Bundesamt
für Sicherheit in der
Informationstechnik

Deutschland
Digital•Sicher•BSI•

A Study of Mechanisms for End-to-End Verifiable Online Voting

Studie von Mechanismen für Ende-zu-Ende Verifizierbare Onlinewahlen

Version history

Version	Date	Changes
2024-01	26.08.2024	Original version

Acknowledgments

This study was conducted by the *famoser GmbH* (Switzerland) on behalf of the *Bundesamt für Sicherheit in der Informationstechnik* as part of the project *Studie zu Ende-zu-Ende- Verifizierbarkeitsmethoden für Online-Wahlen (StuVe)*.

The authors of this study are:

- Florian Moser
- Johannes Müller
- Véronique Cortier
- Alexandre Debant
- Pierrick Gaudry
- Anselme Goetschmann
- Ralf Küsters
- Melanie Volkamer

This study is published in English due to its scientific nature and international audience.

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63, 53133 Bonn, Germany
E-Mail: online-wahlen@bsi.bund.de
Internet: https://www.bsi.bund.de

Executive summary

Introduction. In today's digital world, many elections are conducted online. In such elections, however, there is a risk that voters' votes may be altered unnoticed by the voting system as they pass through electronically, and therefore the will of the voters may not be accurately reflected in the final result. *End-to-end verifiability* is the gold standard to address this problem. With this fundamental property, it is possible to independently verify that the final result matches the votes received, even if parts of the voting system do not work correctly.

Problem statement. Over the past decades, many methods for online voting have been proposed, implemented and analyzed to provide end-to-end verifiability without compromising the secrecy of the voter's vote and overall usability. These works represent a vast body of knowledge, making it difficult to understand which of the proposed methods are the most appropriate and how they can be combined to realize end-to-end verifiable online voting.

Objective of the study. The goal of this study is to provide a low-threshold, systematic, and well-founded introduction to the complex world of end-to-end verifiable online voting.

Approach of the study. We have identified the most important cryptographic building blocks of end-to-end verifiable online voting, described them in detail, evaluated them from different perspectives, and discussed them.

Main results. Our key findings are as follows:

- *Holistic view*: In order to assess whether an online voting system meets the desired characteristics, it is important to know what each method does, but ultimately it is key to look at the voting system as a whole. In particular, even if the individual methods work properly, they must be linked together correctly to provide end-to-end verifiability.
- *Trust assumptions*: The ultimate goal of verifiable online voting systems with vote secrecy is to reduce the required trust in the various system components as much as possible. To effectively distribute trust in practice, it must be ensured that these parties are truly independent of each other.
- *Verifiable tallying*: We can expect any state-of-the-art verifiable online voting system to combine a secret ballot technique with a verifiable privacy-preserving tallying technique. In this way, independent auditors can verify the correctness of the election result, without having to trust the tallying authority, while keeping individual votes secret.
- *Voting device verification*: There is no one-size-fits-all solution to protect against possibly corrupted voting devices. Which voting device verification mechanism is appropriate for a given election depends on various election-specific requirements.
- *Everlasting privacy*: In many elections, it is necessary to protect privacy not only in the foreseeable future, but also in the long run, e.g. against quantum adversaries. There are feasible approaches to guarantee this property, called everlasting privacy, towards anyone who wants to verify the election, i.e., without compromising verifiability.

Conclusion. This study provides developers, regulators, researchers, election officials, decision makers, and all interested parties with a powerful toolbox for end-to-end verifiable online voting.

Contents

1 Introduction

Online elections have become an integral and growing part of our digitalized world. In an online election, voters enter their votes into computers (e.g., smartphones, PCs, tablets), which digitize them and send them over the Internet to a digital ballot box. The received ballots are then counted electronically and the results are made available online.

Secure online voting. The goal of an election in general is to capture and accurately reflect the will of the voters. However, there are two fundamental challenges to achieving this goal that must be addressed:

- *Ballot secrecy*: Every voter should be able to express their true will. Unfortunately, depending on the circumstances of an election, there is a risk that not every voter will be in such a position. For example, there may be a general discomfort with open voting, since voters may fear that they will sooner or later be disadvantaged if they openly express their will. In this case, which applies to many elections, it must be ensured that voters can cast their votes in *secret* and that they must remain secret during and after the election.
- *End-to-end verifiability*: Especially in online elections, there is a risk that votes can be digitally altered: it is not immediately apparent whether votes have been lost, added, or changed on the purely electronic path from the voting devices through the digital ballot box to the result announced online. Such changes can be caused not only by intentional manipulation, but also by unknown software bugs. However, to ensure that the final result is accepted only if it correctly reflects the votes of the voters, even if parts of the voting system do not function properly, the voting system must be *end-to-end verifiable*.

At first glance, it is far from clear whether and, if so, how these two contradictory properties can be combined in online elections. On the one hand, ballot secrecy requires that the individual connections between voters and their votes in the result remain secret, while end-to-end verifiability requires that it is possible to check that these connections remain intact throughout the complete digital trail.

A plethora of knowledge. Fortunately, modern cryptography provides several viable methods for resolving this conflict. For more than forty years, researchers and developers have designed, implemented, and analyzed end-to-end verifiable online voting systems that provide ballot secrecy and other useful properties. Beginning with the pioneering work of Josh Benaloh in the 1980s [29, 17], hundreds of academic papers have been published on this challenge over the past four decades, and numerous online voting systems have been implemented that aim to combine ballot secrecy and end-to-end verifiability using cryptography.

The primary goal of these methods is to place as little trust as possible in the individual components of the voting system, in order to be able to convince oneself as an independent auditor of the correctness of the final result, while at the same time not revealing more information about the individual votes than can be derived from the final result anyway.

While this plethora of knowledge holds enormous potential for the realization of secure online elections, it also poses a great challenge. The jungle of proposals, discussions, analyses and systems is almost impossible to keep track of. What methods exist and what are their goals? How secure and

practical are they? Which components can be combined to build a complete, end-to-end verifiable online voting system? What are the pitfalls?

This work. The purpose of this study is to provide a comprehensive reference to answer the previous questions:

1. *Identification of key methods*: Based on an extensive market and literature analysis, we have selected eight key methods for end-to-end verifiable online voting that we will study in more detail in this work. These methods differ in their purposes, underlying assumptions and cryptographic building blocks.
2. *Introduction of evaluation criteria*: We introduce evaluation criteria for the four most important aspects of secure online voting. These are secrecy, verifiability, usability, and practicality.
3. *Description of methods*: For each of these eight methods, we explain in detail which requirements (cryptographic or systemic) are necessary to realize them. We then describe how these methods are integrated into an online voting system.
4. *Evaluation of methods*: We evaluate each of these methods with regard to the four aspects mentioned above. In this way, we can determine their individual characteristics.
5. *Discussion*: Based on our evaluation results, we discuss the advantages and disadvantages of the different methods and give recommendations for their deployment in practice.

Our study is intended for readers without a strong technical background who want to familiarize themselves with the topic of end-to-end verifiable online elections. While we assume that the readers have some basic knowledge of computer science, we will introduce all the cryptographic and online voting concepts necessary to understand our study from the ground up.

Structure. Our study is organized as follows. In Chapter 2, we present the necessary background in online voting and cryptography to follow our study. In Chapter 3, we introduce our four evaluation criteria. In Chapter 4, the main work of our study, we describe and evaluate the eight different methods we selected. In Chapter 5, we summarize and discuss our results.

2 Background

2.1 End-to-end verifiable online voting

In this section, we give a brief introduction to verifiable online voting systems. We will describe their main components/participants and overall flow, their main security and privacy goals, and their basic common approach to guarantee these properties.

2.1.1 Overview

Table 2.1: Notation for e-voting protocols.

Variable	Meaning
EA	election authority
RA	registration authority
$\mathsf{V}_1, \ldots, \mathsf{V}_n$	voters
v_i	V_i's vote
VD	voting device
ρ	result function
$\mathsf{T}_1, \ldots, \mathsf{T}_m$	tallier
$\mathsf{M}_1, \ldots, \mathsf{M}_l$	mix servers
VS	voting server
BB	bulletin board
A	auditor

An online voting protocol is run between an *election authority* EA, a *registration authority* RA, a set of *voters* $\mathsf{V}_1, \ldots, \mathsf{V}_n$ with voting devices VD, a *voting server* VS, and a *tallier* T (see Table 2.1). The election authority EA is responsible for setting up the election (date, set of candidates, voting method, etc.). The registration authority RA authenticates the voters to ensure that only eligible voters can vote. During the submission phase, each voter V_i enters their vote v_i to their voting device VD, which digitally encodes the vote and submits it over the internet to the voting server VS. In the tallying phase, the voting server VS sends these digital votes to the tallier T, which applies the specified voting method ρ to these votes (e.g., counts the number of votes per candidate), and finally returns the election result $\rho(v_1, \ldots, v_n)$ to VS.

Obviously, all parties in the voting protocol sketched above must be trusted in all important aspects. For example, T would learn the clear choice of each digitally encoded vote, and if T were dishonest, it could manipulate the election outcome undetected. Another example is the voting devices VD, which receive the voters' clear choices and thus learn how each voter voted, and which can secretly manipulate those votes. Such weaknesses can be avoided by designing online voting systems to combine several security and privacy features.

2.1.2 Properties

We describe the most important properties of verifiable online voting. These include secrecy, verifiability, usability and practicality. Here, we explain these properties on an intuitive level, which is

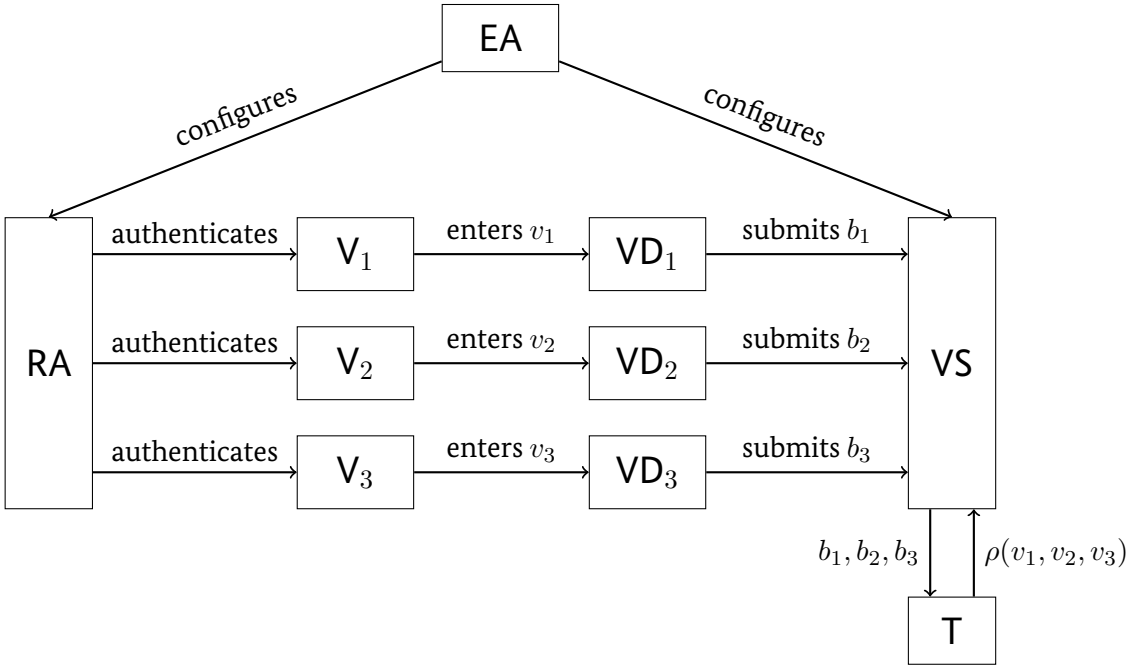

Figure 2.1: Illustration of a voting protocol.

sufficient to follow our exposition, and in Chapter 3, we will define them precisely to later evaluate the different methods in Chapter 4.

Secrecy (Section 3.2). To ensure that election results reflect the true, unbiased will of the voters, the voting system must provide an environment in which voters do not fear disadvantage for casting a vote for their preferred choice. There are different aspects of this challenge that need to be considered for any election and resolved when necessary. In the following, we describe three important properties, namely vote privacy, everlasting privacy, and vote-buying resistance, that relate to this challenge and that we will use in our evaluation. The first property, vote privacy, is the most basic and is commonly required, while the other two may or may not be required depending on the type of election.

- *Vote privacy* (Section 3.2.1) guarantees that the links between individual voters and their votes in the (public) final result remain secret. To this end, it should not be possible for an observer to obtain more information during the course of an election than what can be inferred from the final result. Such malicious observers can be external actors as well as attackers who corrupt parts of the election system. Therefore, the role of the tallier T is often distributed among several entities $\mathsf{T}_1, \ldots, \mathsf{T}_m$ so that only a threshold of them need to be trusted for privacy.[1]

- *Practical everlasting privacy* (Section 3.2.2) guarantees that the audit data used to verify an online election (see below) keeps the individual votes *unconditionally secret*, i.e. without any computational hardness assumptions.

  The background is that the verifiability of voting systems, as we explain below, requires that certain data be shared in order to verify that the voters' will is correctly reflected in the final result. Depending on how this property is realized at the cryptographic level, the audit data keeps the individual votes secret only under certain computational hardness assumptions.

[1] Moreover, depending on how trust is distributed, the protocol becomes more robust in case a tallier is not able (or willing) to participate in tallying.

While such hardness assumptions may hold at the time of the election, there is a risk that they may not hold in the future due to new cryptanalytic techniques or more powerful computers (such as quantum computers). Thus, if an attacker stores conditionally secret audit data and uses it to deduce the votes of individual voters in the future, this leakage can have negative effects on voters even after several decades, depending on the election. Verifiable online voting systems with practical everlasting privacy avoid this threat because their audit data keeps the voters' individual choices unconditionally secret.

- *Vote-buying resistance* (Section 3.2.3) makes it more difficult for the final result to be influenced by vote-buying. While it is not possible to prevent voters from selling their votes by purely technical means, it is possible to build in mechanisms that prevent voters from convincingly proving to third parties (especially vote buyers) how they voted. In this way, the incentive to buy votes is economically undermined.

Verifiability (Section 3.3). Another fundamental challenge is to be confident that the votes cast by voters are accurately reflected in the election result. The gold standard for meeting this challenge is end-to-end verifiability, which can be enhanced to accountability.

