballot secrecy

Discussion of ballot secrecy originates from Chaum [30] and the earliest definitions of ballot secrecy are due to Benaloh *et al.* [10–12]. More recently, Bernhard *et al.* propose a series of ballot secrecy definitions [13, 14, 101, 102]. Smyth [96] shows that these definitions do not detect vulnerabilities that arise when an adversary controls the bulletin board or the communication channel. By comparison, the definition of ballot secrecy that we consider (Definition 9) detects such vulnerabilities and appears to be the strongest definition in the literature.

Beyond the computational model of security, Delaune, Kremer & Ryan formulate a definition of ballot secrecy in the applied pi calculus [43] and Smyth *et al.* show that this definition is amenable to automated reasoning [17, 18, 45, 69, 95]. An alternative definition is proposed by Cremers & Hirschi, along with sufficient conditions which are also amenable to automated rea-

soning [42]. Albeit, the scope of automated reasoning is limited by analysis tools (for example, ProVerif [19]), because the function symbols and equational theory used to model cryptographic primitives might not be suitable for automated analysis (see [5,44,84]).

10.5 Further notions of privacy

Ballot secrecy formalises a notion of free-choice assuming ballots are constructed and tallied in the prescribed manner. Moreover, Smyth's definition assumes the adversary's capabilities are limited to casting ballots on behalf of some voters and controlling the votes cast by the remaining voters. We have seen that voting system Enc2Vote satisfies this definition, but ballot secrecy does not ensure free-choice when adversaries are able to communicate with voters nor when voters deviate from the prescribed voting procedure to follow instructions provided by adversaries. Indeed, the coins used for encryption serve as proof of how a voter voted in Enc2Vote and the voter may communicate those coins to the adversary. Stronger notions of free-choice, such as receipt-freeness [25,48,68,78] and coercion resistance [49,57,62,72,110], are needed in the presence of such adversaries.

Coercion resistance, informally, is the property ensuring a voter cannot prove to a coercer how they voted, even if the coercer actively participates in the voting process. *Receipt-freeness* is a related but weaker property: a voter cannot construct a receipt proving their vote to a third party. Formalising coercion resistance has proven difficult. Haines and Smyth [57] survey four prominent definitions and find all but one to be unsuitable, demonstrating the challenges faced in capturing this property. See also Küsters, Truderung and Vogt [72] for a game-based treatment.

Ballot secrecy does not provide assurances when deviations from the prescribed tallying procedure are possible. Indeed, ballots can be tallied individually to reveal votes. Hence, the tallier must be trusted. Alternatively, we can design election schemes that distribute the tallier's role amongst several talliers and ensure free-choice assuming at least one tallier tallies ballots in the prescribed manner. Extending results in this direction is an opportunity for future work. Ultimately, we would prefer not to trust talliers. Unfortunately, this is only known to be possible for decentralised voting systems, for example [54,58,65–67,92], which are designed such that ballots cannot be tallied individually, but are unsuitable for large-scale elections.

10.5.1 Everlasting privacy

A concern with current electronic voting systems is the longevity of vote privacy. Systems like Helios and Belenios rely on computational assumptions, specifically the hardness of the discrete logarithm problem, for ballot secrecy. Advances in quantum computing or mathematical breakthroughs could eventually render today's encrypted ballots decryptable, motivating *everlasting privacy*: the guarantee that votes remain secret even against computationally unbounded adversaries [56].

Achieving everlasting privacy while maintaining verifiability is challenging, since verifiability requires publishing cryptographic evidence that could eventually be broken. Existing approaches include perfectly hiding commitments, anonymous channels that provide unconditional anonymity, and verifiable secret sharing where destroyed key shares cannot be reconstructed [46]. The trade-offs between everlasting privacy and efficiency remain an active research area.

10.5.2 Post-quantum cryptography

Quantum computers pose a threat to current electronic voting systems. Shor's algorithm [94] can efficiently solve the discrete logarithm and integer factorisation problems underlying ElGamal and RSA encryption, problems that would take classical computers thousands of years to solve. While current quantum computers can only factor small numbers, and significant engineering challenges remain, the cryptography community is actively preparing for a post-quantum future.

Several research efforts have developed post-quantum secure electronic voting based on lattice assumptions [3, 4, 31]. Recent work by Farzaliyev et al. [47] provides comprehensive lattice-based constructions for zero-knowledge proofs of ballot correctness, while Hough et al. [60] achieve significant efficiency improvements using NTRU-based constructions.

The NIST post-quantum cryptography standardisation (2022) provides a foundation for building post-quantum electronic voting systems, but the specialised primitives required, such as homomorphic encryption, verifiable shuffles, and range proofs, need additional research. Transitioning deployed systems like Helios and Belenios to post-quantum security remains an open challenge.

11 Case studies

Armed with the concepts and techniques from the previous sections, we can examine electronic election schemes that exist in the literature as case studies. Our aim is to demonstrate how the security definitions and proof techniques developed earlier apply to real voting systems, highlighting both their strengths and vulnerabilities.

Although there are various election schemes implemented by private companies, we cannot focus on these as they generally do not present their cryptographic designs in sufficient detail for rigorous analysis.

We will focus on the family of election schemes known as Helios. Helios is particularly well-suited for our case study because it is open-source, well-documented, has been subject to extensive academic scrutiny, and has been deployed in real elections including those of the International Association for Cryptologic Research (IACR).

11.1 Case study I: Helios

Helios can be informally modelled as the following election scheme (further details appear in Figure 9):

Setup generates a key pair for an asymmetric additively-homomorphic encryption scheme, proves correct key generation in zero-knowledge, and outputs the key pair and proof.

Vote enciphers the vote's bitstring encoding to a tuple of ciphertexts, proves in zero-knowledge that each ciphertext is correctly constructed and that the vote is selected from the sequence of candidates, and outputs the ciphertexts coupled with the proofs.

Tally selects ballots from the bulletin board for which proofs hold, homomorphically combines the ciphertexts in those ballots,[1] decrypts the homomorphic combination to reveal the election outcome, and announces the outcome, along with a zero-knowledge proof of correct decryption.

[1] In Section 3.5, we discuss the importance of homomorphic encryption in election schemes.

Figure 9 Helios: Ballot construction and tallying

Algorithm Vote inputs a vote v selected from candidates $1, \ldots, nc$ and computes ciphertexts $c_1, \ldots, c_{nc-1}$ such that if $v < nc$, then ciphertext c_v contains plaintext 1 and the remaining ciphertexts contain plaintext 0, otherwise, all ciphertexts contain plaintext 0. The algorithm also computes proofs $\sigma_1, \ldots, \sigma_{nc}$ demonstrating correct computation. Proof σ_j demonstrates that ciphertext c_j contains 0 or 1, where $1 \leq j \leq nc - 1$, and proof σ_{nc} demonstrates that the homomorphic combination of ciphertexts $c_1 \otimes \cdots \otimes c_{nc-1}$ contains 0 or 1. The algorithm outputs the ciphertexts and proofs.