- *End-to-end verifiability* ensures that it is possible to verify that the final election result is correct, even if some of the participants are corrupted. Since the main purpose of verifiability is to protect against possibly corrupted parties, verifiability should (ideally) not be based on any trust assumptions.

- *Accountability* is a stronger notion of verifiability. While verifiability enables auditors to verify whether the final election result correctly reflects the votes chosen by the voters, verifiability alone is not sufficient to detect which parties manipulated the final outcome. Accountability solves this problem as it enables one to identify misbehaving parties individually and hold these parties accountable. This property can be particularly useful in practice to resolve/avoid possible disputes and to increase the system's robustness.

Usability (Section 3.4). In order to guarantee the secrecy and verifiability properties described above not only in theory, but also in practice, it is crucial that the human components of voting systems (especially the voters) can successfully perform their roles. Otherwise, the security of the voting system may be rendered completely insecure in real-world elections, even if it appears to work properly in gray theory.

Therefore, it is important to study the *usability* of a voting system. Special attention is paid to the voting procedures and to the individual verification procedures of the voters. Only if the voters can perform these procedures efficiently (in a reasonable time), effectively (reaching the goal), and satisfactorily (in a comfortable way), they fulfill their desired function: casting a vote and, if supported by the system, verifying that the voting device processed the cast vote as intended. If these steps are not sufficiently usable by voters, this has fundamental implications. If voters are unable to cast their vote, this prevents them from participating in the election, which undermines the universality of the election. If voters are unable to verify their voting devices, corrupt voting devices can manipulate votes unnoticed, which subverts the integrity of the election result.

Practicality (Section 3.5). It is the *practicality* of a voting system that ensures that it can be implemented correctly in practice and that the resulting implementation is efficient enough for the intended election. We will consider the following two basic aspects of practicality:

- *Implementability* (Section 3.5.1) captures the effort required to implement a mechanism. There are various constraints (e.g., required skills, limited resources, and the complexity of

the protocol) that decrease implementability, while some enablers (e.g., existing software libraries) can increase it.

- The *efficiency* (Section 3.5.2) of a mechanism limits the election settings to which it can be applied in practice. These settings are determined by, among other things, the equipment of the voters and election officials, and the complexity of the ballots (e.g., number of candidates). Therefore, before using a voting system for a particular election, it is important to be able to assess whether the mechanism will be efficient in that environment.

2.1.3 General approach

While it may not be intuitively obvious that the conflicting properties of verifiability and vote privacy can be combined, there are several methods to efficiently resolve this seeming contradiction using cryptography. These methods differ, among other things, in the specific functions they are intended to fulfill within the electoral system. In the following, we present the main components according to their function.

Bulletin board. Verifiable e-voting protocols commonly implement a *bulletin board* BB, which broadcasts all data that is necessary to verify the correctness of the final result. Depending on the election scenario or voting system, this data may be accessible to everyone (voters, observers, etc.) or only to certain parties. In the following, we consider the parties with access to this data as a single entity, which we call the *auditor* A.

Secret ballots. During the submission phase, the voters $\mathsf{V}_1, \ldots, \mathsf{V}_n$ post some information about their votes on BB. In order to protect their own privacy, voters seal their votes v_i in secret ballots. Most commonly, the voters encrypt their secret votes and post the resulting ciphertext to BB. The ciphertext can only be decrypted by the tallier T, or in case of distributed trust, jointly by the different talliers $\mathsf{T}_1, \ldots, \mathsf{T}_m$.

Authenticated ballots. Authenticating ballots is necessary to ensure their eligibility. The internal registration authority RA responsible for this task has special power because it can effectively decide which ballots are accepted for voting and which are not. To limit the power of RA, it is helpful to introduce alternative authentication methods that do not require trusting RA. A common way to do this is to use digital signatures, whose trustworthiness is guaranteed by system-independent means (e.g., electronic IDs). While the building block itself does not require any election-specific properties, it can prevent a corrupted RA from stuffing the ballot box with illegitimate ballots.

Voting device verification. Since the voting devices use cryptographic techniques to keep the voters' individual choices secret, it is impossible for a human voter to directly check that their voting device VD cast their vote as intended. There exist different techniques to empower voters to verify that their original choices have been processed correctly. Some techniques employ, for example, separate audit devices or applications, while others use human-readable codes.

Verifiable privacy-preserving tallying. In the tallying phase, the talliers $\mathsf{T}_1, \ldots, \mathsf{T}_m$ employ some secret knowledge (e.g., private keys) to process the secret ballots on BB and compute the final result $res = \rho(v_1, \ldots, v_n)$. During this process, the individual links between the voters and their choices in the final result must be kept secret to maintain vote privacy. The two most common ways to guarantee this property are secret shuffling of the ballots and secret aggregation of the choices:

- *Secret shuffling*: The links between the received secret ballots and the output plaintext ballots are obscured by randomly shuffling the secret ballots and then outputting only the shuffled ballots in plaintext at the end. This method is equivalent to the way ballot secrecy is maintained in traditional paper-based elections: voters place their completed secret ballots in a ballot box, which is then shaken out at the end of the election so that the individual ballots in the pile can no longer be assigned to individual voters.
- *Secret aggregation*: This method secretly adds up the individual votes for each candidate, and then outputs these totals. Thus, the final result is the total number of votes cast for each candidate. Each individual vote thus disappears in the anonymized total. This type of counting corresponds to non-digital elections where each candidate has its own ballot box and voters can cast their votes in the form of stones (or similar) into these individual ballot boxes. The number of stones per ballot box then equals the total number of votes cast for that candidate.

The talliers then post the result to the bulletin board BB. To prove that they executed the privacy-preserving technique correctly, the talliers also compute a proof that can be verified by anyone with access to the encrypted ballots and the election result. This proof does not reveal any secret knowledge of the talliers, such as information about private keys. The talliers then post this proof to BB.

2.2 Cryptography

All verifiable online voting systems have cryptographic building blocks at their core. In this section, we will introduce the most important recurring cryptographic building blocks:

- *Public-key encryption* (Section 2.2.1) is used in most verifiable voting systems to encrypt sensitive data, such as votes, with a public key so that only selected parties who know the corresponding secret key can read it.
- *Commitments* (Section 2.2.2) are often used for similar purposes, with the difference that the sensitive data cannot be read with a message-independent secret key, but only with specific information that is generated during the individual commit process and then shared with selected parties.
- *Digital signatures* (Section 2.2.3) are commonly used in voting systems so that different parties can verify that the messages they receive are from the indicated party.
- *Zero-knowledge proofs* (Section 2.2.4) allow a party to prove that it performed a certain computational step correctly, without having to reveal any further information (such as the secret key used in the computation). These building blocks are central to combine the competing but desirable properties of public verifiability and secrecy of votes.
- *Threshold secret sharing* (Section 2.2.5) can be used to distribute information about a secret (e.g., a secret key) among multiple parties, so that more than a certain threshold of them must cooperate to recover the secret from their individual shares.

For more detailed information on modern cryptography, we refer the reader to the relevant literature, for example [82, 54, 55, 19] (English) or [105] (German).

2.2.1 Public-key encryption

A *public-key encryption (PKE) scheme* enables everyone to encrypt messages so that the content of the resulting ciphertexts can only be read by the designated receiver but no-one else.

Figure 2.2: Illustration of a public-key encryption scheme with public/private key pair (pk, sk).

Notation. In general, a *public-key encryption (PKE) scheme* is a tuple of polynomial-time algorithms $\mathcal{E} = (\mathsf{KG}_e, \mathsf{Enc}, \mathsf{Dec})$:[2]

- The probabilistic key generation algorithm KG_e outputs a public/private key pair (pk, sk).[3]
- The probabilistic encryption algorithm Enc takes as input a public key pk and a message m, and outputs a ciphertext e.
- The deterministic decryption algorithm Dec takes as input a secret key sk and a ciphertext e, and outputs a message m'.

[2] Cryptographic algorithms usually depend on a *security parameter* that determines the concrete security level of the scheme. Throughout this paper, we assume that the security parameter is an implicit input to all algorithms.

[3] We assume that the public key pk implicitly defines the set of valid messages $\mathcal{M}$.

Correctness. A PKE scheme is *correct* if and only if for all public/private key pairs $(pk, sk) \leftarrow \mathsf{KG}_e$, for all messages m, and for all encryptions $e \leftarrow \mathsf{Enc}(pk, m)$ of m with public key pk, the decryption of e with the secret key sk returns the originally encrypted message m, i.e., $m = \mathsf{Dec}(sk, e)$.

Security. A PKE scheme is secure if any ciphertext $\mathsf{Enc}(pk, m)$ does not leak any information about the secret message m if the corresponding decryption key sk is secret. More precisely, for our application, we need a PKE scheme that is secure against chosen-plaintext attacks (CPA). This notion guarantees that for any public/secret key pair (pk, sk) of this PKE scheme, an attacker cannot distinguish whether a ciphertext c encrypts message m_0 or m_1, even if the attacker knows the public key pk and can choose the messages m_0, m_1. See, e.g., [82] for a formal definition of this notion.

We note that the security of PKE schemes (and of public-key cryptography in general) can only be guaranteed for computationally limited adversaries. The reason is that computationally unbounded adversaries can always "brute-force" the secret key sk from the corresponding public key pk, i.e., try out all possible secret keys sk' until they find the correct one sk. Therefore, the security of specific PKE schemes is based on certain problems that are assumed to be computationally intractable for potential adversaries.

2.2.2 Commitments

A *commitment scheme (CS)* enables everyone to commit to a message so that:

1. Anyone, who does not know how the resulting commitment was created, is unable to derive the original message (*hiding*).
2. And everyone can be convinced by the committing party that the commitment in fact commits to the original message (*binding*).

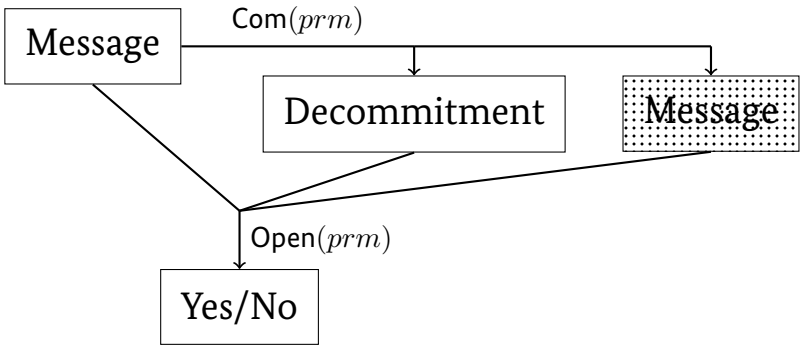

Figure 2.3: Illustration of a commitment scheme.

Notation. A *commitment scheme (CS)* (Figure 2.3) is a tuple of polynomial-time algorithms $\mathcal{C} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$:

- The probabilistic setup algorithm Setup outputs the public parameters.
- The probabilistic commitment algorithm Com takes as input the public parameters prm and a message m, and outputs a commitment/opening pair (c, d); the value d is sometimes called *decommitment*.
- The opening/verification algorithm Open takes as input public parameters prm, a message m, a commitment c, and an opening value d, and outputs a bit b.

The commitment algorithm Com is run by the committer to commit to a message m. The opening algorithm Open can be run by anyone who wants to verify that a given commitment c actually commits to m; to do so, the committer reveals the opening value d to the verifier.

If the committer runs Com and reveals the correct opening value d, then the opening algorithm returns 1 ("accept"), otherwise 0 ("reject"); this is formalized by the following correctness property.

Correctness. A CS is *correct* if and only if for all public parameters $prm \leftarrow$ Setup, for all messages m, and for all commitment/opening pairs $(c, d) \leftarrow \mathsf{Com}(prm, m)$, the opening algorithm accepts the tuple (prm, m, c, d), i.e., $\mathsf{Open}(prm, m, c, d) = 1$.

Security. The *hiding* property of commitment scheme guarantees that any observer cannot distinguish whether a given commitment c is for some message m_0 or m_1. This notion is formalized similar to the CPA notion for the secrecy of PKE schemes (Section 2.2.1). For all public parameters $prm \leftarrow$ Setup and for all possible message pairs (m_0, m_1), the distributions of their resulting commitments $\mathsf{Com}(prm, m_0)$ and $\mathsf{Com}(prm, m_1)$ are indistinguishable. Depending on the specific CS, the distributions are (in order of hardness) computationally indistinguishable, statistically indistinguishable, or identical.

The *binding* property of commitment schemes guarantees that the committer cannot create a commitment c that is valid for two different messages $m_0 \neq m_1$. This notion is formalized as follows. For all public parameters $prm \leftarrow$ Setup, the probability that any potential adversary can compute a tuple (c, m_0, m_1, d_0, d_1) such that $m_0 \neq m_1$ and $\mathsf{Open}(prm, m_0, c, d_0) = 1$ and $\mathsf{Open}(prm, m_1, c, d_1) = 1$ is negligible. Depending on the specific CS, the binding property can be unconditional or based on the hardness of a computational problem.

Since commitment schemes are *keyless*, such a scheme can in principle be unconditionally hiding or unconditionally binding (but never both). This is in contrast to PKE scheme, whose secrecy is always conditional (Section 2.2.1). Unconditionally hiding commitment schemes are useful to design verifiable online voting protocols with practical everlasting privacy (Section 4.3).

2.2.3 Digital signatures

A *digital signature (DS) scheme* enables a party to sign messages so that everyone can convince themselves that the signatures were created by that party but no-one else.

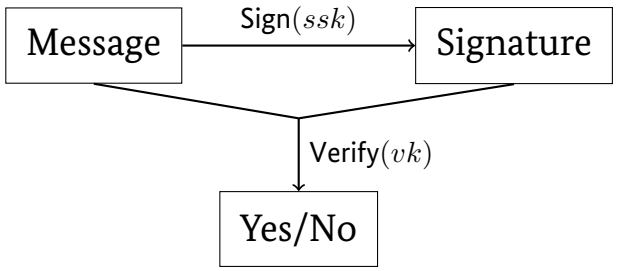

Figure 2.4: Illustration of a digital signature scheme with verification/signing key pair (vk, ssk).

Notation. A *digital signature (DS) scheme* (Figure 2.4) is a tuple of algorithms $\mathcal{S} = (\mathsf{KG}_s, \mathsf{Sign}, \mathsf{Verify})$:

- The probabilistic key generation algorithm KG_s outputs a verification/signing key pair (vk, ssk).
- The probabilistic signature algorithm Sign takes as input a secret signing key ssk and a message m, and outputs a digital signature s.

- The verification algorithm Verify takes as input a verification key vk, a message m, and a signature s, and outputs a bit b.

Correctness. A DS scheme is *correct* if and only if for all verification/signing key pairs $(vk, ssk) \leftarrow \mathsf{KG}_s$, for all messages m, and for all signatures $s \leftarrow \mathsf{Sign}(ssk, m)$, the verification algorithm accepts the tuple (vk, m, s), i.e., $\mathsf{Verify}(vk, m, s) = 1$.

Security. On an intuitive level, a DS scheme is secure if it is infeasible to create a valid fresh signature without knowing the secret signing key, even when a possible attacker can choose arbitrary messages to be signed and returned by the signer (which knows the secret signing key ssk). This property is reflected in the common notion of *EUF-CMA (Existential Unforgeability under Chosen Message Attack)* security, which we assume to hold for the DS schemes used in this work.

2.2.4 Zero-knowledge proofs

A *zero-knowledge proof (ZKP)* enables a party to prove that a certain statement holds true (*soundness*) without revealing any information beyond the correctness of that statement (*zero-knowledge*).

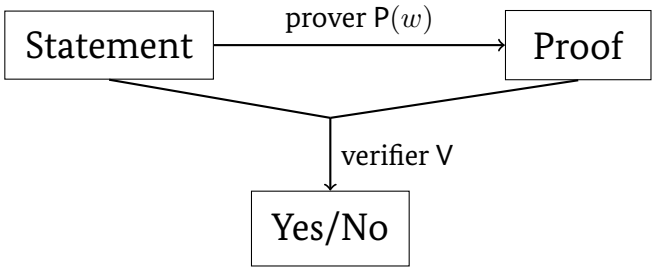

Figure 2.5: Illustration of a non-interactive zero-knowledge proof for some statement with witness w.

Notation. A ZKP is a protocol that is run between two parties, the prover and the verifier. The prover wants to convince the verifier that a certain statement, denoted x, is true. The prover has some secret knowledge, the *witness* w. The prover uses w to prove that x is correct without revealing any information about w beyond what can be derived from the statement x anyway.

More formally, each ZKP is defined for a language $\mathcal{L}$, which specifies the supported statements. The languages of the ZKPs that we study in this paper are all subsets of the complexity class NP. They can thus be represented as $\mathcal{L} = \mathcal{L}_{\mathcal{R}} = \{x \colon \exists w \colon (x, w) \in \mathcal{R}\}$, where the variable x denotes a *statement* that is in relation $\mathcal{R}$ with some *witness* w.

Most of the ZKP systems used for verifiable e-voting are *non-interactive zero-knowledge proof (NIZKP)* systems. In these systems, the prover creates a proof without interacting with the verifier. In particular, anyone who receives the proof can verify its correctness. A NIZKP for some language $\mathcal{L}_{\mathcal{R}} = \{x \colon \exists w \colon (x, w) \in \mathcal{R}\}$ is a tuple of algorithms (P, V):

- The probabilistic prover algorithm P takes as input a statement/witness pair $(x, w) \in \mathcal{R}$, and outputs a proof π.
- The deterministic verifier algorithm V takes as input a statement x and a proof π, and outputs a bit b.

Correctness. A NIZKP (P, V) is *correct* if and only if for all statement/witness pair $(x, w) \in \mathcal{R}$ and all proofs $\pi \leftarrow \mathsf{P}(x, w)$, the verifier accepts the proof π, i.e., $\mathsf{V}(x, \pi) = 1$.

Soundness. On an intuitive level, the *soundness* property of a NIZKP guarantees that a dishonest prover cannot create a valid proof for a false statement. This notion is formalized as follows. For all invalid statements $s^* \notin \mathcal{L}$ and any (possibly corrupted) P^*, the probability that the verifier V accepts a "proof" π^* from P^* is negligibly small.

Zero-knowledge. On an intuitive level, the *zero-knowledge* property of a NIZKP guarantees that no more information can be extracted from the proof than the correctness of the statement. This notion is formalized as follows. For any possibly corrupted verifier V^*, there exists a probabilistic polynomial-time algorithm S^* (the *simulator*) that outputs for all $x \in \mathcal{L}$ a simulated "proof" π^*. The point is that the simulator creates this "proof" without knowing the witness, but such that simulated proofs are indistinguishable from real proofs $\pi \leftarrow \mathsf{P}(x, w)$ that were computed with the witness. For this purpose, the simulator is allowed to go back to an earlier point in the protocol run, and use knowledge of the later trace (*rewinding*); this allows the simulator (unlike the actual prover) to anticipate possible challenges of the verifier.

2.2.5 Threshold secret sharing

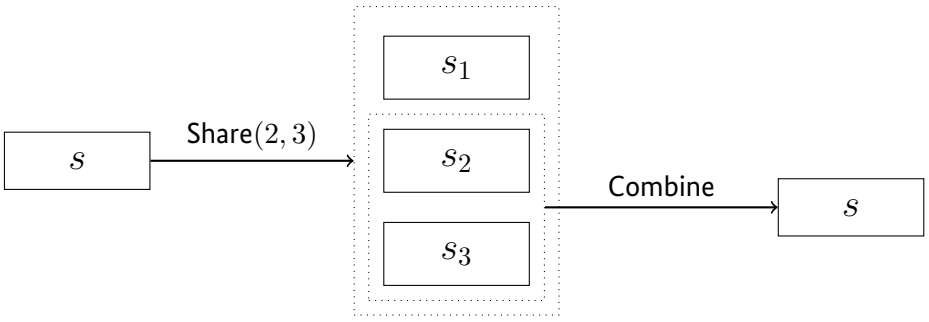

Figure 2.6: Illustration of a secret sharing scheme with three shares, out of which two are needed to reconstruct the secret.

A (t, n)-*threshold secret sharing scheme* allows to distribute knowledge about a secret s in the form of shares $s_1, \ldots, s_n$ among n different parties such that any t of the n shares are sufficient to reconstruct the secret s, but any set of less than t shares reveals nothing about s.

Notation. A threshold secret sharing scheme (over some arbitrary finite set $\mathcal{S}$) is a pair of efficient algorithms $(\mathsf{Share}, \mathsf{Combine})$:

- Share is a probabilistic *sharing algorithm* that takes as input (t, n, s), where $t \leq n$ are positive integers and $s \in \mathcal{S}$, and returns a vector $(s_1, \ldots, s_n)$ with elements in $\mathcal{S}$.
- Combine is a deterministic *combining algorithm* that takes as input $(I, (s_i)_{i \in I})$, where I is a subset of $\{1, \ldots, n\}$ of size t and $(s_i)_{i \in I}$ is a vector of elements in $\mathcal{S}$, and returns an element $s \in \mathcal{S}$.

Correctness. A (t, n)-threshold secret sharing scheme is *correct* if for all $s \in \mathcal{S}$, for all outputs $s_1, \ldots, s_n$ of $\mathsf{Share}(t, n, s)$, and for all subsets I of $\{1, \ldots, n\}$ with size t, we have $\mathsf{Combine}(I, (s_i)_{i \in I}) = s$.

Security. A (t, n)-threshold secret sharing is *secure* if every set of $t - 1$ shares output by $\mathsf{Share}(t, n, s)$ reveals nothing about s. This notion is formalized similarly to the CPA-security of PKE schemes (Section 2.2.1) and to the hiding property of commitment schemes (Section 2.2.2). We say that a (t, n)-threshold secret sharing scheme is secure, if for all secrets s, s', the distributions of any set of less than t shares of $\mathsf{Share}(t, n, s)$ and of any set of less than t shares of $\mathsf{Share}(t, n, s')$ are indistinguishable.

3 Evaluation criteria

We introduce the four criteria that we will use to evaluate the different methods for verifiable online voting (Chapter 4): vote secrecy (Section 3.2), verifiability (Section 3.3), usability (Section 3.4), and practicality (Section 3.5).

Remark 1: Expressiveness

The following points should be noted:

- *Scoring system*: For three of the four properties (secrecy, usability, and practicality) we will introduce scoring systems that give some indication of the degree of effect on the property. We would like to make it clear at this point that these numbers should not be taken as absolute truths, but serve to provide a rough classification.
- *Focus and scope*: While the focus of our evaluation is on the individual methods, the entire electoral system must always be considered for a complete analysis. Such a comprehensive analysis, taking into account all components of the online voting system in detail, is beyond the scope of this study.

Before going into the technical definitions of these criteria, we first describe the legal background of online voting in general, and how our criteria relate to it.

3.1 Legal background

The Council of Europe describes the common principles for democratic elections of its member states in its *Code of Good Practice in Electoral Matters* [35]. The Code identifies the principles of *universal*, *equal*, *free* and *secret* elections with *direct suffrage*. In addition, basic conditions must be met, including the rule of law, respect for fundamental rights (e.g. freedom of expression, freedom of assembly), stability of the electoral law and effective procedures.

To implement the election principles for e-voting, as defined in the Code of Good Practice, the Council of Europe describes in its *Recommendation CM/Rec(2017)51 on standards for e-voting* [36] a set of minimum requirements that an online voting system must meet in particular. These minimum requirements must be fulfilled by any online voting system, while they may be supplemented by additional standards of the legal framework applicable to the specific election. For example, in Germany, elections at the national level are additionally governed by the principle of "the public nature of elections", which requires that essential parts of the election be publicly verifiable ("öffentlich nachvollziehbar") [23].

We summarize the minimum requirements as given by the Recommendation as follows:

- *Universal*: The voting system must be easy to use and accessible to all voters, and voters must be aware that they are voting in a real election.
- *Equal*: Official voting information must be presented equally to other voting channels. Further, the system must effectively authenticate voters and their ballots, respecting the "one person, one vote" principle.
- *Free*: The voting system must not influence the voter's intention, and the voter must be able to determine whether their vote was cast as intended. Further, the voting system must prove that all authenticated votes are accurately included in the final result, and the final result consists only of votes of eligible voters.
- *Secret*: The voting system must effectively protect the secrecy of voter registers, votes, and preliminary results. In addition, the voter must not receive a receipt for their vote, and their unsealed vote must not be traceable to them.

In this study, we focus on the technical aspects to implement these legal minimum requirements. We use the following approaches:

- *Secrecy*: The voters' choices remain secret except for what is learned from the election result. This realizes the secret voting principle.
- *Verifiability*: The announced result of the election is the actual choice of the voters. This realizes the principles of equal and free elections.
- *Usability*: The voting system is efficient, effective and satisfying to use, while remaining accessible to all. This realizes the principle of universal suffrage.
- *Practicality*: The voting system, respectively the concrete mechanisms it is composed of, is feasible to implement and operate.

3.2 Secrecy

We present our privacy criterion to evaluate the verifiable e-voting mechanisms. We divide it into three subcriteria: Vote privacy (Section 3.2.1), everlasting vote privacy (Section 3.2.2) and vote buying resistance (Section 3.2.3).

Model. To analyze the three secrecy aspects, we refine our general voting protocol model (Section 2.1) as follows:

- *Registration authority*: We regard that part of the election authority EA that registers voters as a separate entity, called the *registration authority* RA.
- *Talliers*: For each tallier T, we distinguish between the *tally trustee* (the party that processes the submitted secret ballots), and the *decryption trustee* (which decrypts the processed ballots). For the decryption trustee, we distinguish even further between the *decryption machine* (the computer that the decryption trustee uses) and the *secret key storage* (the medium on which the secret decryption key material is stored).

We further assume that the roles of all authorities and talliers can be distributed among different entities (to reduce trust).

Types of information. We further distinguish between the following types of information:

- *Public information*: This includes all general information provided to the voters (e.g., dates, election parameters, talliers' public key). Such data could, for example, be shared on a public part of the bulletin board.
- *Semi-public information*: This includes all verification data on the bulletin board BB. This contains, among others, the voters encrypted ballots and possible data shared by the election authorities. Since they can be accessed by the election auditors, we consider this data semi-public.
- *Receipts*: This includes any information that a voter, who follows their prescribed role, receives (e.g. by email) or that a voter is asked to save.
- *Secret information*: Some data is kept secret by the trusted parties. This includes, for example, decryption keys if the decryption authorities are trusted. The adversary never gets access to this data, so these assets do not appear in our evaluation.

Overall secrecy score. We assign an overall score for secrecy to each online voting system. The idea is: the higher the score, the better the protection of secrecy.

The overall score is composed of three partial scores, namely the scores for vote privacy, vote-buying resistance, and everlasting privacy. For each of these aspects, the voting system can reach a certain level and the number of this level corresponds to the score for this aspect.

We then combine these partial scores as follows to determine the overall score for secrecy. If the score of vote privacy is zero, then the overall score is also zero. Otherwise, to evaluate the overall secrecy score, we add together the doubled score of vote privacy (because this is the most fundamental secrecy property), the score of everlasting privacy, and the score of vote-buying resistance.

3.2.1 Vote privacy

Vote privacy guarantees that an observer or an attacker cannot learn any information about the votes of individual voters beyond what can be inferred from the public election result. We evaluate the vote privacy of a voting system according to the types of attackers against which it provides vote privacy. The stronger the potential attackers are, the higher we rate the vote privacy of the voting system. The final level provides the strongest protection of vote privacy that an online voting system can achieve (under realistic, practical assumptions).

At the first level, we consider only external attackers. These attackers have access to all data sent over the network during the election, to the public information available to all participants, and to all receipts that voters receive when they cast their votes. We expect any voting system with a minimum level of vote privacy to keep voters' votes secret from such attackers.

Vote privacy: Level 1

The attacker learns all data sent on the network, all public information, and all voter receipts.

To achieve levels higher than level 1, the voting system must guarantee vote privacy even if the attacker can passively or even actively corrupt certain components of the voting system.

We start with level 2 by assuming that the attacker can either (1) passively corrupt the voting server and learn all of the internal data of that server, in particular the log data of individual voters and their votes cast,[1]

Vote privacy: Level 2

The attacker can additionally control (at most) one of the following machines:

(1) An honest-but-curious voting server: the voting server follows its dedicated program, but may leak all internal data to the attacker.

(2) One malicious tally trustee and one malicious decryption machine: the attacker can fully control the actions of one tally trustee and one decryption machine and learns all internal data.