Algorithm Tally inputs a bulletin board $\mathfrak{bb}$; selects all the ballots $b_1, \ldots, b_k \in \mathfrak{bb}$ for which proofs hold, that is, ballots $b_i = \mathsf{Enc}(pk, m_{i,1}), \ldots, \mathsf{Enc}(pk, m_{i,nc-1}), \sigma_{i,1}, \ldots, \sigma_{i,nc}$ such that proofs $\sigma_{i,1}, \ldots, \sigma_{i,nc}$ hold, where $1 \leq i \leq k$; forms a matrix of the encapsulated ciphertexts, that is,

$$\begin{array}{ccc} \mathsf{Enc}(pk, m_{1,1}), & \ldots, & \mathsf{Enc}(pk, m_{1,nc-1}) \\ \vdots & & \vdots \\ \mathsf{Enc}(pk, m_{k,1}), & \ldots, & \mathsf{Enc}(pk, m_{k,nc-1}); \end{array}$$

homomorphically combines the ciphertexts in each column to derive the encrypted outcome, that is,

$$\mathsf{Enc}(pk, \Sigma_{i=1}^{k} m_{i,1}), \quad \ldots, \quad \mathsf{Enc}(pk, \Sigma_{i=1}^{k} m_{i,nc-1});$$

decrypts the homomorphic combinations to reveal the frequency of votes $1, \ldots, nc - 1$, that is,

$$\Sigma_{i=1}^{k} m_{i,1}, \quad \ldots, \quad \Sigma_{i=1}^{k} m_{i,nc-1};$$

computes the frequency of vote nc by subtracting the frequency of any other vote from the number of ballots for which proofs hold, that is, $k - \sum_{j=1}^{nc-1} \sum_{i=1}^{k} m_{i,j}$; and announces the outcome as those frequencies, along with a proof demonstrating correctness of decryption.

Verify checks the proofs and accepts the outcome if these checks succeed.

Helios was first released in 2009 as *Helios 2.0*, and the current release is *Helios 3.1.4*. A new release was planned but never materialised.[2] Henceforth, we'll refer to the planned release as *Helios'12*.

The proofs $\sigma_1, \ldots, \sigma_{nc}$ in Figure 9 are *disjunctive proofs* (or *OR-proofs*): each proves that a ciphertext encrypts 0 *or* 1 without revealing which. This is achieved by constructing a real proof for the true case and simulating a proof for the false case—possible because the underlying proof system is zero-knowledge. The Fiat-Shamir transformation (Section 3.11) makes these proofs non-interactive, allowing them to be posted alongside ballots on the bulletin board.

11.1.1 Helios 2.0

Analysis by Cortier & Smyth [40, 41] demonstrates that Helios 2.0 fails to provide ballot secrecy. Building on the specification of Helios 2.0 developed by Smyth, Frink, and Clarkson [103], Smyth [96] establishes that Ballot-Secrecy cannot be satisfied.

[2] `http://documentation.heliosvoting.org/verification-specs/helios-v4`, published c. 2012, accessed 21 Sep 2017. The specification document remains marked "NOT FINAL" and "IN PROGRESS" as of 2025, and the planned Fall 2012 release never occurred. Development activity on the main Helios repository has been minimal since approximately 2016.

Theorem 4. *Helios 2.0 does not satisfy* Ballot-Secrecy.

Cortier & Smyth [40, 41] trace the vulnerability to Helios tallying meaningfully related ballots. Specifically, Helios ballots are malleable: starting from a ballot $c_1, \ldots, c_{nc-1}, \sigma_1, \ldots, \sigma_{nc}$, any permutation χ on $\{1, \ldots, nc-1\}$ gives $c_{\chi(1)}, \ldots, c_{\chi(nc-1)}, \sigma_{\chi(1)}, \ldots, \sigma_{\chi(nc-1)}, \sigma_{nc}$. Thus, ballots are malleable, which is incompatible with ballot secrecy (§10.3).

Proof sketch. Suppose an adversary queries the oracle with (distinct) inputs $v_0, v_1 \in \{1, \ldots, nc-1\}$ to derive a ballot for v_β, where integer $nc \geq 3$ is chosen by the adversary and bit β is chosen by the challenger. Further suppose the adversary picks a permutation χ on $\{1, \ldots, nc-1\}$, abuses malleability to derive a related ballot b for $\chi(v_\beta)$, and outputs bulletin board $\{b\}$. The board is balanced, because it does not contain the ballot output by the oracle. Suppose the adversary performs the following computation on input of election outcome $\mathfrak{v}$: if $\mathfrak{v}[\chi(v_0)] = 1$, then output 0, otherwise, output 1. Since b is a ballot for $\chi(v_\beta)$, it follows by correctness that $\mathfrak{v}[\chi(v_0)] = 1$ iff $\beta = 0$, and $\mathfrak{v}[\chi(v_1)] = 1$ iff $\beta = 1$, hence, the adversary wins the game. □

For simplicity, the proof sketch considers an adversary that omits ballots from the bulletin board. Voters might detect such an adversary, because Helios satisfies individual verifiability, hence, voters can discover if their ballot is omitted. The proof sketch can be extended to avoid such detection: Let b_1 be the ballot output by the oracle in the proof sketch and suppose b_2 is the ballot output by a (second) oracle query with inputs v_1 and v_0. Further suppose the adversary outputs (the balanced) bulletin board $\{b, b_1, b_2\}$ and performs the following computation on input of election outcome $\mathfrak{v}$: if $\mathfrak{v}$ corresponds to votes $v_0, v_1, \chi(v_0)$, then output 0, otherwise, output 1, where χ is the permutation chosen by the adversary. Hence, the adversary wins the game.

Chang-Fong & Essex show that Helios 2.0 does not satisfy universal verifiability [26, Section 4.1], and Smyth, Frink & Clarkson use their result to prove that the completeness aspect of Universal-Verifiability is not satisfied [103].[3]

Theorem 5. *Helios 2.0 does not satisfy* Completeness.

Chang-Fong & Essex attribute the vulnerability to not checking the suitability of cryptographic parameters nor checking that ballots are constructed from such parameters.

Proof sketch. Suppose an adversary computes a ciphertext and masks a term of that ciphertext. Moreover, suppose the adversary falsifies a proof of correct construction in a manner that hides malice. In particular, the adversary computes the proof such that an exponent will evaluate to zero during verification, which causes cancellation of the mask. (This is possible because verification does not check that ballots are constructed from suitable cryptographic parameters.) Suppose the adversary computes a bulletin board containing the masked ciphertext and proof. Moreover, suppose that the challenger tallies that board. The masked ciphertext will be homomorphically combined with other ciphertexts and decrypted, because the proof holds. Yet, the proof of correct decryption constructed by the challenger will fail, due to the masked ciphertext, hence, the adversary wins the game. □

The vulnerability was mitigated against in Helios 3.1.4 by performing the necessary checks.

[3] Chang-Fong & Essex present a vulnerability [26, Section 4.2] that should violate Soundness. We leave the proof of this result as an exercise for the reader.

11.1.2 Helios 3.1.4

Ballots remain malleable in Helios 3.1.4, hence, ballot secrecy is not satisfied, and Smyth [96] proves that Ballot-Secrecy is not satisfied, using the formal description of Helios 3.1.4 by Smyth, Frink & Clarkson [103].

Corollary 6. *Helios 3.1.4 does not satisfy* Ballot-Secrecy.

A proof of Corollary 6 follows from Theorem 4, because Helios 3.1.4 does not address issues arising from related ballots.

Bernhard, Pereira & Warinschi show that Helios 3.1.4 does not satisfy universal verifiability [14, Section 3], and Smyth, Frink & Clarkson use their result to prove that the soundness aspect of Universal-Verifiability is not satisfied.[4]

Theorem 7. *Helios 3.1.4 does not satisfy* Soundness.

Bernhard *et al.* attribute vulnerabilities to application of the Fiat-Shamir transformation without inclusion of statements in hashes (that is, weak Fiat-Shamir).

Proof sketch. Suppose an adversary partially computes a proof of ciphertext construction, before computing a ciphertext and without computing a key pair. In particular, suppose the adversary computes the challenge hash. (This is possible because weak Fiat-Shamir does not include statements in hashes, hence, ciphertexts are not included in hashes.) Further suppose the adversary computes a private key as a function of that hash, challenges as functions of the hash and the private key, and responses as functions of the challenges and some coins. Moreover, suppose the adversary computes a public key (from the private key) and a proof of correct key generation. That proof is valid, because the private key could have been correctly computed. Suppose the adversary encrypts some plaintext m (such that $m > 1$) to a ciphertext, using the aforementioned coins. Further suppose the adversary proves correct decryption of that ciphertext. That proof is valid, because the ciphertext is well-formed. Finally, suppose the adversary claims $(m, m-1)$ is the election outcome corresponding to the ballot containing the ciphertext and falsified proof of correct construction. The verification procedure will accept that outcome, because all proofs hold, yet the election outcome is clearly invalid, hence, the adversary wins the game. □

11.1.3 Helios'12

Helios'12 is intended to mitigate against vulnerabilities. In particular, this specification incorporates the Fiat-Shamir transformation, rather than weak Fiat-Shamir. There are plans to incorporate what is called *ballot weeding*, that is, to omit meaningfully related ballots from tallying. Smyth, Frink & Clarkson show that Helios'12 does not satisfy universal verifiability [103], and Smyth shows that ballot secrecy is not satisfied either [96].

Remark 8. *Helios'12 does not satisfy* Soundness.

Proof sketch. Suppose an adversary constructs a ballot, abuses malleability to derive a related ballot, and tallies both ballots. Ballot weeding will omit at least one of those ballots. (Helios'12 does not yet define a particular ballot weeding mechanism, hence, the precise behaviour is unknown. Nonetheless, we are assured that at least one ballot will be omitted, because the ballots are related.) Hence, tallying produces an election outcome that omits a vote, which soundness forbids, thus, the adversary wins the game. □

[4] Bernhard, Pereira & Warinschi present a vulnerability [26, p632] that should violate Completeness. Again, the proof of this result is left as another exercise for the reader.

Remark 9. *Helios'12 does not satisfy* Ballot-Secrecy.

Proof sketch. Neither ballot weeding nor the Fiat-Shamir transformation eliminate the vulnerability we identified in Helios 3.1.4, hence, we conclude by the proof sketch of Theorem 4. □

We point out that Remarks 8 & 9 are stated informally, because there is no formal description of Helios'12. Such a description can be derived as a straightforward variant of Helios 3.1.4 that uses ballot weeding and applies the Fiat-Shamir transformation (rather than the weak Fiat-Shamir transformation). But arguably these details provide little value, so we do not pursue them.

11.1.4 Helios'16

Smyth, Frink & Clarkson [103] propose Helios'16, a variant of Helios that uses the Fiat-Shamir transformation and non-malleable ballots, to overcome the aforementioned vulnerabilities. They prove that Helios'16 satisfies verifiability, and Smyth [96] proves that ballot secrecy is satisfied too.

Theorem 10. *Helios'16 satisfies both* Individual-Verifiability *and* Universal-Verifiability.

Proof sketch. Smyth *et al.* [98, 103] prove that ElGamal produces ciphertexts that do not collide for correctly generated keys. Hence, Helios'16 ballots do not collide, because they contain ElGamal ciphertexts constructed using such keys. Thus, Helios'16 satisfies Individual-Verifiability. Smyth, Frink & Clarkson also prove that Universal-Verifiability is satisfied. Their proof shows that tallying discards ill-formed ballots and that the remaining ballots all contain ciphertexts that encipher bitstring encodings of votes, hence, the homomorphic combination of those ciphertexts contain the encrypted outcome, which is decrypted to reveal the correct outcome (Soundness). Moreover, they show that such outcomes are always accepted (Completeness). □

Theorem 11. *Helios'16 satisfies* Ballot-Secrecy.

Proof sketch. Smyth proves that Helios'16 satisfies IND-CVA [96, Proposition 21] and Smyth, Frink & Clarkson prove that Universal-Verifiability is satisfied too (Theorem 10), moreover, Smyth proves that Helios'16 uses zero-knowledge tallying proofs, which suffices for Ballot-Secrecy (§10.3). □

These results (summarised in Table 1) provide strong motivation for future Helios releases being based upon Helios'16, since it is the only variant of Helios which is proven to satisfy both ballot secrecy and verifiability. (As a side remark, beyond secrecy and verifiability, eligibility, which we mentioned in Section 1, is known not to be satisfied [77, 104, 105].)

A note on real-world deployment. Despite the cryptographic vulnerabilities identified in the research literature, Helios has continued to be used for various elections. The International Association for Cryptologic Research (IACR) has used Helios annually since 2010 to elect board members. However, in 2025, a trustee lost their private key during the IACR election, making it impossible to decrypt the results. The IACR voided the election and subsequently implemented additional safeguards, including a 2-out-of-3 key threshold mechanism. This incident illustrates that even cryptographically sound systems can fail due to operational issues, highlighting the importance of key management and threshold cryptography in practice.

	Helios 2.0	Helios 3.1.4	Helios'12	Helios'16
Ballot secrecy	✗	✗	✗	✓
Individual verifiability	✓	✓	✓	✓
Universal verifiability	✗	✗	✗	✓

Cortier & Smyth identify a secrecy vulnerability in Helios 2.0 and Helios 3.1.4 [41], and Smyth shows the vulnerability is exploitable in Helios'12 when the adversary controls ballot collection [96]. Moreover, Smyth proves that Helios'16 satisfies ballot secrecy. Bernhard, Pereira & Warinschi identify universal-verifiability vulnerabilities in Helios 2.0 and Helios 3.1.4 [14], Chang-Fong & Essex identify vulnerabilities in Helios 2.0 [26], and Smyth, Frink, & Clarkson identify a vulnerability in Helios'12 [103]. Moreover, Smyth, Frink, & Clarkson prove that Helios'16 satisfies individual and universal verifiability.

Table 1: Summary of Helios security results

11.2 Case study II: Helios Mixnet

Helios Mixnet can be informally modelled as the following election scheme:

Setup generates a key pair for an asymmetric homomorphic encryption scheme, proves correct key generation in zero-knowledge, and outputs the key pair and proof.

Vote enciphers the vote to a ciphertext, proves correct ciphertext construction in zero-knowledge, and outputs the ciphertext coupled with the proof.

Tally selects ballots from the bulletin board for which proofs hold, mixes the ciphertexts in those ballots, decrypts the ciphertexts output by the mix to reveal the election outcome (that is, the frequency distribution of votes) and any ill-formed votes (that is, votes that are not selected from the sequence of candidates), and announces that outcome, along with zero-knowledge proofs demonstrating correct decryption.

Verify checks the proofs and accepts the outcome if these checks succeed.

We have seen no implementation of Helios Mixnet by either Adida [1] or Bulens, Giry & Pereira [23]. Tsoukalas *et al.* [107] released *Zeus* as an independent variant (or fork) of Helios spliced with mixnet code to derive such an implementation. Building upon this, Yingtong Li released *helios-server-mixnet* as an extension of Zeus with threshold asymmetric encryption and some other minor changes.

As noted before in Theorem 5, Helios 2.0 does not satisfy completeness, which means that any implementations of Helios Mixnet did not satisfy completeness until Helios was patched because such implementations build off the Helios code and do not add code to check cryptographic parameters. In addition to the lack of completeness, Smyth [98] identified a soundness vulnerability in Helios Mixnet.

Remark 12. *Zeus does not satisfy* Soundness.