At level 3, we strengthen the potential attackers by providing them with the secret key shares of the decryption trustees up to a threshold and allowing them to learn all data necessary to verify the election result, including all intermediate ciphertexts and corresponding (non-interactive zero-knowledge) proofs.

Vote privacy: Level 3

The attacker can additionally access secret key storage of up to a threshold of the decryption trustees, and all data provided to the auditors (i.e., all semi-public information).

At level 4, we also consider the registration authorities, whose trustworthiness is relevant to vote privacy for the following reason. Since it is the task of the registration authorities to ensure the eligi-

[1] Despite its practical importance, we do not consider in our evaluation the case where the attacker can actively corrupt the voting server, neither in terms of verifiability nor in terms of privacy. The reason for this is that in our model, the voting server also hosts the bulletin board, and in this work we do not study methods for securing the bulletin board (and thus the voting server) against active attacks. We refer to Section 4.10.1 for work that addresses this problem. or (2) actively corrupt a subset of the parties responsible for counting the votes (the talliers and the decryption authorities). Scenario (1) requires that the votes are not sent to the voting server in clear text, but in encrypted form. Scenario (2) requires that the votes in the result output by the tallying parties cannot be linked to the identities of individual voters.

bility of the ballots, sufficiently many corrupt registration authorities can prevent voters from voting (in a more or less subtle way). This would reduce the number of ballots and thus the *anonymity set* in which individual ballots should hide. To reach level 4, we expect the voting system to provide vote privacy even if up to a threshold many registration authorities can be actively corrupted. In addition, we also allow possible attackers to actively control up to a threshold many of the decryption trustees and tally trustees.

Vote privacy: Level 4

The attacker can additionally control up to a threshold of the registration authorities, up to a threshold of the decryption trustees, and up to a threshold of the tally trustees.

At the highest level, we expect that the voting system also protects vote privacy even if the voters' voting clients are corrupted.

Vote privacy: Level 5

The attacker can additionally corrupt all voting clients.

3.2.2 Everlasting privacy

The general idea of everlasting privacy is that the audit data that are necessary to verify an election should not compromise vote privacy, even if computational hardness assumptions are broken after the election (e.g., because all the decryption keys are leaked, new insights into its cryptanalysis, or more powerful computers). We divide the degree of everlasting privacy into four levels, with everlasting privacy being stronger the higher the level reached. The lowest level is 0, which is given when no higher level can be achieved or when no reasonable degree of verifiability can be provided (Section 3.3).

At the first level, we make the general minimal assumption that all public information is also available to the attacker, and the more specific assumption that the attacker obtains all secret keys of the decryption trustees.

Everlasting privacy: Level 1

The attacker has access to all public information and to the secret key storage of all decryption trustees.

At the next level, the attacker can also access all the data needed to verify the election.

Everlasting privacy: Level 2

The attacker additionally has access to all data provided to auditors (semi-public information).

At the highest level, we make the weakest trust assumption under which everlasting privacy can be achieved at all: that the attacker can access all data except the data of the voting server.

Everlasting privacy: Level 3

The attacker additionally has access to the data of all participants except the voting server.

3.2.3 Vote-buying resistance

The general principle is that the system should protect against vote buying, i.e. a voter should be able to lie about their vote and their material so that they cannot convince a buyer.

We will use the term *short-term voting material* to refer to any election-specific material (login, password, credential, etc.), in contrast to the *long-term voting material* which refers to any election-independent material (e.g., ID card, professional login and password, single sign-on).

There are different types of vote-buying and they cannot be directly compared (i.e., they are neither stronger nor weaker than the other). We take this into account by specifying that voting systems can in principle achieve the same level of vote-buying resistance even if they protect against other types of vote-buying.

At level 1, we consider two different types of influence. In the first type, the attacker receives all receipts from the voter it wants to influence. In the second type, the attacker is physically present and looks over the voter's shoulder as they cast their vote. If the voting system protects against one of these types of attacks, it achieves level 1.

Vote-buying resistance: Level 1

At least one of the following two scenarios is prevented:

(1) The attacker can have access to the full public data. If the voter behaves honestly and provides all their receipts (if any) to an attacker, the attacker cannot be convinced that the voter voted in a certain way.

(2) The attacker can physically observe the voter while voting but does not make any further checks. The voter has a way to still vote for their preferred candidate, without the attacker being able to notice, even if that candidate is not the attacker's candidate.

If the voting system prevents both scenarios (1) and (2), even if the voter hands over their short-term voting material (see above) to the attacker, the voting system reaches level 3. Because of the special challenge of achieving this goal, we go directly from level 1 to level 3 in this case.

Vote-buying resistance: Level 3

Both scenarios from level 1 can be prevented, even when the voter provides the attacker with their short-term voting material.

To reach level 4, the voting system must also protect against vote buyers who have access to semi-public information (audit data), or who are able to successfully cast one vote on behalf of the voter (but the voter can still cast another vote that secretly nullifies the attacker's vote). If it protected against both, this reaches level 5.

Vote-buying resistance: Level 4

In addition, the attacker has at least one of the following abilities:

(1) The attacker has access to the data provided to auditors.

(2) The attacker can successfully cast a vote on behalf of the voter.

Vote-buying resistance: Level 5

The attacker has both abilities (1) and (2), as defined at level 4.

3.3 End-to-end verifiability

We present the verifiability criteria that we will use to analyze the mechanisms for verifiable online voting. We first describe the notion of verifiability which we then use to develop our criteria for assessing the level of verifiability of online voting systems.

3.3.1 Notion

Our notion of verifiability is the result of an earlier systematic review on verifiability notions for e-voting protocols [32].

Idea. Our notion of end-to-end verifiability considers whether the votes of the (human) voters that enter the system at one end correctly match the final result that is output at the other end of the system. An *end-to-end verifiable* voting system is characterized by the fact that it provides mechanisms for voters or observers to check that the data at both ends do indeed match; if they do not match, one of the checks "sounds the alarm".

This means that the purpose of verifiability is to ensure that even if one or more components of the voting system fail to perform their tasks correctly (whether intentionally or not), it is possible to detect that something has gone wrong. However, if all of these components are corrupted simultaneously and collectively, then it is difficult to provide any guarantees at all about the security of the voting system in general, and its verifiability in particular.

This central observation shows that an e-voting system can only provide verifiability under specific assumptions regarding the correctness or honesty of its participants/components. Therefore, a main conclusion of the survey on verifiability notions [32] was that it is important to state these trust assumptions explicitly. We will also follow this principle.

Definition. We represent the assumption that a participant X is honest by $\mathsf{hon}(\mathsf{X})$. This *individual trust assumption* states X cannot be corrupted and behaves as specified by the protocol. Now, we represent the *overall trust assumption* τ as a combination of "OR" and "AND" of different individual trust assumptions $\mathsf{hon}(\mathsf{X})$. For example, $\tau = \mathsf{hon}(\mathsf{VS})$ AND $\mathsf{hon}(\mathsf{T})$ describes that the voting server VS and the tallier T are assumed to be honest, i.e., cannot be corrupted.

Our definition of end-to-end verifiability is now as follows. Let τ be a trust assumption, and let $\mathcal{A}$ be a set of algorithms/attacks that an attacker can execute under the trust assumption τ. We say that a voting system is *end-to-end verifiable* under the trust assumption τ against $\mathcal{A}$ if under any attack $A \in \mathcal{A}$ (obeying the trust assumption τ), the probability that the final result published by the voting system is accepted, even though this result does not match the voters' votes, is negligible. We refer to [32] for the formal definition.

In our analysis, unless otherwise stated, we assume that all corrupted components and channels can be controlled or intercepted by the same attacker. From a security perspective, these global and active attackers represent the most pessimistic case, i.e. the greatest threat.

3.3.2 Criteria

We now present our approach to evaluate the end-to-end verifiability of a voting system.

Idea. The main idea of our evaluation is based on an explicit and precise description of the trust assumptions under which a voting system is verifiable. Based on these trust assumptions, it is then possible to make general statements about the end-to-end verifiability of the analyzed voting system, and to assess in which specific election scenarios the system should be used and in which ones it should not.

Model. To study verifiability, we refine our general model (Section 2.1) as follows:

- *Voter*: We distinguish between the *human voter*, their *voting device* VD (the device that voters use to cast their votes), and their *audit device* AD (the device that voters use to perform their individual checks).
- *Registration authority*: We consider the part of the election authority EA that is responsible for registering voters (and optionally providing them with authentication tokens) to be a separate entity, the *registration authority* RA.

Trust assumption. Based on the decomposition introduced above, we can now model any trust assumption τ as OR and AND combinations of the literals hon(RA), hon(VD), hon(AD), hon(T), and hon(VS), where hon(X) stands for the fact that the participant X is honest (see above). Since we assume that the auditor A is always is honest, we do not explicitly include the trust assumption for this party.

We note any AND/OR combination can be represented in the form

$$(\ldots \text{ AND } \ldots) \text{ OR } (\ldots \text{ AND } \ldots) \text{ OR } (\ldots \text{ AND } \ldots) \text{ OR } \ldots,$$

i.e, by an OR combination of pure AND-clauses. For example,

$$\text{hon(RA) AND } (\text{hon(VS) OR hon(T)})$$

is equivalent to

$$(\text{hon(RA) AND hon(VS)}) \text{ OR } (\text{hon(RA) AND hon(T)}).$$

We therefore focus on these AND combinations in our analysis.

Assessment. The identification of a trust assumption τ in the above form is the basis for assessing the end-to-end verifiability of the voting system under investigation. Generally speaking, the less assumptions the formula τ contains, the higher is the level of end-to-end verifiablity a system provides. However, not all assumptions have the same weight. More specifically:

- *General assessment*: First, a voting system cannot be considered verifiable if all parties need to be trusted. Therefore, if the trust assumption τ contains the term (hon(RA) AND hon(VD) AND hon(T) AND hon(VS) AND hon(AD)), then this indicates that the respective voting system is not verifiable.

  Second, of all the participants, a corrupt tallier T can most easily manipulate the complete final result if its output is not (or cannot be) verified.[2] Therefore, we expect that the trust assumption hon(T) should be avoided in any reasonable verifiable voting system, which reduces the possible combinations for such systems to seven AND combinations.

- *Specific assessment*: We also use the trust assumption τ to evaluate more precisely in which types of elections the voting system is verifiable. For example, the assumption that the registrar is honest is stronger the less information about the voters (number, identities, etc.) is made public and integrated into the voting system. This example illustrates that the same trust assumption τ does not lead to the same evaluation in two different contexts. Therefore, in our evaluation, we will consider in which contexts the respective trust assumption is justified and in which contexts it is not.

In summary, when evaluating the end-to-end verifiability of a voting system, we combine the general and specific perspectives based on the corresponding trust assumption.

[2] Moreover, there are efficient and established methods to verify the tallying of the ballots, while the situation is more difficult for the other components.

Accountability. While end-to-end verifiability guarantees the correctness of the final result, a stronger feature is desirable from a practical point of view. Namely, it is helpful to be able to identify the party or parties responsible for the error. For example, if a voting system uses a mix net (Section 4.5) consisting of multiple mix servers, and it can be observed that a particular mix server is not working correctly, then this mix server can be replaced without changing the rest of the system. In addition, potential identification serves as a deterrent against corrupt participants who fear penalties if it is discovered that they have misbehaved.

This desired property is called *accountability* [102]. It is a stronger form of verifiability, making it possible not only to determine whether the final result matches the votes cast, but also to identify the responsible component (or at least limit the possible sources of error) in the event of a detected error. Because of its practical usefulness, we will positively evaluate accountability as a bonus of end-to-end verifiable systems.

3.4 Usability

We present our usability criterion to evaluate the mechanisms that we study in this work. We first present some insights from usability evaluations of e-voting systems (i.e., systems with actual user interfaces), including what we can learn from these studies for usability criteria and evaluation at the mechanism level. We then present the usability-relevant factors we use in this work, which form the abstraction layer on which we evaluate the mechanism. Finally, we present the evaluation criterion itself, which assigns a rating to each mechanism. Our criteria focuses on mechanisms which require user involvement.

3.4.1 Background

In the following, we explain the most important aspects of usability and the factors that research on the usability of e-voting has so far identified as influencing these aspects. These findings form the basis for the definition of our criteria (Sections 3.4.2 and 3.4.3).

Usability. The ISO standard for usability [78] considers the following aspects:

- *Efficiency*: This is the (total) time that users need to complete their task. In our application, this is the time to complete authentication, vote casting, and individual verification.
- *Effectiveness*: This property describes how accurately and completely users perform their task. In our application, there are essentially two types of potential errors: failing to cast a vote, or failing to apply the individual verification mechanism. The latter can be divided into failing to detect manipulations, or failing to report a detected manipulation to the right place.
- *Satisfaction*: This property measures how comfortable users found the task. In our application, this could for example be affected by voters complaining about having to use two devices.

Related work. We base our evaluation criteria on the results of previous studies in this area (see, e.g., [121, 1, 89, 41, 98, 4, 111, 90, 113, 140, 136]). In the following, we will discuss in more detail the two studies that are most directly relevant to our criteria for our study:

- Marky et al. [110] use expert evaluation, specifically independent of a concrete user interface, to assess the usability of mechanisms in the context of remote electronic voting, focusing on the cast as intended verifiability. The capabilities we identify are inspired by this work, while we choose a more expressive structuring to evaluate our more diverse set of mechanisms. Furthermore, the authors of [110] only quantify the number of required assumptions, but do not consider differences in complexity or time required for the capabilities considered.
- Kulyk et al. apply human factors from other security domains to the context of online voting [96]. We consider the pitfalls they identified when determining the capabilities required by voters.

3.4.2 Factors

We divide usability-relevant factors into *preconditions* that must be met before the voter can use the mechanism, and the actual *steps* that the voter must take to execute the mechanism. Since we evaluate these mechanisms independent of the specific user interfaces, we stick to the technical properties: the data exchanged with the voter and the voter's steps during the exchange. Note that this excludes steps that may be very important in an actual user interface (for example, an overview of all selected candidates before really casting the vote), but are not required by the mechanism under evaluation.

Preconditions. For each mechanism, there exist certain requirements that are necessary to execute the mechanism. This includes in particular (but not exclusively) the following potential requirements:

- Remembering secret(s) (e.g. a password for the SSO service).
- Having tokens (e.g. eID card) available.
- Having additional devices available (e.g. smartcard reader, or smartphone).
- Verifying that you are connected to the correct web service (e.g. checking server certificate).
- Installing and using the correct application (e.g. vote casting app, app to verify certain steps).

In our evaluations, we do not consider the requirements, as they are not part of the mechanism itself. However, we document requirements explicitly when describing the mechanism, as they still may have an impact (e.g. maybe not all users have access to an additional device).

Steps. If the relevant preconditions are met, the voter executes the concrete steps of the mechanism. This includes in particular (but is not limited to) the following possible actions:

- Entering data (e.g. codes, tokens) manually or via QR code(s).
- Selecting options (e.g. ticking boxes next to candidates/parties).
- Comparing data (e.g. selected vote, random looking characters, images).
- Looking up data (e.g. using the browser's search function).
- Forwarding data (e.g. sending receipts to talliers via email).

Each of these steps may include aspects which make it harder for the voter to execute the step effectively (e.g. because the step is mentally demanding to perform). We consider this in our evaluation. What we unfortunately cannot evaluate on the abstraction layer aimed for by this study are aspects of accessibility; while we note that none of the evaluated mechanisms present an additional obvious hurdle.[3]

3.4.3 Criteria

We present our criteria to evaluate the usability of a voting mechanism. In general, *usability* can be decomposed into efficiency, effectiveness, and satisfaction. In this work, we do not evaluate satisfaction, since satisfaction reported in previous work was consistently high, regardless of the mechanism (see, e.g., [113]), and since this would require a real user study. Here, as we have no concrete user interfaces given, we however instead perform an expert evaluation. Therefore, we focus on efficiency and effectiveness, for which we present scoring systems.

Efficiency. To evaluate the efficiency level, we count the number of relevant steps to execute the mechanism (see Section 3.4.2). If the mechanism requires only one relevant step, it achieves the highest score, which is 5. For each additional step, we reduce this score by one.

We reduce this subscore by one for steps that require non-trivial effort per candidate, rather than per ballot (e.g., when voters need to compare a string for each candidate). This condition captures cases where a mechanism scales poorly to larger ballots.

[3] However, accessibility tools to execute some of the steps, such as readers or magnifiers, may introduce additional devices and therefore also additional trust assumptions for secrecy and verifiability.

Effectiveness. To evaluate the level of effectiveness, we count the number of all steps that are difficult, disruptive, optional, or have a large impact, as explained next:

- *Difficult*: We consider a step to be difficult if it requires actions that are challenging and not common in "everyday" applications (browsing, online banking, messaging, etc.). Such steps carry a significant risk of not being executed correctly.
- *Disruptive*: We consider a step to be disruptive if voters need to take actions at different times. For example, if voters need to be active before the actual voting and then later during the submission phase, or if the first part of the verification takes place during voting and the other part at the end of the voting period. Such breaks are likely to result in errors and low motivation to continue at a later time.
- *Optional*: We consider a (verification) step to be optional if votes can be cast without performing the verification step. The reason is that optional security steps have a negative effect on the manipulation detection rate.
- *Large impact*: We consider a step to have a large impact if an incorrect execution of that step enables the adversary to insert or change ballots (as opposed to "only" drop ballots).
- *Additional device*: Some steps need to be executed on a device different from the voting device. We consider this also to place a burden on the voter.

If a mechanism does not include any of such steps, it achieves the highest effectiveness score, which is 5. Otherwise, for each such step, we reduce the score by one.

Overall usability score. The overall usability of a mechanism is rated as the minimum of the efficiency and effectiveness subscores.

3.5 Practicality

We propose a practicality criterion to evaluate e-voting mechanisms.

We divide our criterion into two sub-criteria: implementability and efficiency, each receiving a score between 1 and 5. Implementability represents the complexity of developing an implementation for a mechanism (Section 3.5.1), while efficiency reflects the computational and communication costs of running a voting system using that mechanism (Section 3.5.2).

Overall practicality score. To combine the implementability and efficiency scores into an overall practicality score, we use a weight of 2/3 for implementability and 1/3 for efficiency. The reason is that implementation difficulties can lead to serious security problems.

3.5.1 Implementability

Implementability captures the effort required to implement a mechanism. We do not give a precise definition of the concept here, but propose a reasonably objective grading scale.

We begin by listing the various aspects that affect implementability, and follow with a grading scale to quantify the impact of each aspect. We classify the aspects of implementability into *constraints*, which imply complexity of the mechanism and its implementation, and *enablers*, which facilitate the implementation.

To evaluate the overall implementability of a mechanism, we start with 5, subtract the points corresponding to constraints, and add the points for enablers. The resulting sum is trimmed to fit in the range $[1, 5]$.

Constraint: skills (Table 3.1). A mechanism may require the person or group of people implementing it to have some knowledge or skill in a particular domain. We consider these to be any skills, in addition to the "standard" background of a software engineer, that are required to understand the particular algorithm or to implement it correctly.

Examples of such skills include cryptography (e.g. elliptic curves, zero-knowledge proofs), systems (e.g. mailing system, virtualization), parallelism (e.g. MPI, POSIX threads), secure hardware (e.g. JavaCard programming), Blockchain (e.g. Smart Contract programming), or web security (e.g. CSRF, CSP).

Table 3.1: Evaluation of required skills.

Score	Description
0	Only common software engineering knowledge is needed
-1	Advanced knowledge in one topic needed
-2	Advanced knowledge in multiple topics is needed

Constraint: resources (Table 3.2). A mechanism may rely on existing resources, physical or digital, to enable its implementation, making the task more complex.

Examples of such resources include sources of randomness in a constrained environment, low-latency/high-throughput network between agents, key management systems (e.g. HSM), multiple interacting devices, anonymous/untappable channels, special printing (e.g., special paper, unusual folding, scratch cards), or personal trusted hardware (e.g. eID).

Constraint: protocol complexity (Table 3.3). The number of agents involved in the mechanism, as well as the pattern of communication between the agents, has a significant impact on the effort required to implement the mechanism: more agents means that more independent pieces of

Table 3.2: Evaluation of required resources.

Score	Description
0	No specific resource is needed
-1	Some reasonably obtainable resource is needed
-2	Some hard to obtain or setup or manageable resource is needed

software must be developed, and the communication between the agents must be managed and synchronized. Therefore, the more agents are involved and the more complex their communication is, the more negatively we evaluate this constraint. If the mechanism requires up to two more agents, we consider this *few* agents, otherwise we consider this *many* agents.

Table 3.3: Evaluation of protocol complexity.

Score	Description
0	Few agents
-1	Many agents with independent actions
-2	Many agents, with multiple dependent interactions

Enabler: software library (Table 3.4). If a library exists that implements the mechanism, the implementability is greatly increased, possibly to the point where the question becomes moot. Since the answer to the question of whether a library exists is rarely a simple yes or no, we use the following categories to evaluate the quality of a library:

- *High*: Actively maintained, well documented, and has a permissive license. The library covers exactly the mechanism.
- *Medium*: Passively maintained, lacking some documentation, or with a restrictive license. The library only partially covers the mechanism, or some relevant part is hard to extract.
- *Low*: Unmaintained or uncommon language, poorly documented and difficult to use.

Table 3.4: Evaluation of enablers.

Score	Description
0	No library or reference implementation, the algorithm is only described in white papers or research papers
1	Low quality or prototype implementation for part or for the entire mechanism, but not usable as-is
2	High quality library for an important building block, or medium quality library for the entire mechanism
3	One or more high quality libraries for the entire mechanism, well specified and documented

3.5.2 Efficiency

We study two aspects of efficiency: communication and computation. We evaluate the communication overhead and the computation overhead of a mechanism separately, and the minimum of both is its overall efficiency.

In our evaluations, to estimate the data size and computation time, we will consider an election with 100 candidates from which a voter can choose one.

Communication (Table 3.5). The main communication costs of a mechanism are the size of the data to be transferred and the complexity of the communication:

- *Data size*: We look at the total size of all messages and divide it by the number of voters. We use 1 MB per voter as a definition of medium size.
- *Complexity*: We count the number of communication rounds, i.e. the number of times an agent (server, voting device, audit device, ...) has to wait for data from others before it can send messages again. The higher the number of rounds, the worse the communication complexity. If the number of rounds is in relation to the number of servers and trustees, we use *some* rounds, else we use *many* rounds.

Table 3.5: Evaluation of communication overhead.

Score	Description
1	many rounds and big size
2	some rounds and big size, or many rounds and medium size
3	some rounds and medium size
4	1 round and medium size
5	0-1 round and small size

Computation (Table 3.6). To evaluate the computational overhead of a mechanism, we consider a large scale election with 100,000 voters on current retail hardware. We distinguish between the computational cost on the voters' side and on the servers' side.

- *Voter*: We assume that the voter is using a browser or app on their personal computer or phone. We distinguish between less than 1 second, less than 1 minute, and more than 1 minute of computation time.
- *Server*: We distinguish between a few minutes, a few hours, and many hours (more than five) of computation time on a single processor. We value positively when the algorithm can be distributed or parallelized.

Table 3.6: Evaluation of computational cost.

Score	Time Server	Time Client
1	$\geq$ few hours and not distributable and not parallelizable	$>$ 1min
2	$\geq$ few hours, distributable or parallelizable	$>$ 1min
3	$\leq$ few hours	$\leq$ 1min
4	$\leq$ few hours, distributable or parallelizable	$\leq$ 1s
5	$\leq$ few minutes	$\leq$ 1s

4 Evaluation

This chapter represents the main part of our work: the description and our evaluation of different methods for verifiable online voting.

Before we study these techniques in Section 4.2 to Section 4.9, we start with an overview of our evaluation. Please read this introduction carefully before proceeding with the evaluation.

4.1 Overview

From the results of our market and literature analysis, we selected eight different techniques that are used to implement verifiable online voting. These eight methods are the cryptographic building blocks that we consider most relevant for the practical implementation of verifiable online voting. In addition to these methods, there are of course other useful building blocks, which we do not discuss in detail in this paper, but which we briefly describe in Section 4.10.

Selected methods. The methods that we will evaluate in this paper can be categorized as follows (recall Section 2.1.3):

- **Secret ballots**: We study two different methods for keeping individual ballots secret, namely **malleable public-key encryption** (Section 4.2) and **malleable commitments** (Section 4.3), which are both compatible with the following verifiable tallying techniques.
- **Verifiable tallying**: We study two different methods to verifiably tally ballots in a privacy-preserving way. These are **homomorphic aggregation** (Section 4.4) and **verifiable mixing** (Section 4.5).
- **Authenticated communication**: We study how digital signatures (Section 4.6) can be used in verifiable online voting to resolve possible disputes and to distribute trust in the authentication of ballots.
- **Voting device verification**: We study three different methods that enable individual human voters to verify that their voting devices correctly processed their secret votes. These are **audit-or-cast** (Section 4.7), **cast-and-audit** (Section 4.8), and **return codes** (Section 4.9). All of these techniques are compatible with the two secret ballot techniques (see above). Depending on how they are implemented, these techniques protect against potentially corrupted voting applications or even potentially corrupted voting machines.

Evaluation. We study these eight techniques with our criteria for vote secrecy (Section 3.2), verifiability (Section 3.3), usability (Section 3.4), and practicality (Section 3.5) that we defined in Chapter 3.

Relationship to subproperties. End-to-end verifiability of e-voting systems is sometimes divided into three subtypes: *individual verifiability* (voters can verify that the votes they cast are included in the tally), *universal verifiability* (it can be independently verified that the tally is correct), and *eligibility verifiability* (it can be verified that only eligible voters participate in the election).

Although this separation contains subtle pitfalls and end-to-end verifiability cannot be easily derived from these sub-properties (see below), we would like to illustrate their relationship to the selected methods:

- *Individual verifiability*: Essentially, individual verifiability allows voters to check that (1) their voting device has processed their vote correctly and (2) their processed vote is included in the tally. The first property is called *cast-as-intended*; the methods audit-or-cast (Section 4.7), cast-and-audit (Section 4.8), and return codes (Section 4.9) aim to ensure this property. The second property, called *recorded-as-cast*, ensures that cast ballots are stored and remain stored on the bulletin board. Recorded-as-cast is usually achieved by voters verifying that their ballots have been correctly stored by the voting server or posted to the bulletin board (depending on the system). Since this check is independent of the mechanisms that we study in this work, we assume in the following that honest voters make this check.

- *Universal verifiability*: Universal verifiability guarantees that the secret votes in the digital ballot box are tallied correctly. Therefore, this property is sometimes called *tallied-as-recorded*. This property can be achieved by combining one of the two secret ballot methods (malleable PKE and malleable commitments, see Sections 4.2 and 4.3) with one of the two verifiable tallying methods (homomorphic aggregation and verifiable mixing, see Sections 4.4 and 4.5).

- *Eligibility verifiability*: How to verify or ensure that only eligible voters cast a vote is primarily a feature of the voting *system*, based on effective authentication of voters. Nevertheless, eligibility verifiability can be improved using cryptography, for example by integrating digital signatures to distribute the authentication of voters to parties independent of the system.

However, a word of caution at this point. Even if a voting system provides all three of these properties, this does not guarantee the end-to-end verifiability of the system. For example, it is necessary to ensure that the *data is consistent* throughout the course of an election and that so-called *clash attacks* [103] are prevented. Therefore, when analyzing the security of an online voting system, it is crucial to consider the entire system and not to use any shortcuts!

Limitations. We want to be clear about the limitations of our work:

1. *Focus*: In this study, we focus on the cryptographic methods of verifiable online voting systems. In addition to these core components, there are many other aspects that are important for realizing verifiable online elections in practice. These include, but are not limited to, general IT security considerations for complex and distributed systems, the establishment of an information security management system (ISMS) and the development of an appropriate user interface (UI).

2. *Scope*: We do not study all existing important cryptographic building blocks. In particular, we always assume in our evaluation that the bulletin board is implemented securely, i.e., that it provides a consistent, unbiased view of the audit material to all authorized participants. For completeness, we briefly describe other relevant methods that we have identified in our market and literature analysis in Section 4.10.

3. *Formal rigidity*: The intended audience of this paper is newcomers to the field of verifiable online voting, and this paper is intended to give them a first, well-founded overview of the various methods available in this area. Therefore, we have deliberately avoided a rigid mathematical description and evaluation in favor of accessibility. For formal analysis, we will refer to corresponding work in the academic literature.