Smyth traces the vulnerability to the weak Fiat-Shamir transformation. This vulnerability was reported to the developers of Zeus and helios-server-mixnet in 2018, who promptly adopted and deployed the proposed fix [98, Section 4].

Proof sketch. We use the term subdistribution to refer to an incomplete frequency distribution of votes, meaning one that accounts for only a subset of the actual votes cast. For the proof sketch,

suppose an adversary constructs some ballots and mixes the ciphertexts in those ballots. Further suppose the adversary decrypts the ciphertexts output by the mix to reveal the frequency distribution of votes and then selects some ciphertexts that decrypt to a (strict) subdistribution. The adversary can then prove correct decryption of those ciphertexts and falsify proofs of the remaining ciphertexts decrypting to arbitrary elements of the message space (by exploiting a vulnerability against Helios [14] due to the weak Fiat-Shamir transformation). Finally, suppose the adversary claims the subdistribution of votes is the election outcome. The verification procedure will accept that outcome, because all proofs hold, yet the election outcome excludes votes, hence, the adversary wins the game. □

Similarly, voting system helios-server-mixnet does not satisfy Soundness when a (n, n)-threshold is used [98].

Smyth proposes a formal description of Helios Mixnet that uses the Fiat-Shamir transformation and proves that Ballot-Secrecy, Individual-Verifiability, and Universal-Verifiability are satisfied [96, 99].

Theorem 13. *Helios Mixnet satisfies both* Individual-Verifiability *and* Universal-Verifiability.

Proof sketch. In the Helios system, ballots do not collide 10, because they contain ElGamal ciphertexts constructed using correctly generated keys, which means that Individual-Verifiability is satisfied. This also means Universal-Verifiability is satisfied, because tallying discards ill-formed ballots and votes, hence the mix gives the correct outcome (corresponding to Soundness), and such outcomes are always accepted (corresponding to Completeness).

□

Theorem 14. *Helios Mixnet satisfies* Ballot-Secrecy.

Proof sketch. Smyth [98] demonstrates that Helios Mixnet can be derived from Enc2Vote(Π) by using suitable tallying and verification algorithms. Smyth also proves that Universal-Verifiability is satisfied, which is enough for Ballot-Secrecy (§10.3), assuming Π satisfies IND-PA0 and well-definedness. □

11.3 Case study III: Belenios

Belenios is a secret verifiable electronic voting system developed by Cortier, Gaudry, and Glondu [35], building upon the Helios codebase but addressing several of its security limitations. The system has been deployed in hundreds of real-world elections since its creation, including academic institutions, associations, and notably in experiments during French legislative elections [36].

11.3.1 Key differences from Helios

The most significant change in Belenios with respect to Helios is the addition of digital signatures to attest that ballots come from eligible voters associated with a given credential [6]. This addresses a fundamental limitation of Helios: vulnerability to ballot stuffing by a malicious bulletin board.

In Helios, a dishonest bulletin board could add ballots without anyone noticing, because there is no mechanism to verify that ballots originated from legitimate voters. Belenios provides unforgeability: nobody can forge a fake signature.

11.3.2 Limitations and extensions

Like Helios, Belenios is not coercion-resistant: voters may prove how they voted by revealing the randomness used to produce their ballot, or they may sell their voting credentials.

Belenios does not ensure *participation privacy*: the publicly available election data reveals whether a particular voter participated in the election. Although this information is typically available in traditional paper-based elections (anyone can observe people entering a polling station), an Internet voting system without participation privacy reveals voter identities on a much larger scale by publishing them online [71].

Several extensions to Belenios have been proposed. *BeleniosVS* [33] provides secrecy and verifiability even against a corrupted voting device. Recent work has added *cast-as-intended* verification mechanisms [37], allowing voters to verify that their device correctly encoded their intended vote. In 2024, Belenios underwent a certification campaign in France [20], providing additional assurance for its use in official elections.

11.3.3 Comparison with Helios

Table 2 summarises the key differences between Helios and Belenios.

	Helios	Belenios
Ballot secrecy	✗*	✓
Individual verifiability	✓	✓
Universal verifiability	✗*	✓
Eligibility verifiability	✗	✓
Resistance to ballot stuffing	✗	✓
Participation privacy	✗	✗
Receipt-freeness	✗	✗
Coercion resistance	✗	✗

*As discussed in Section 11.1, existing Helios implementations (2.0, 3.1.4) do not satisfy these properties, though the proposed Helios'16 variant does.

Table 2: Comparison of Helios and Belenios security properties

The primary advantage of Belenios over Helios is eligibility verifiability and resistance to ballot stuffing attacks. Both systems share similar limitations regarding coercion resistance and participation privacy, which remain active areas of research.

12 Summary

We have introduced and detailed concepts from cryptography that can be used in designing election schemes. We have given a brief introduction to game-based cryptography, which we used to formulate elections as games. Our emphasis has been on studying definitions of secrecy and verifiability, which we formally defined with cryptography games.

For definition and illustration purposes, we have examined a proposed election scheme based on four building blocks, namely the primitives Setup, Vote, Tally, and Verify, as illustrated in Figure 1 and, more precisely, in Figure 5. We have detailed these primitives or algorithms in Definition 2, which we now briefly recall. Setup randomly generates a public and private key pair, where the voters use the public key and the tallier uses the private key. Vote allows the voters to vote

using the public key. Tally adds up the encrypted votes, which is possible with a homomorphic asymmetric encryption scheme. Verify uses the public key or keys from the primitive Setup and a proof generated by the primitive Tally to verify the election outcome is correct.

We have then stated what we mean exactly by secrecy and verifiability by introducing further concepts.

We have defined election correctness as requiring that honestly executed elections produce outcomes that match the cast votes with overwhelming probability, ensuring the scheme functions as intended when all parties follow the protocol correctly.

We have defined soundness as requiring the Verify algorithm to only accept correct outcomes; Definition 5. To complement soundness, we have defined completeness as requiring election outcomes (that are produced by algorithm Tally) to be accepted by algorithm Verify; Definition 6. We have then defined universal verifiability as a combination of soundness and completeness; Definition 7.

Using the aforementioned concepts, we have then examined the voting schemes Helios, Helios Mixnet, and Belenios through detailed case studies.

12.1 Other voting systems

Beyond the Helios family, several other electronic voting systems have received formal security analysis.

JCJ [62] introduced the first formal treatment of coercion-resistant electronic voting, allowing voters to produce fake credentials that are indistinguishable from real ones, thereby enabling them to deceive coercers. *Civitas* [32] implements the JCJ protocol with practical improvements, and its source code is publicly available. However, the JCJ/Civitas approach has quadratic complexity in the number of submitted ballots during the tallying phase, limiting its applicability to large-scale elections. Moreover, the original JCJ definition of coercion resistance appears to require a patch to be satisfiable [57], and recent work by Cortier et al. [38] has identified potential weaknesses in the ballot cleansing procedure that may leak information to coercers.

Athena [100] advances the JCJ paradigm by delivering a verifiable, coercion-resistant voting system with linear complexity, addressing the scalability limitations of JCJ/Civitas. The scheme preserves the fake credential approach that underpins coercion resistance while restructuring the cryptographic operations to avoid the quadratic cost of credential validation. This makes coercion-resistant remote voting more practical for elections with large numbers of voters.

Prêt à Voter [29] uses paper-based cryptographic mechanisms with mix-network tallying and has been studied extensively for ballot secrecy and verifiability. *Scantegrity* [27, 28] provides end-to-end verifiability for optical scan systems using invisible ink confirmation codes. We do not provide the same depth of analysis for these systems, but refer interested readers to the cited works and the systematization by Cortier et al. [34], which surveys verifiability notions across multiple voting protocols.