Conventions. For ease of presentation and evaluation, the following conventions are used throughout this chapter:

- *Bulletin board*: While it is important to ensure that the bulletin board is implemented correctly (see above), the methods we study in this paper are essentially independent of this system component. Therefore, in this study we assume (as stated above) that the bulletin board is working correctly. We also assume that the voting server hosts the bulletin board and is implemented correctly. In Section 4.10.1 we will describe approaches to reduce the trust in the bulletin board independently from the cryptographic components and thus improve the overall security.
- *Ballot authentication*: Most of the techniques we will present and evaluate in this paper are essentially independent of how ballots are authenticated. The only exception is Section 4.6, where we come to the use of digital signatures to make authentication more independent of the voting system. Therefore, in all other sections we always assume that the registration authority RA is honest and correctly authenticates the voters to the voting server VS.
- *Voting and audit devices*: Since it is not directly relevant to the cryptographic methods, in this study we will always consider each device as the entirety of the personal system that the voter uses. Therefore, we will not distinguish between the actual machine, the operating system, and the applications.

Software libraries. In our practicality analysis, we will refer to software libraries that implement the (building blocks of the) different mechanisms that we study in this work. More specifically, we use the following high-quality and well-maintained libraries we are aware of: CHVote, ElectionGuard, Verificatum, Belenios, and Swiss Post. For convenience, we typically refer to only one specific implementation of a given cryptographic building block in one of these libraries because the alternative implementations (if any) in the other libraries are usually equivalent (but may be written in different languages). Our specific references do not imply that we consider one of these implementations to be better or worse than the other.

4.2 Malleable public-key encryption

In verifiable online vote, using *malleable* public-key encryption (PKE) is the most common method for voters to cast their votes in such a way that the integrity of their ballots can be verified throughout the election process without compromising the privacy of their individual votes. At a high level, the basic encryption property (Section 2.2.1) guarantees the secrecy of individual votes, while the malleability of these ciphertexts is used to process the encrypted votes in a verifiable privacy-preserving manner.

For this to work, the PKE scheme must provide certain features, which we first explain along with a common implementation that we consider in this work (Section 4.2.1). We then describe how malleable public-key encryption is used in verifiable online elections for secret voting (Section 4.2.2). Finally, we apply our evaluation criteria to analyze the mechanism (Section 4.2.3).

4.2.1 Requirements

We specify the features that malleable public-key encryption schemes need to satisfy in addition to the basic ones (Section 2.2.1) when used for verifiable online voting. These features are malleability, proof of plaintext knowledge, verifiable key generation, verifiable decryption, and distributed decryption. We also specify an implementation that realizes these features and that we analyze.

Malleability. The malleability of PKE schemes is the foundation of many advanced cryptographic features that are used for verifiable e-voting. Depending on how the encrypted ballots are tallied, one of the following forms of malleability is required:

- *Additively homomorphic* (for *homomorphic aggregation*, see Section 4.4): In an *additively homomorphic* PKE scheme $(\mathsf{KG}_e, \mathsf{Enc}, \mathsf{Dec})$, there exists a deterministic polynomial-time algorithm Hom that takes as input ciphertexts $e_i = \mathsf{Enc}(pk, m_i)$ $(i \in I)$ and returns a ciphertext e, such that $\mathsf{Dec}(sk, e) = \sum_{i \in I} m_i$. It is important to note that Hom does not take any m_i or sk as input.[1]

  See Figure 4.1 for an illustration.

- *Re-randomizable* (for *verifiable mix nets*, see Section 4.5): In a *re-randomizable* PKE scheme $(\mathsf{KG}_e, \mathsf{Enc}, \mathsf{Dec})$, there exists a probabilistic polynomial-time algorithm ReRand that takes as input a public key pk and an arbitrary ciphertext e that decrypts to some message m with the corresponding secret key sk, and returns a new ciphertext e' that decrypts to the same message m with the secret key sk and such that the result of ReRand is computationally indistinguishable from that of Enc. It is important to note that ReRand does not take m or sk as input.

  See Figure 4.2 for an illustration.

[1] In addition, there are other types of homomorphism that differ in terms of which operations they are homomorphic. The most important types are *multiplicatively homomorphic* (homomorphic with respect to the multiplication of messages), *leveled homomorphic* (homomorphic with respect to arbitrary functions/circuits with restricted depth), and *fully homomorphic* (homomorphic with respect to arbitrarily complex functions/circuits). Such types of homomorphism can be useful for the design of verifiable online voting systems with particularly strong secrecy properties (see Section 4.10).

Remark 2: Malleability warning notice

The malleability of PKE schemes is a double-edged sword, because it can also be exploited by an attacker to undermine both the vote privacy or the verifiability of a voting system if it is not used in a targeted and localized manner:

- *Vote privacy*: It is necessary that voters cast their votes independently [50]. For example, *replay attacks* are a prominent class of attacks that exploit the missing independence of ballots. In a replay attack, an attacker submits a targeted voter's vote on behalf of some corrupted voters. In this way, the targeted voter's choice is amplified in the final result, which undermines their vote privacy (see [115] for an efficiency analysis of replay attacks).

  The simplest type of replay attack is to copy the targeted voter's ballot. Therefore, ballots that match an already accepted ballot must not be accepted. However, there are more subtle types of replay attacks that cannot be prevented in this way and require other measures. In fact, an attacker can exploit the malleability of the PKE scheme to re-randomize the target voter's encrypted ballot.[a] To protect against such attacks, we always combine the "raw" manipulable public-key encryption scheme with a so-called *proof of knowledge* (see below).

  With this proof, each voting device must prove that it also knows the secret vote cast, without revealing which vote it is to preserve the secrecy of the vote. However, an attacker does not know which vote was chosen by the targeted honest voter, because that is precisely its goal. Thus, the attacker cannot generate their own valid proof of knowledge of the targeted voter's secret vote. To prevent the attacker from copying the proof of knowledge of the targeted voter, no ballots are accepted that contain a proof of knowledge that is already contained in another accepted ballot.

- *Verifiability*: When the ballots are tallied homomorphically (Section 4.4), then it must be guaranteed that the encrypted message in fact belongs to the set of valid choices.

  In fact, if in this case no further protection is in place, then a single corrupt voter could determine the final result as they wish, without this manipulation being detected. Instead of a correct ballot, such a voter could encrypt a message that adds an arbitrary number of votes to the desired candidate and simultaneously subtracts those votes from the undesired ones. For example, in an election with three choices A, B, C a corrupt voter could submit the fake ballot $(10, -5, -5)$ to add ten votes to choice A, stolen equally from B and C.

  Therefore, we always combine the "raw" homomorphic public-key encryption scheme with a proof of set membership (see below), when we tally the ballots homomorphically (Section 4.4).[b]

This demonstrates that great care must be taken to ensure that malleable PKE schemes are malleable only for their intended purpose and not otherwise.

[a] In addition to replay attacks, there are other ways to exploit the malleability of a PKE scheme to undermine vote privacy. See for example [117].

[b] Note that when we tally them with a verifiable mix net (Section 4.5), we do not need such a proof of set membership. The reason is that each voter's individual input appears in the (anonymized) final result, so invalid votes can be removed directly and do not change the result.

Proof of plaintext knowledge. To ensure that the voters cast their votes independently, and thus to guarantee vote privacy (see Remark 2), each voter must prove that they know their secret vote.

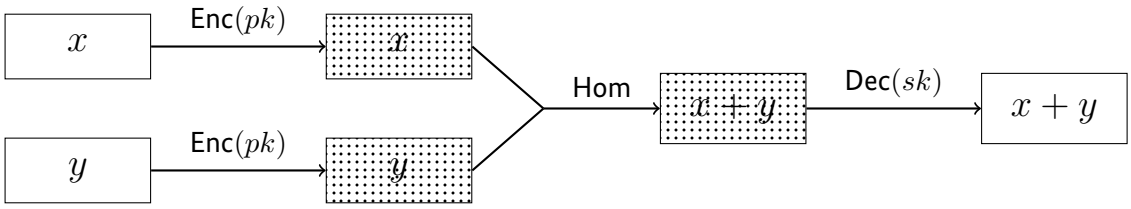

Figure 4.1: Illustration of an additively homomorphic public-key encryption scheme (Section 4.2.1) with public/private key pair (pk, sk).

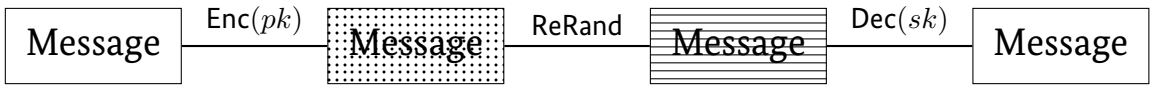

Figure 4.2: Illustration of a re-randomizable public-key encryption scheme (Section 4.2.1) with public/private key pair (pk, sk).

For this purpose, we use a proof of plaintext knowledge to prove knowledge of a message m in a given public-key encrypted ciphertext $e = \mathsf{Enc}(pk, m)$, without revealing any other information (in particular, no information about the message m).

Proof of set membership (for homomorphic aggregation, Section 4.4). To ensure that only valid votes are homomorphically aggregated, and thus to guarantee verifiability (see Remark 2), each voter must prove that the encrypted message belongs to the set of valid choices. For this purpose, we use a proof of set membership to prove that a message m in a given public-key encrypted ciphertext $e = \mathsf{Enc}(pk, m)$ belongs to a certain set $\mathcal{C}$, without revealing any other information (in particular, no information about the message m).

Verifiable key generation. For a voting system to be verifiable, it is important to ensure that the public key pk has been generated correctly. For example, if the public key pk is malformed, it can be exploited by malicious voters to create malformed ballots, or by malicious talliers to covertly manipulate ballots, even if all other security mechanisms are correctly integrated. Furthermore, if the public key pk is computed in a distributed fashion to distribute trust, but the well-formedness of pk is not guaranteed, then a single malicious tallier could exploit this flaw to subvert its contribution to the computation of pk, allowing that single tallier to learn how individual voters voted.

To protect against these threats, the party or parties (in the case of distributed decryption/key generation) computing the public key must provide independently verifiable proof that the output pk is a valid output of the key generation algorithm KG_e. To protect the secrecy of the PKE scheme and thus the privacy of the voting system, this proof must not reveal any information (in particular, about the secret key or the secret key shares) other than the correctness of the public key.

Verifiable decryption. At the end of the two privacy-preserving tallying methods that we analyze in this study (Sections 4.4 and 4.5), the processed ciphertexts are decrypted. This step is performed internally by the tallier, since its private decryption key is used, which must remain secret. In order to verify that the decrypted plaintexts (which represent the final result) match the messages in the ciphertexts, the tallier must provide a proof of correct decryption. This property is known as *verifiable decryption*. As with verifiable key generation, such a proof should not reveal any information about the secret key sk.

Distributed decryption. If a PKE is used as described above, then the tallier who owns the secret key sk can decrypt all individual votes and thus learns how each individual voter has voted. To prevent this, the secret key sk should be generated and distributed among multiple talliers in a way that requires a certain number of talliers to work together to decrypt a ciphertext that was computed with the talliers' joint public key pk. This property is called *distributed decryption*.

There are two flavors of distributed decryption: *full secret sharing* and *threshold secret sharing* (Section 2.2.5). In the first case, the secret key is shared in such a way that *all* m talliers (the share holders) must contribute to decrypt a ciphertext. In the second case, only a threshold $t < m$ of the talliers are required to cooperate; threshold secret sharing improves the robustness of the scheme, for example against loss of secret key shares or unavailable talliers.

Implementation. In the context of verifiable online voting, ElGamal's PKE [47] scheme is the most common implementation in today's systems. Therefore, we study this implementation.

Remark 3: ElGamal PKE

In this excursion, we describe ElGamal's original PKE scheme. This description is intended for readers with a mathematical background who want to understand how malleable public-key encryption can be technically realized. Since this excursion is not necessary for understanding our study, feel free to skip this part.

- The key generation algorithm KG_e chooses a cyclic group $\mathcal{G}$ of (large) prime order p in which the Decisional Diffie-Hellman (DDH) assumption is assumed to hold true. Next, this algorithm chooses a random generator g of $\mathcal{G}$, and a uniformly random number s from $\mathbb{Z}_p$. The resulting public key pk consists of the tuple $(\mathcal{G}, p, g, h = g^s)$, while the secret key sk is $(\mathcal{G}, p, g, s)$.
- The encryption algorithm Enc takes as input a public key $pk = (\mathcal{G}, p, g, h = g^s)$ and a message $m \in \mathcal{G}$. It then chooses a uniformly random r from $\mathbb{Z}_q$, and returns as ciphertext e the tuple $(u = g^r, v = h^r \cdot m) \in \mathcal{G}^2$. The purpose of h^r is to mask the message m, while $u = g^r$ can be used to unmask it again with the help of the secret s, as described next.
- The decryption algorithm Dec takes as input a secret key $sk = (\mathcal{G}, p, g, s)$ and a ciphertext $e = (u, v)$. It recovers the encrypted message m by "stripping of" the mask by computing $v \cdot u^{-s}$; in fact, for $(u, v) = (g^r, h^r \cdot m)$ and $h = g^s$, we have $v \cdot u^{-s} = g^{sr} \cdot g^{-sr} \cdot m = m$.

In the *exponential version* of ElGamal's scheme, ciphertexts are of the form $(u = g^r, v = h^r \cdot g^m)$, where $m \in \mathbb{Z}_q$ is the message. In this version, ciphertexts are decrypted as $\log_g(v \cdot u^{-s})$, which can be done efficiently when the set of possible messages is relatively small.

ElGamal's scheme is (IND-CPA) secure under the so-called *Decisional Diffie-Hellman (DDH)* assumption, which is generally believed to be intractable on the (classical) computers we use today.[2] ElGamal's original scheme is multiplicatively homomorphic (in the underlying group $\mathcal{G}$) and re-randomizable. Often, an exponential version of the scheme is used, which is additively homomorphic (in the underlying group $\mathbb{Z}_q$) and also re-randomizable. We refer to Remark 3 for the mathematical background.

We study the following implementations of the features that are required in addition to the malleability property:

[2] There are algorithms for quantum computers that can efficiently solve the DDH assumption and thus break the secrecy of ElGamal's scheme. While no real quantum computer exists today that can compute these algorithms, the development of more powerful quantum computers poses a risk to public-key cryptography currently in use, including for online voting. See Section 4.3 for more details.