12.2 Related tutorials and surveys

Several other works provide introductions to electronic voting security. Bernhard et al. [15] give a gentle introduction to cryptographic voting aimed at non-specialists. The systematizations of knowledge by Bernhard et al. [13] on ballot privacy definitions and Cortier et al. [34] on verifiability notions provide comprehensive treatments of the definitional landscape. For readers interested in the formal methods approach to analyzing voting protocols, Delaune et al. [43] develop techniques based on the applied pi calculus.

13 Conclusion

To better understand electronic voting, we have used a game-based approach coupled with formal definitions, such as soundness and completeness, that shows how third parties can use these tools to detect subtle vulnerabilities in voting systems. We have seen how analysis drives development and ultimately leads to systems that are provably secure. This clearly demonstrates the need for security definitions coupled with analysis to ensure security of voting systems.

For such a dynamic topic as electronic voting, it is impossible to give an up-to-date and complete overview, but this tutorial offers a good starting point for interested students, researchers, practitioners, and readers. We hope this tutorial advances the reader's understanding of these topics and themes. Overall, we want to inspire more research and work in electronic voting, with the ultimate aim of enabling democratic institutions with secure voting schemes.

A List of Symbols

The following tables collect the notation used throughout this tutorial, grouped by theme.

Cryptographic primitives

Π	Encryption scheme, consisting of the tuple $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.
Gen	Key generation algorithm for an encryption scheme.
Enc	Encryption algorithm.
Dec	Decryption algorithm.
pk	Public key for an asymmetric encryption scheme.
sk	Private (secret) key for an asymmetric encryption scheme.
$\mathcal{M}$	Message space (space of plaintexts) for an encryption scheme.

Election scheme

Γ	Election scheme, a tuple $(\mathsf{Setup}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{Verify})$.
Setup	Setup algorithm that generates keys and election parameters.
Vote	Vote algorithm that constructs a ballot from a vote.
Tally	Tally algorithm that computes the election outcome.
Verify	Verify algorithm that audits the election outcome.
$\mathsf{Enc2Vote}$	Construction that builds an election scheme from an encryption scheme.

Election parameters

κ	Security parameter.
nc	Number of candidates in an election.
mc	Maximum number of candidates.
nb	Number of ballots in an election.
mb	Maximum number of ballots.

Votes and ballots

v	A vote (selected from candidates $1, \ldots, nc$).
b	A ballot (encrypted vote).
$\mathfrak{bb}$	Bulletin board (collection of ballots).
$\mathfrak{v}$	Election outcome (vote tallies for each candidate).
pf	Tallying proof demonstrating correctness of the outcome.
s	Audit outcome.
$correct\text{-}outcome$	Function mapping a bulletin board to the vector of vote counts per candidate.
$balanced$	Predicate that holds when vote vectors are balanced (equal totals).

Assignments and general notation

$\leftarrow$	Deterministic assignment.
$\leftarrow_R$	Uniformly random sampling from a finite set.
$\mathcal{A}$	Adversary in a security game.
$\mathcal{B}$	Second adversary or reduction algorithm in a security proof.
$\mathcal{O}$	Oracle in a security game.
$\mathsf{Succ}(\cdot)$	Success probability of a game (probability of outputting $\top$).
negl	Negligible function.

Security properties

IND-CPA	IND-CPA: indistinguishability under chosen-plaintext attack.
IND-PA0	IND-PA0: indistinguishability under plaintext-awareness.
CNM-CPA	CNM-CPA: ciphertext non-malleability under chosen-plaintext attack.
Ballot-Secrecy	Ballot secrecy game.
IND-CVA	Ballot independence game (IND-CVA).
Individual-Verifiability	Individual verifiability.
Universal-Verifiability	Universal verifiability.
Completeness	Completeness (honest outcomes pass verification).
Soundness	Soundness (verification rejects incorrect outcomes).

B Glossary of Voting Terms

The following table defines voting-specific terminology used throughout this tutorial.

Core verifiability concepts

Verifiability	The ability to prove that no undue influence has occurred in an election and that the outcome correctly reflects the votes cast.
Individual verifiability [103]	A voter can check whether their ballot is collected and included in the election record. Formalized as the requirement that ballots do not collide.
Universal verifiability [103]	Anyone can check whether an election outcome corresponds to the votes expressed in collected ballots. Formalized as the combination of soundness and completeness.

Components of universal verifiability

Soundness [103]	The verification algorithm only accepts outcomes that correspond to votes expressed in collected ballots. An adversary cannot produce a proof for an incorrect outcome that passes verification.
Completeness [103]	The verification algorithm accepts outcomes computed honestly by the tallying algorithm. Ensures that correctly computed election outcomes are not rejected.
Injectivity [103]	Ballots for distinct votes never collide. Ensures that each ballot can be interpreted as encoding at most one vote.

Core security properties

Ballot secrecy [96]	A voter's vote is not revealed to anyone, including election organizers. Formalized as the inability to distinguish between an instance of the voting system in which voters cast some votes from another instance in which the voters cast a permutation of those votes.

Fine-grained verifiability

End-to-end verifiability	The combination of cast-as-intended, stored-as-cast, and tallied-as-stored verifiability. Provides comprehensive verification throughout the voting process.
Cast-as-intended [34]	A voter can verify that their voting device correctly encoded their intended vote into a ballot.
Stored-as-cast [34]	A voter can verify that their ballot was correctly stored on the bulletin board as they cast it.
Tallied-as-stored [34]	Anyone can verify that the ballots stored on the bulletin board were correctly tallied to produce the election outcome.

Related properties

Unforgeability	Only authorized voters can construct valid ballots. Prevents adversaries from injecting fraudulent ballots into the election.
Eligibility verifiability [103]	Deprecated term, used as a synonym for unforgeability in earlier work. Current terminology prefers unforgeability.
Ballot independence	Ballots constructed by honest voters do not leak information about votes cast by other honest voters. Related to the IND-CVA security game.
Final-agreement [59]	A property ensuring that all participants eventually agree on the final state of the bulletin board. Necessary for verifiability with non-idealized bulletin boards.

C Asymmetric encryption

We briefly described public-key or asymmetric encryption in Section 3.2. Now we give a more precise definition, which can be found in cryptography textbooks, such as Katz and Lindell [63].

Definition 12 (Asymmetric encryption scheme [63, 103])**.** *An* asymmetric encryption scheme *is a tuple of probabilistic polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$*, such that:*

- **Gen**, *denoted* $(pk, sk, \mathcal{M}) \leftarrow \mathsf{Gen}(\kappa)$*, inputs a security parameter* κ *and outputs a key pair* (pk, sk) *and message space* $\mathcal{M}$*.*
- **Enc**, *denoted* $c \leftarrow \mathsf{Enc}(pk, m)$*, inputs a public key* pk *and message* $m \in \mathcal{M}$*, and outputs a ciphertext* c*.*
- **Dec**, *denoted* $m \leftarrow \mathsf{Dec}(sk, c)$*, inputs a private key* sk *and ciphertext* c*, and outputs a message* m *or an error symbol. We assume* Dec *is deterministic.*

Moreover, the scheme must be correct*: there exists a negligible function* negl*, such that for all security parameters* κ *and messages* m*, we have*

$$\Pr[(pk, sk, \mathcal{M}) \leftarrow \mathsf{Gen}(\kappa); c \leftarrow \mathsf{Enc}(pk, m) : m \in \mathcal{M} \Rightarrow \mathsf{Dec}(sk, c) = m] > 1 - \mathsf{negl}(\kappa)\,.$$

D Definition of IND-PA0

In Section 4.1 we quickly discussed the importance of indistinguishability, which we defined with a game. Now we give a definition of *indistinguishability under parallel attack*, as defined by Bellare & Sahai [9], which we used in Section 10.2.

Definition 13 (IND-PA0 [9])**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be an asymmetric encryption scheme,* $\mathcal{A}$ *be an adversary,* κ *be the security parameter, and* $\mathsf{IND\text{-}PA0}(\Pi, \mathcal{A}, \kappa)$ *be the following game.*

$\mathsf{IND\text{-}PA0}(\Pi, \mathcal{A}, \kappa) =$

1 $(pk, sk, \mathcal{M}) \leftarrow \mathsf{Gen}(\kappa)$;
2 $(m_0, m_1) \leftarrow \mathcal{A}(pk, \mathcal{M}, \kappa)$;
3 $\beta \leftarrow_R \{0, 1\}$;
4 $c \leftarrow \mathsf{Enc}(pk, m_\beta)$;
5 $\mathbf{c} \leftarrow \mathcal{A}(c)$;
6 $\mathbf{m} \leftarrow (\mathsf{Dec}(sk, \mathbf{c}[1]), \ldots, \mathsf{Dec}(sk, \mathbf{c}[|\mathbf{c}|])$;
7 $g \leftarrow \mathcal{A}(\mathbf{m})$;
8 **return** $(g = \beta\ ) \wedge \bigwedge_{1 \le i \le |\mathbf{c}|} (c \neq \mathbf{c}[i])$;

In the above game, we require the plaintexts $m_0, m_1 \in \mathcal{M}$ *and* $|m_0| = |m_1|$*. We say that the encryption scheme* Π *satisfies indistinguishability under parallel attack* IND-PA0*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a negligible function* negl*, such that for all security parameters* κ*, the inequality*

$$\mathsf{Succ}(\mathsf{IND\text{-}PA0}(\Pi, \mathcal{A}, \kappa)) \le \frac{1}{2} + \mathsf{negl}(\kappa)\,,$$

holds.

References

[1] Ben Adida. Helios: Web-based Open-Audit Voting. In *USENIX Security'08: 17th USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.

[2] R. Michael Alvarez and Thad E. Hall. *Electronic Elections: The Perils and Promises of Digital Democracy*. Princeton University Press, 2010.

[3] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, and Tjerand Silde. Verifiable mix-nets and distributed decryption for voting from lattice-based assumptions. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1–15. ACM, 2023.

[4] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, Tjerand Silde, and Thor Tunge. Lattice-based proof of shuffle and applications to electronic voting. In *Topics in Cryptology – CT-RSA 2021*, volume 12704 of *LNCS*, pages 227–251. Springer, 2021.

[5] Myrto Arapinis, Sergiu Bursuc, and Mark Ryan. Reduction of Equational Theories for Verification of Trace Equivalence: Re-encryption, Associativity and Commutativity. In *POST'12: First Conference on Principles of Security and Trust*, volume 7215 of *LNCS*, pages 169–188. Springer, 2012.

[6] Sevdenur Baloglu, Sergiu Bursuc, Sjouke Mauw, and Jun Pang. Election verifiability revisited: Automated security proofs and attacks on helios and belenios. In *34th IEEE Computer Security Foundations Symposium (CSF)*, pages 1–16. IEEE, 2021.

[7] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *CRYPTO'98: 18th International Cryptology Conference*, volume 1462 of *LNCS*, pages 26–45. Springer, 1998.

[8] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS'93: 1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[9] Mihir Bellare and Amit Sahai. Non-malleable Encryption: Equivalence between Two Notions, and an Indistinguishability-Based Characterization. In *CRYPTO'99: 19th International Cryptology Conference*, volume 1666 of *LNCS*, pages 519–536. Springer, 1999.

[10] Josh Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Department of Computer Science, Yale University, 1996.

[11] Josh Benaloh and Moti Yung. Distributing the Power of a Government to Enhance the Privacy of Voters. In *PODC'86: 5th Principles of Distributed Computing Symposium*, pages 52–62. ACM Press, 1986.

[12] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections. In *STOC'94: 26th Theory of computing Symposium*, pages 544–553. ACM Press, 1994.

[13] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *S&P'15: 36th Security and Privacy Symposium*, pages 499–516. IEEE Computer Society, 2015.

[14] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In *ASIACRYPT'12: 18th International Conference on the Theory and Application of Cryptology and Information Security*, volume 7658 of *LNCS*, pages 626–643. Springer, 2012.

[15] David Bernhard and Bogdan Warinschi. Cryptographic Voting — A Gentle Introduction. In *Foundations of Security Analysis and Design VII*, volume 8604 of *LNCS*, pages 167–211. Springer, 2014.

[16] Eric C. Bjornlund. *Beyond Free and Fair: Monitoring Elections and Building Democracy*. Woodrow Wilson Center Press / Johns Hopkins University Press, 2004.

[17] Bruno Blanchet and Ben Smyth. Automated reasoning for equivalences in the applied pi calculus with barriers. In *CSF'16: 29th Computer Security Foundations Symposium*, pages 310–324. IEEE Computer Society, 2016.

[18] Bruno Blanchet and Ben Smyth. Automated reasoning for equivalences in the applied pi calculus with barriers. *Journal of Computer Security*, 26(3):367–422, 2018.

[19] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. *ProVerif 1.96: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, 2016.

[20] Angèle Bossuat, Eloïse Brocas, Véronique Cortier, Pierrick Gaudry, Stéphane Glondu, and Nicolas Kovacs. Belenios: the certification campaign. In *SSTIC 2024 – Symposium sur la sécurité des technologies de l'information et des communications*, 2024.

[21] Debra Bowen. Secretary of State Debra Bowen Moves to Strengthen Voter Confidence in Election Security Following Top-to-Bottom Review of Voting Systems. California Secretary of State, press release DB07:042, August 2007.

[22] Peter Brent. The Australian ballot: Not the secret ballot. *Australian Journal of Political Science*, 41(1):39–50, 2006.

[23] Philippe Bulens, Damien Giry, and Olivier Pereira. Running Mixnet-Based Elections with Helios. In *EVT/WOTE'11: Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*. USENIX Association, 2011.

[24] Bundesverfassungsgericht (Germany's Federal Constitutional Court). *Use of voting computers in 2005 Bundestag election unconstitutional*, March 2009. Press release 19/2009.

[25] Pyrros Chaidos, Véronique Cortier, Georg Fuschbauer, and David Galindo. BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme. In *CCS'16: 23rd ACM Conference on Computer and Communications Security*, pages 1614–1625. ACM Press, 2016.

[26] Nicholas Chang-Fong and Aleksander Essex. The Cloudier Side of Cryptographic End-to-end Verifiable Voting: A Security Analysis of Helios. In *ACSAC'16: 32nd Annual Conference on Computer Security Applications*, pages 324–335. ACM Press, 2016.

[27] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity II: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *EVT'08: Electronic Voting Technology Workshop*. USENIX Association, 2008.

[28] David Chaum, Aleks Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan Sherman, and Poorvi Vora. Scantegrity: End-to-End Voter-Verifiable Optical-Scan Voting. *IEEE Security and Privacy*, 6(3):40–46, 2008.

[29] David Chaum, Peter Y. A. Ryan, and Steve Schneider. A Practical Voter-Verifiable Election Scheme. In *ESORICS'05: 10th European Symposium On Research In Computer Security*, volume 3679 of *LNCS*, pages 118–139. Springer, 2005.