- *Proof of knowledge*: We study the use of the non-interactive zero-knowledge proof (NIZKP) (Section 2.2.4) of plaintext knowledge for ElGamal ciphertext described, for example, in [58].
- *Proof of set membership (for homomorphic aggregation)*: We consider the implementation of the proof of set membership by a NIZKP of knowledge for the corresponding relation. Specifically, we consider the implementation in [58] for the case of ElGamal encrypted votes. This NIZKP of set membership consists of NIZKPs of membership in $\{0, 1\}$ for each ciphertext per candidate and a NIZKP of membership in $\{0, 1\}$ for the homomorphically aggregated candidate ciphertext; from the correctness of these proofs it follows that the vector representation of the ballot is binary with exactly one 1 entry.

  Since these proofs of set membership imply knowledge of the plaintext, we do not need a separate proof of knowledge in this case.
- *Verifiable key generation*: We study the use of a NIZKP of knowledge (Section 2.2.4) to let the tallier prove that it knows the secret sk corresponding to a public key pk. Specifically, we choose the common implementation based on the non-interactive pre-image proof in Schnorr's identification scheme [130] (see also Section 5.4 in [58]). This implementation is described, for example, in the CHVote specification (see Algorithm 8.7 in [58]).
- *Verifiable decryption*: We study the use of a NIZKP of knowledge to let the tallier prove that it used the secret key sk to decrypt an ElGamal ciphertext e under the public key pk to a message m. Specifically, we choose again the common implementation based on Schnorr's identification scheme, which is also described, for example, in the CHVote specification (see Example 3 in Section 5.4 in [58]).
- *Distributed key generation/decryption*: We study the use of full and threshold secret sharing to distribute the ElGamal key generation and decryption among different talliers such that m or $t < m$ of the talliers need to collaborate to decrypt a ciphertext c that was encrypted under their joint public key pk. The algorithms that are run by individual talliers are combined with the NIZKP for verifiable key generation and verifiable decryption (see above). For a full secret sharing, see, for example, Protocol 7.1 (verifiable distributed key generation) and Protocol 7.8 (verifiable distributed decryption) in [58]. For verifiable threshold secret sharing, see [53, 31].

There are different ways to realize the underlying algebraic group of the ElGamal PKE scheme and these NIZKPs. For our evaluation, we study the implementations that use the elliptic curve *Curve25519* to instantiate this group.

4.2.2 Description

We now describe how a PKE scheme with the above properties is used in verifiable online voting.

In the setup phase, the tallier T runs the key generation algorithm KG_e to obtain a public/secret key pair pk, sk; in case of distributed decryption, the key generation algorithm KG_e is run in a distributed fashion by the different talliers $\mathsf{T}_1, \ldots, \mathsf{T}_m$ so that each tallier T_j obtains its secret key share sk_j. The tallier(s) also provide(s) a proof for the correctness of the public key. After that, the public key pk and its proof are posted to the bulletin board BB.

In the ballot submission phase, each voter V_i enters their choice v_i to the voting device VD, which encrypts v_i under pk to obtain the secret ballot e_i and computes a corresponding proof of knowledge π_i of v_i. Depending on the tallying method, the voters compute and append more data to e_i, as we explain in Sections 4.4 and 4.5. Afterwards, the voting device sends (e_i, π_i) (and the additional data, if any) to the voting server VS. The voting server stores (e_i, π_i) on BB if neither e_i nor π_i appear in an already accepted ballot. Here, we abstract away from the way the voter/voting device authenticates to VS and assume that the registration authority RA correctly established such an authenticated channel (recall our conventions in Section 4.1).

Remark 4: Identifiable vs anonymous ballots

Depending on the voting protocol, the encrypted ballots of the voters on the bulletin board are individually linked to the voters (e.g., connected with the voters' IDs) or anonymous (without any identifiable information). In the following, we discuss the main advantages and disadvantages of these two approaches.

Anonymous ballots:

- *Everlasting privacy*: Since the ballots do not contain any information about the identities of the associated voters, anonymous ballots can be used to achieve vote privacy without the use of cryptography, in particular independent of any hardness assumptions, and therefore provide everlasting privacy.
- *Participation privacy*: For the same reason as before, anonymous ballots can be used to keep secret which voters participated in the election and which did not.
- *Verifiability*: The main disadvantage of anonymous ballots is their lower level of transparency. First, the eligibility of the votes cast cannot be independently verified, which can for example be exploited to stuff the voting server with illegitimate ballots. Second, it opens up the possibility of so-called *clash attacks* (Remark 8) with which corrupted voting devices can secretly alter voters' choices.

Identifiable ballots:

- *Everlasting privacy*: Only special cryptographic techniques can guarantee that the voting system will provide everlasting privacy. This is possible, for example, with commitments (see Section 4.3). In contrast to anonymous ballots, therefore, for everlasting privacy one must trust that the cryptographic procedures employed are correct.
- *Participation privacy*: Again, only special cryptographic methods can guarantee that the voting system provides privacy. This is possible, for example, with the methods from [118, 65, 109], which we will not discuss further in this study.
- *Verifiability*: Since the (encrypted) ballots are assigned to individual voters, the eligibility of the ballots submitted can be checked immediately and independently, and clash attacks can be prevented.

In short, anonymous ballots are an easy way to achieve strong privacy properties, but they come at the cost of a lower level of transparency than identifiable ballots, which require the use of advanced cryptographic methods to provide strong privacy properties.

There are two ways the talliers can process the encrypted votes. In the simple case, they directly decrypt the encrypted votes with their secret key shares, then compute the final result, and finally post it to BB. In the other case, the talliers first process the encrypted votes in a privacy-preserving way, and only then decrypt the processed votes to compute and share the final result; we will study the two most common approaches to privacy-preserving tallying in Sections 4.4 and 4.5.

4.2.3 Analysis

We now evaluate the level of secrecy, verifiability, usability, and practicality of malleable public-key encryption according to our criteria defined in Chapter 3.

Secrecy. We analyze the secrecy of the mechanism based on our criteria introduced in Section 3.2. Here we will focus on the simple case of direct decryption and computation of the final result. In Sections 4.4.3 and 4.5.3, we will study cases where the votes are first processed in a privacy-preserving way.

Vote privacy. In the simple case (without privacy-preserving tallying), the voting scheme achieves **vote privacy level 1**, but does not achieve vote privacy level 2 according to our privacy criteria (Section 3.2.1). The reason why level 1 is achieved is that the voters' choices are transmitted in encrypted manner over the network, but the attacker does not know the corresponding secret key sk. However, level 2 is not achieved because the votes are decrypted without previous privacy-preserving processing, which is why every single tallier learns the individual links between the voters and their votes. Therefore, if the adversary can control one tallier (as assumed for level 2), then the adversary would also be able to break vote privacy.

We will show that in combination with homomorphic aggregation (Section 4.4) or with verifiable shuffling (Section 4.5), higher levels of vote privacy can be guaranteed.

Everlasting privacy. This mechanism provides **everlasting privacy level 0** because it does not provide a reasonable level of verifiability. In fact, as we observe below, we need to trust all components of the voting system for the correctness of the final result.

Vote-buying resistance. This mechanism does not protect against vote-buying because an attacker could learn how the voter voted by observing the voter during the ballot submission (as assumed for vote-buying resistance level 1). Therefore, this mechanism provides **vote-buying resistance level 0**.

We note that a simple option to protect against the first type of *Level 1* attackers is re-voting, which would allow voters to overwrite their previously cast ballots. To protect against any stronger vote-buying attacks, more complicated mechanisms must be integrated, which we do not study in detail in this work, but which are mentioned in Section 4.10.

Table 4.1: Secrecy score summary.

Aspect	**Score**
Vote Privacy	1
Everlasting Privacy	0
Vote Buying Resistance	0
Secrecy	2

Verifiability. We analyze the verifiability of the mechanism based on our criteria introduced in Section 3.3.

Recall from Section 3.3 that the purpose of end-to-end verifiability is to reduce or even remove trust in the system components regarding the correctness of election result. Using solely malleable public-key encryption does not offer any advantages in this respect because we still have to trust all components.

Table 4.2: Verifiability trust assumptions.

hon(RA) AND hon(T) AND hon(VD) AND hon(VS)

This shows that malleable PKE must always be used in combination with a verifiable privacy-preserving mechanism, such as homomorphic aggregation (Section 4.4) or verifiable mixing (Section 4.5) mechanisms. We evaluate the combined mechanisms in Sections 4.4.3 and 4.5.3.

Usability. We analyze the usability of the mechanism based on our criteria from Section 3.4.

The voter's procedure consists only of entering the preferred candidate into the voting device. The voting device then performs the encryption and any other necessary computation by itself. We evaluate the procedure as follows:

- *Efficiency*: The voter performs one step, namely entering the chosen candidate. This corresponds to an efficiency score of 5.
- *Effectiveness*: The single step has none of the characteristics which would reduce its effectiveness. Therefore, the effectiveness score is 5.

Table 4.3: Usability score summary.

Aspect	**Score**
Efficiency	5
Effectiveness	5
Usability	$\min(\cdot,\cdot) = 5$

Practicality. We analyze the practicality of the mechanism based on our criteria introduced in Section 3.5.

We analyze the implementation of mallable public-key encryption including the required proofs. For the proofs, we separate between *no threshold* secret sharing, and *threshold* secret sharing, when the characteristics are different according to our evaluation criteria. Further, we consider differences in implementation when used in a verifiable mix net (see Section 4.5) or homomorphic aggregation (see Section 4.4).

We assume the implementation to employ elliptic curve cryptography, using the standard Curve25519 [106]. The curve has good characteristics to enable efficient and secure implementations [18]. Group elements and scalars each have a size of 256bit, i.e. 32B. On the voter's side, based on benchmarks from Belenios done in Firefox on a 2019 laptop with a 1.9GHz CPU, a single modular exponentiation takes about 0.32ms. On the server's side, based in benchmarks from the Bernstein comparison[3], a modular exponentiation takes about 0.11ms.[4] Compared to modular exponentiations, modular multiplications are less computationally demanding, but for simplicity (and as it does not make much of a difference in our scenario), we use the same estimation for both.

We evaluate the implementability constraints (Section 3.5.1) as follows:

- *Skills*: Knowledge in cryptography is necessary to implement and integrate the mechanism correctly. We deduce one point.
- *Resources*: No specific resources are needed for the mechanism.

[3] See `https://bench.cr.yp.to/impl-scalarmult/curve25519.html`. We choose the most performant implementation running on an AMD Ryzen 7 7700 processor at 3.8 GHz. The implementation needs $421'580$ cycles, which corresponds to $421'580 \div (3.8 \times 10^9) \approx 11 \times 10^{-5} = 0.11$ms.

[4] This is faster because the implementation takes advantage of hardware-specific characteristics. As the system provider typically knows the exact hardware used in the server, the provider may choose an implementation that has been adapted accordingly.

- *Protocol complexity*: Due to the distribution, there are many agents involved. With *no threshold* secret sharing, the actions of the agents phases are essentially independent, leading us to deduce one point. With *threshold* secret sharing, the talliers' setup and tallying phases include more involved interactions, as illustrated in Protocol 7.1 and 7.8 of [58], leading us to deduce two points.

We evaluate the implementability enablers (Section 3.5.1) as follows:

- *Software library*: There exist several well-documented, ready-to-deploy libraries that implement malleable PKE, the proof of plaintext knowledge, and the proof of set membership (for homomorphic aggregation) as needed for the mechanism:
  - Verificatum/VJSC: Javascript, MIT license, actively maintained, library providing full functionality to encrypt a ballot with ElGamal;
  - ElectionGuard: C++/Rust, MIT license, in development, library with various functionalities;
  - CHVote: Java, AGPLv3, actively maintained, cryptographic core of a voting system;
  - Belenios: OCaml, AGPLv3, actively maintained, voting system;
  - SwissPost's system: Java, proprietary license, actively maintained, voting system.

  We therefore grant three points.

We evaluate the communication efficiency (Section 3.5.2) as follows. Both approaches have data size *small*. Concerning the number of blocking rounds, *no threshold* needs 0 rounds, while *threshold* needs *some* rounds. This results in an overall communication efficiency score of 5 for *no threshold*, and 3 for *threshold*. We achieve their intermediate scores as follows:

- *Data size*: First, we note that a single ElGamal ciphertext consists of two group elements.

  When we tally the ballots with a verifiable mix net (Section 4.5), we can encrypt the complete choice in a single ciphertext. The corresponding proof of plaintext knowledge is done with 2 scalars (see Section 5.4.1 in [58]). The ciphtertext together with the proof results therefore in $2 \cdot 32\mathsf{B} + 2 \cdot 32\mathsf{B} = 128\mathsf{B}$ per ballot.

  When we tally the ballots with homomorphic aggregation (Section 4.4), we consider the approach of representing each choice as a binary vector, where the 1 entries indicate a voter's preferred candidates. Proving membership in the set $\{0, 1\}$ consists of 4 scalars: 2 challenges and 2 responses for each element in the set (see, e.g., Section 5.4 of [58]). A membership proof is needed for each candidate as well as for the homomorphic aggregation of the candidate entries, which corresponds to $100 + 1 = 101$ proofs for an election with 100 candidates. When we put these numbers together, a ballot consists of 200 group elements for the encryptions and $101 \cdot 8 = 808$ scalars for the NIZKPs. This results in $200 \cdot 32\mathsf{B} + 808 \cdot 32\mathsf{B} \approx 32\mathsf{KB}$.

  This shows that in both cases, the size of the ballot is well below our $1\mathsf{MB}$ threshold, hence the data is *small*.

- *Complexity*: With *no threshold* secret sharing, no blocking communication is necessary. With *threshold* secret sharing, the protocol includes blocking steps impacting the communication efficiency. As described in Section 3.1.1 of [53] and with more details in [31], 3 rounds per tallier are required to build partial decryption keys. Assuming the usage of homomorphic aggregation (see Section 4.4), the talliers perform the decryption in 2 rounds, which results in a total of 5 rounds per tallier.

  For *no threshold* secret sharing, we therefore have 0 blocking rounds, while for *threshold* secret sharing, we have *some* blocking rounds.

We evaluate the computational efficiency (Section 3.5.2) as follows. For both approaches, the server time is less than a few minutes, and the voter time less than a second, hence both have an overall computational efficiency score of 5. We achieve the intermediate scores as follows:

- *Server time*: Only the voting server is active in the ballot submission phase and it only needs to store (or forward) the incoming ballots.[5] The time required for these operations is less than a few minutes in total.
- *Voter time*: First, we note that forming a single ciphertext requires two exponentiations.