[30] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24:84–90, 1981.

[31] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A homomorphic LWE based E-voting scheme. In *Post-Quantum Cryptography (PQCrypto)*, volume 9606 of *LNCS*, pages 245–265. Springer, 2016.

[32] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a Secure Voting System. In *S&P'08: 29th Security and Privacy Symposium*, pages 354–368. IEEE Computer Society, 2008.

[33] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. BeleniosVS: Secrecy and verifiability against a corrupted voting device. In *32nd IEEE Computer Security Foundations Symposium (CSF)*, pages 367–381. IEEE, 2019.

[34] Veronique Cortier, David Galindo, Ralf Küsters, Johannes Mueller, and Tomasz Truderung. SoK: Verifiability Notions for E-Voting Protocols. In *S&P'16: 37th IEEE Symposium on Security and Privacy*, pages 779–798. IEEE Computer Society, 2016.

[35] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. Belenios: A simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning*, volume 11565 of *LNCS*, pages 214–238. Springer, 2019.

[36] Véronique Cortier, Pierrick Gaudry, Stéphane Glondu, and Sylvain Ruhault. French 2022 legislatives elections: A verifiability experiment. In *E-Vote-ID 2023*, volume 14230 of *LNCS*, pages 1–16. Springer, 2023.

[37] Véronique Cortier, Pierrick Gaudry, Anselme Goetschmann, and Sophie Lemonnier. Belenios with cast-as-intended: Towards a usable interface. In *Electronic Voting (E-Vote-ID 2024)*, volume 15014 of *LNCS*, pages 1–17. Springer, 2024.

[38] Véronique Cortier, Pierrick Gaudry, and Quentin Yang. Is the JCJ voting system really coercion-resistant? In *CSF'24: 37th IEEE Computer Security Foundations Symposium*, pages 365–380. IEEE Computer Society, 2024. ePrint 2022/430.

[39] Véronique Cortier, Benedikt Schmidt, Constantin Cătălin Drăgan, Pierre-Yves Strub, Francois Dupressoir, and Bogdan Warinschi. Machine-Checked Proofs of Privacy for Electronic Voting Protocols. In *S&P'17: 37th IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2017.

[40] Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. In *CSF'11: 24th Computer Security Foundations Symposium*, pages 297–311. IEEE Computer Society, 2011.

[41] Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.

[42] Cas Cremers and Lucca Hirschi. Improving Automated Symbolic Analysis for E-voting Protocols: A Method Based on Sufficient Conditions for Ballot Secrecy. arXiv, Report 1709.00194, 2017.

[43] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, July 2009.

[44] Stéphanie Delaune, Steve Kremer, Mark D. Ryan, and Graham Steel. Formal analysis of protocols based on TPM state registers. In *CSF'11: 24th Computer Security Foundations Symposium*, pages 66–80. IEEE Computer Society, 2011.

[45] Stéphanie Delaune, Mark D. Ryan, and Ben Smyth. Automatic verification of privacy properties in the applied pi-calculus. In *IFIPTM'08: 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security*, volume 263 of *International Federation for Information Processing (IFIP)*, pages 263–278. Springer, 2008.

[46] Denise Demirel, Jeroen van de Graaf, and Roberto Araújo. Improving Helios with everlasting privacy towards the public. In *EVT/WOTE'12: Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*. USENIX Association, 2012.

[47] Valeh Farzaliyev, Calvin Pärn, Heleen Saarse, and Jan Willemson. Lattice-based zero-knowledge proofs in action: Applications to electronic voting. *Journal of Cryptology*, 38(1):6, 2025.

[48] Ashley Fraser, Elizabeth A. Quaglia, and Ben Smyth. A critique of game-based definitions of receipt-freeness for voting. In *ProveSec'19: 13th International Conference on Provable anc Practical Security*, LNCS. Springer, 2019.

[49] Ryan W. Gardner, Sujata Garera, and Aviel D. Rubin. Coercion Resistant End-to-end Voting. In *FC'09: 13th International Conference on Financial Cryptography and Data Security*, volume 5628 of *LNCS*, pages 344–361. Springer, 2009.

[50] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[51] Craig Gentry. Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53(3):97–105, 2010.

[52] Oded Goldreich. *Foundations of cryptography: Basic Techniques*. Cambridge University Press, 2003.

[53] Rop Gonggrijp and Willem-Jan Hengeveld. Studying the Nedap/Groenendaal ES3B Voting Computer: A Computer Security Perspective. In *EVT'07: Electronic Voting Technology Workshop*. USENIX Association, 2007.

[54] Jens Groth. Efficient maximal privacy in boardroom voting and anonymous broadcast. In *FC'04: 8th International Conference on Financial Cryptography*, volume 3110 of *LNCS*, pages 90–104. Springer, 2004.

[55] Andrew Gumbel. *Steal This Vote: Dirty Elections and the Rotten History of Democracy in America*. Nation Books, 2005.

[56] Thomas Haines, Rafieh Mosaheb, Johannes Müller, and Ivan Pryvalov. SoK: Secure e-voting with everlasting privacy. *Proceedings on Privacy Enhancing Technologies*, 2023(1):279–293, 2023.

[57] Thomas Haines and Ben Smyth. SoK: Surveying definitions of coercion resistance. Cryptology ePrint Archive, Report 2019/822, 2019.

[58] Fao Hao, Peter Y. A. Ryan, and Piotr Zieliński. Anonymous voting by two-round public discussion. *Journal of Information Security*, 4(2):62 – 67, 2010.

[59] Lucca Hirschi, Lara Schmid, and David Basin. Fixing the Achilles heel of E-Voting: The bulletin board. In *34th IEEE Computer Security Foundations Symposium (CSF)*, pages 1–17. IEEE, 2021.

[60] Patrick Hough, Caroline Sandsbråten, and Tjerand Silde. More efficient lattice-based electronic voting from NTRU. Cryptology ePrint Archive, Paper 2023/933, 2023. `https://eprint.iacr.org/2023/933`.

[61] Douglas W. Jones and Barbara Simons. *Broken Ballots: Will Your Vote Count?*, volume 204 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford University, 2012.

[62] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-Resistant Electronic Elections. In *WPES'05: 4th Workshop on Privacy in the Electronic Society*, pages 61–70. ACM Press, 2005.

[63] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Chapman & Hall/CRC, 2014.

[64] Judith G. Kelley. *Monitoring Democracy: When International Election Observation Works, and Why It Often Fails*. Princeton University Press, 2012.

[65] Dalia Khader, Ben Smyth, Peter Y. A. Ryan, and Feng Hao. A Fair and Robust Voting System by Broadcast. In *EVOTE'12: 5th International Conference on Electronic Voting*, volume 205 of *Lecture Notes in Informatics*, pages 285–299. Gesellschaft für Informatik, 2012.

[66] Shahram Khazaei and Mehri Rezaei-Aliabadi. A rigorous security analysis of a decentralized electronic voting protocol in the universal composability framework. *Journal of Information Security and Applications*, 43:99–109, 2018.

[67] Aggelos Kiayias and Moti Yung. Self-tallying elections and perfect ballot secrecy. In *PKC'01: 3rd International Workshop on Practice and Theory in Public Key Cryptography*, volume 2274 of *LNCS*, pages 141–158. Springer, 2002.

[68] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-end verifiable elections in the standard model. In *EUROCRYPT'15: 34th International Conference on the Theory and Applications of Cryptographic Techniques*, volume 9057 of *LNCS*, pages 468–498. Springer, 2015.

[69] Petr Klus, Ben Smyth, and Mark D. Ryan. ProSwapper: Improved equivalence verifier for ProVerif. `http://www.bensmyth.com/proswapper.php`, 2010.

[70] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an Electronic Voting System. In *S&P'04: 25th Security and Privacy Symposium*, pages 27–40. IEEE Computer Society, 2004.

[71] Oksana Kulyk, Vanessa Teague, and Melanie Volkamer. Extending Helios towards private eligibility verifiability. In *E-Voting and Identity (VoteID 2015)*, volume 9269 of *LNCS*, pages 57–73. Springer, 2015.

[72] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. A Game-Based Definition of Coercion-Resistance and its Applications. *Journal of Computer Security*, 20(6):709–764, 2012.

[73] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash Attacks on the Verifiability of E-Voting Systems. In *S&P'12: 33rd IEEE Symposium on Security and Privacy*, pages 395–409. IEEE Computer Society, 2012.

[74] Arend Lijphart and Bernard Grofman. *Choosing an electoral system: Issues and Alternatives*. Praeger, 1984.

[75] Yehuda Lindell. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*. Springer, 2017.

[76] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[77] Maxime Meyer and Ben Smyth. Exploiting re-voting in the helios election system. *Information Processing Letters*, 2019.

[78] Tal Moran and Moni Naor. Receipt-Free Universally-Verifiable Voting with Everlasting Privacy. In *CRYPTO'06: 26th International Cryptology Conference*, volume 4117 of *LNCS*, pages 373–392. Springer, 2006.

[79] C. Andrew Neff. Practical high certainty intent verification for encrypted votes. Technical report, VoteHere, 2004.

[80] Pippa Norris. *Why Elections Fail*. Cambridge University Press, 2015.

[81] Organization for Security and Co-operation in Europe. *Document of the Copenhagen Meeting of the Conference on the Human Dimension of the CSCE*, 1990.

[82] Organization of American States. *American Convention on Human Rights, "Pact of San Jose, Costa Rica"*, 1969.

[83] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[84] Miriam Paiola and Bruno Blanchet. Verification of Security Protocols with Lists: From Length One to Unbounded Length. In *POST'12: First Conference on Principles of Security and Trust*, volume 7215 of *LNCS*, pages 69–88. Springer, 2012.

[85] Sunoo Park and Ronald L Rivest. Towards secure quadratic voting. *Public Choice*, 172(1-2):151–175, 2017.

[86] Sunoo Park, Michael Specter, Neha Narula, and Ronald L. Rivest. Going from bad to worse: from Internet voting to blockchain voting. *Journal of Cybersecurity*, 7(1):tyaa025, 2021.

[87] Elizabeth A Quaglia and Ben Smyth. Secret, verifiable auctions from elections. *Theoretical Computer Science*, 730:44–92, 2018.

[88] Ronald L. Rivest. Remarks at the Cryptographers' Panel. RSA Conference, San Francisco, February 2020. Moderated by Zulfikar Ramzan.

[89] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[90] Ronald L. Rivest and Warren D. Smith. Three Voting Protocols: ThreeBallot, VAV, and Twin. In *EVT'07: Electronic Voting Technology Workshop*. USENIX Association, 2007.

[91] Thomas Saalfeld. On Dogs and Whips: Recorded Votes. In Herbert Döring, editor, *Parliaments and Majority Rule in Western Europe*, chapter 16. St. Martin's Press, 1995.

[92] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *CRYPTO'99: 19th International Cryptology Conference*, volume 1666 of *LNCS*, pages 148–164. Springer, 1999.

[93] Nicole Schweikardt. Arithmetic, first-order logic, and counting quantifiers. *ACM Transactions on Computational Logic*, 6(3):634–671, July 2005.

[94] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[95] Ben Smyth. *Formal verification of cryptographic protocols with automated reasoning*. PhD thesis, School of Computer Science, University of Birmingham, 2011.

[96] Ben Smyth. Ballot secrecy: Security definition, sufficient conditions, and analysis of Helios. Cryptology ePrint Archive, Report 2015/942, 2018.

[97] Ben Smyth. First-past-the-post suffices for ranked voting. `https://bensmyth.com/publications/2017-FPTP-suffices-for-ranked-voting/`, 2018.

[98] Ben Smyth. Verifiability of Helios Mixnet. In *Voting'18: 3rd Workshop on Advances in Secure Electronic Voting*, LNCS. Springer, 2018.

[99] Ben Smyth. Verifiability of Helios Mixnet. Cryptology ePrint Archive, Report 2018/017, 2018.

[100] Ben Smyth. Athena: A verifiable, coercion-resistant voting system with linear complexity. Cryptology ePrint Archive, Paper 2019/761, 2019.

[101] Ben Smyth and David Bernhard. Ballot secrecy and ballot independence coincide. In *ESORICS'13: 18th European Symposium on Research in Computer Security*, volume 8134 of *LNCS*, pages 463–480. Springer, 2013.

[102] Ben Smyth and David Bernhard. Ballot secrecy and ballot independence: definitions and relations. Cryptology ePrint Archive, Report 2013/235 (version 20141010:082554), 2014.

[103] Ben Smyth, Steven Frink, and Michael R. Clarkson. Election Verifiability: Cryptographic Definitions and an Analysis of Helios and JCJ. Cryptology ePrint Archive, Report 2015/233 (version 20170213:132559), 2017.

[104] Ben Smyth and Alfredo Pironti. Truncating TLS Connections to Violate Beliefs in Web Applications. In *WOOT'13: 7th USENIX Workshop on Offensive Technologies*. USENIX Association, 2013. (First appeared at Black Hat USA 2013.).

[105] Ben Smyth and Alfredo Pironti. Truncating TLS Connections to Violate Beliefs in Web Applications. Technical Report hal-01102013, INRIA, 2015.

[106] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. Security Analysis of the Estonian Internet Voting System. In *CCS'14: 21st ACM Conference on Computer and Communications Security*, pages 703–715. ACM Press, 2014.

[107] Georgios Tsoukalas, Kostas Papadimitriou, Panos Louridas, and Panayiotis Tsanakas. From Helios to Zeus. *Journal of Election Technology and Systems*, 1(1), 2013.

[108] UK Electoral Commission. *Key issues and conclusions: May 2007 electoral pilot schemes*, May 2007.

[109] United Nations. *Universal Declaration of Human Rights*, 1948.

[110] Dominique Unruh and Jörn Müller-Quade. Universally Composable Incoercibility. In *CRYPTO'10: 30th International Cryptology Conference*, volume 6223 of *LNCS*, pages 411–428. Springer, 2010.

[111] Scott Wolchok, Eric Wustrow, J. Alex Halderman, Hari K. Prasad, Arun Kankipati, Sai Krishna Sakhamuri, Vasavya Yagati, and Rop Gonggrijp. Security Analysis of India's Electronic Voting Machines. In *CCS'10: 17th ACM Conference on Computer and Communications Security*, pages 1–14. ACM Press, 2010.

[112] Scott Wolchok, Eric Wustrow, Dawn Isabel, and J. Alex Halderman. Attacking the Washington, D.C. Internet Voting System. In *FC'12: 16th International Conference on Financial Cryptography and Data Security*, volume 7397 of *LNCS*, pages 114–128. Springer, 2012.

[113] Filip Zagórski, Richard T. Carback, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L. Vora. Remotegrity: Design and Use of an End-to-End Verifiable Remote Voting System. In *ACNS'13: 11th International Conference on Applied Cryptography and Network Security*, volume 7954 of *LNCS*, pages 441–457. Springer, 2013.