  When we tally the ballots with a verifiable mix net (Section 4.5), a single ciphertext is sufficient. The corresponding proof of plaintext knowledge requires another 2 exponentiations. Hence, the total time to compute the ballot is therefore $4 \cdot 0.32\text{ms} = 1.28\text{ms}$.

  When we tally the ballots homomorphically (Section 4.4), we again consider the approach which results in one ciphertext per candidate (see the *Data size* evaluation for details). Each membership proof requires 4 exponentiations (see, e.g., Section 5.4 of [58]: a proof of plaintext knowledge requires 2 exponentiations, thus a membership proof for a set with 2 elements requires $2 \cdot 2 = 4$). In an election with 100 candidates where a voter can select at most one of them, a ballot therefore needs $200 \cdot 0.32\text{ms} = 64\text{ms}$ to encrypt, and then $404 \cdot 0.32\text{ms} = 129.3\text{ms}$ to generate the NIZKP. The total ballot construction time remains clearly under 1s.

Table 4.4: Practicality score summary.

Aspect	*no threshold*	*threshold*
Skills	-1	-1
Resources	0	0
Protocol complexity	-1	-2
Software library	+3	+3
Implementability (I)	$\text{trim}(5 + \text{sum}(\cdot)) = 5$	5
Communication	5	3
Computation	5	5
Efficiency (E)	$\min(\cdot, \cdot) = 5$	3
Practicality	$\frac{2}{3}I + \frac{1}{3}E = 5$	4.33

[5] We evaluate the server time to verify the proofs as part of the verifiable mix net (see Section 4.5), respectively the homomorphic aggregation (see Section 4.4).

4.3 Malleable commitments

An interesting alternative to the use of public-key encryption for public secrecy of votes is the use of commitment schemes. Since commitment schemes are keyless, their secrecy can be unconditional in contrast to public-key encryption schemes (see Section 2.2.2). Unconditionally hiding commitments therefore represent a way to provide practical everlasting privacy without compromising verifiability (recall Remark 4).

At a high level, the hiding property of the commitment scheme (Section 2.2.2) guarantees the secrecy of individual votes, while the malleability of these commitments is used to process the committed votes in a verifiably privacy-preserving manner. For this to work, the commitment scheme must provide certain features, which we first explain along with a common implementation that we consider in this work (Section 4.2.1). We then describe how malleable commitments are used in verifiable online elections for secret voting (Section 4.2.2). Finally, we apply our evaluation criteria to analyze the mechanism (Section 4.2.3).

4.3.1 Requirements

We specify the features that malleable commitment schemes need to satisfy in addition to the basic ones (Section 2.2.2) when used for verifiable online voting. These features are malleability, proof of plaintext knowledge, and threshold secret sharing of committed messages. We also specify the implementation of this building block that we analyze in our work.

Malleability. As with public-key encryption schemes (see Section 4.2), the malleability of commitment schemes (CS) is the basis for many advanced cryptographic features that are used for verifiable online voting. Depending on how the committed ballots are tallied, one of the following forms of malleability is required:

- *Additively homomorphic* (for *homomorphic aggregation*, see Section 4.4): In an *additively homomorphic* CS $\mathcal{C} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$, there exists a deterministic polynomial-time algorithm Hom that takes the parameters prm and commitments $(c_i)_{i\in I}$ to messages $(m_i)_{i\in I}$ as input and returns a commitment c that can be opened to $\sum_{i\in I} m_i$ with opening value $d = \sum_{i\in I} d_i$, where $(d_i)_{i\in I}$ are the corresponding individual secret opening values. It is important to note that Hom does not take any message m_i or any opening value d_i as input.

  See Figure 4.3 for an illustration.

- *Re-randomizable* (for *verifiable mixing*, see Section 4.5): In a *re-randomizable* CS, there exists a probabilistic polynomial-time algorithm ReRand that takes as input the parameters prm and an arbitrary commitment c that can be opened to some message m with opening value d, and returns a new commitment c' that can be opened to the same message m with a different opening value d' and such that the distributions of c and c' are indistinguishable. It is important to note that ReRand does not take m or the opening value d of c as input.

  See Figure 4.4 for an illustration.

Remark 5: Malleability warning notice

Just as with malleable PKE schemes (Remark 2), malleable commitment schemes must be combined with NIZKPs so that they are malleable only for their intended purpose and not otherwise.

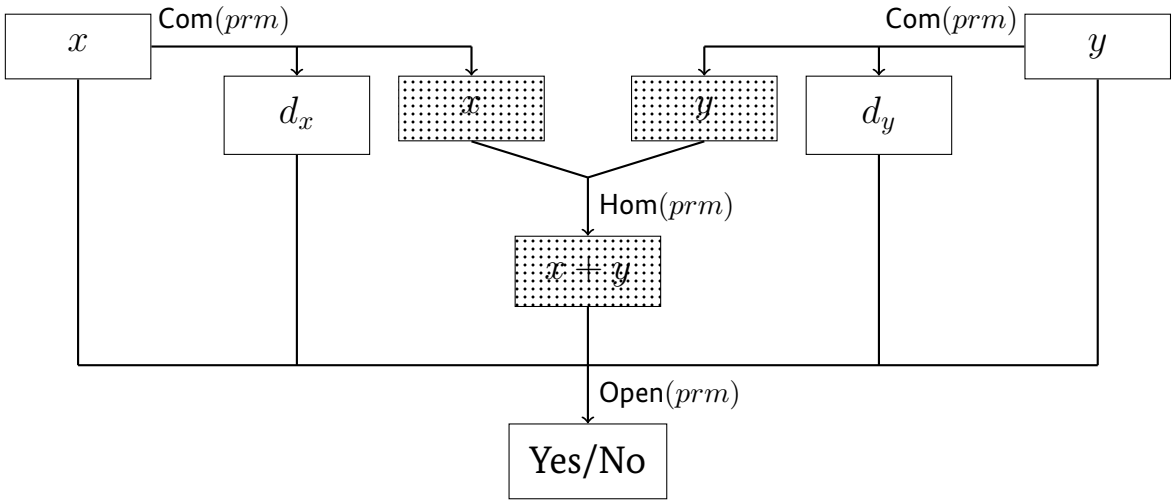

Figure 4.3: Illustration of an additively homomorphic commitment scheme (Section 4.3.1).

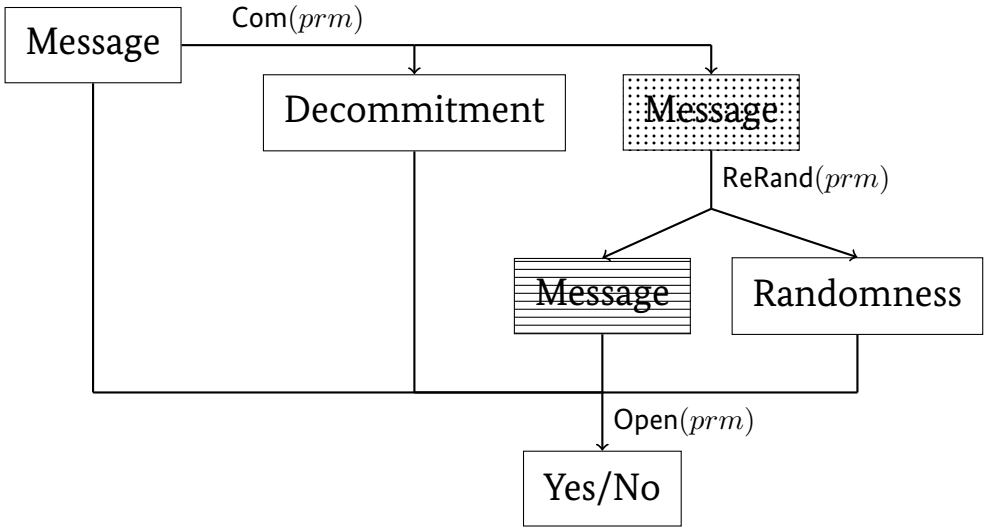

Figure 4.4: Illustration of a re-randomizable commitment scheme (Section 4.3.1).

Proof of plaintext knowledge. To ensure that the voters cast their votes independently, and thus to guarantee vote privacy (see Remark 2), each voter must prove that they know their secret vote. For this purpose, we use a proof of plaintext knowledge to prove knowledge of a message m in a given commitment $c = \mathsf{Com}(prm, m)$, without revealing any other information (in particular, no information about the message m).

Proof of set membership (for homomorphic aggregation, Section 4.4). To ensure that only valid votes are homomorphically aggregated, and thus to guarantee verifiability (see Remark 2), each voter must prove that the committed vote belongs to the set of valid choices. For this purpose, we use a proof of set membership to prove that a message m in a given commitment $c = \mathsf{Com}(prm, m)$ belongs to a certain set $\mathcal{C}$, without revealing any other information (in particular, no information about the message m).

Threshold secret sharing. Just as in the case of PKE (Section 4.2), it can be useful to distribute trust among the talliers so that threshold of them need to collaborate to open a committed message. This

technique can be used when committed ballots are homomorphically aggregated (Section 4.4).

Implementation: Pedersen commitments. To implement the commitment scheme, we choose Pedersen's scheme [124], which is unconditionally hiding, computationally binding under the Discrete Logarithm (DL) assumption, and both additively homorphic and re-randomizable. This is the most popular (unconditionally hiding) commitment scheme due to its simplicity, efficiency and well-studied security. See, for example, [58] for an implementation of Pedersen's commitment scheme.

We study the following implementations of the features that are required in addition to the malleability property:

- *Proof of knowledge*: We study the use of the non-interactive zero-knowledge proof (NIZKP) (Section 2.2.4) of plaintext knowledge for Pedersen commitments described, e.g., in [48].

- *Proof of set membership (for homomorphic aggregation)*: We consider the implementation of the proof of set membership by a NIZKP of knowledge for the corresponding relation. Specifically, we consider the implementation in [48] for the case of Pedersen commitments. This NIZKP of set membership consists of NIZKPs of membership in $\{0, 1\}$ for each commitment per candidate and a NIZKP of membership in $\{0, 1\}$ for the homomorphically aggregated candidate commitment; from the correctness of these proofs it follows that the vector representation of the ballot is binary with exactly one 1 entry.

  Since these NIZKPs are proofs of knowledge, we do not need a separate proof of knowledge.

- *Threshold secret sharing*: To secretly share messages in Pedersen commitments, we choose Shamir's threshold secret sharing scheme [132], which can be used to divide the secret among n receivers in such a way that at least $k \leq n$ (arbitrary) receivers must combine their shares to reconstruct the message (Section 2.2.5).

Remark 6: Pedersen's commitment scheme

In this excursion, we describe Pedersen's commitment scheme. This description is intended for readers with a mathematical background who want to understand how malleable commitments can be technically realized. Since this excursion is not necessary for understanding our study, feel free to skip this part.

- To generate the public parameters, the setup algorithm Setup chooses a cyclic group $\mathcal{G}$ of order q in which the DL assumption is assumed to hold true. It then chooses two generators g and h of $\mathcal{G}$ independently uniformly at random. The resulting public parameters are $prm = (\mathcal{G}, p, q, g, h)$.

- To commit to a message $m \in \mathbb{Z}_q$ for public parameters $prm = (\mathcal{G}, p, q, g, h)$, the commitment algorithm Com chooses a uniformly random r from $\mathbb{Z}_q$ and returns the commitment/opening pair $(c = g^m \cdot h^r, d = (m, r))$. The (public) commitment c then hides the message m, while the (private) opening value d can be used to verify/prove that c is a commitment to m.

- To open a commitment $c \in \mathcal{G}$ with opening value $d = (m, r) \in \mathbb{Z}_q^2$, the opening algorithm Open returns 1 if $c = g^m \cdot h^r$ and otherwise returns 0.

4.3.2 Description

We explain how a commitment scheme with the above features is used in verifiable online voting.

In the setup phase, the election authority EA runs the algorithm Setup of the commitment scheme to obtain the public parameters prm and then post prm to the bulletin board BB.[6]

In the ballot submission phase, the voting device VD commits to the voter's choice v_i under prm to obtain the commitment/opening pair (c_i, d_i) and computes a corresponding proof of knowledge π_i of v_i. Depending on the tallying method, the voting device computes and appends more data to c_i (see Sections 4.4 and 4.5). Afterwards, the voting device VD sends (c_i, π_i) (and the additional data, if any) to the voting server VS. The voting server stores (c_i, π_i) on BB if neither c_i nor π_i appear in an already accepted ballot. Moreover, VD encrypts d_i under the tallier's public key pk and sends the resulting ciphertext to them (for example via the voting server VS). Here, as in Section 4.3.2, we abstract away from the way the voter/voting device authenticates to VS and assume that the registration authority RA correctly established such an authenticated channel (recall our conventions in Section 4.1).

If the ballots are tallied homomorphically (Section 4.4), then the trust in T can be distributed among different talliers $\mathsf{T}_1, \ldots, \mathsf{T}_m$ as follows. First, the voting device uses the secret sharing scheme to secretly share the voter's vote v_i among the talliers. Then, the voter performs the casting process described before for each share v_i^j of their vote and the corresponding tallier T_j. If the ballots are tallied with a mix net, then there exist different techniques to distribute the trust without dividing the commitments into different shares (see Section 4.5 for details).

In the tallying phase, the talliers process the committed votes in a privacy-preserving way to compute the election result (see Sections 4.4 and 4.5).

4.3.3 Analysis

Secrecy. We evaluate the three subproperties of vote secrecy.

Vote privacy. Using only malleable commitments achieves **vote privacy level 1** of our evaluation criteria (Section 3.2.1) for the same reasons that using only malleable public-key encryption achieves level 1 (Section 4.2.3).

This observation shows that this method (just like malleable public-key encryption) must always be combined with a privacy-preserving tallying mechanism to provide a reasonable level of vote privacy (see Sections 4.4.3 and 4.5.3), vote privacy can be guaranteed.

Everlasting privacy. This mechanism provides **everlasting privacy level 0** because it does not provide a reasonable level of verifiability. In fact, as we observe below, we need to trust all components of the voting system for the correctness of the final result.

Vote-buying resistance. This mechanism does not protect against vote-buying because an attacker could learn how the voter voted by observing the voter during the ballot submission (as assumed for vote-buying resistance level 1). Therefore, this mechanism provides **vote-buying resistance level 0**.

[6] It is important that this step be done honestly, because the correctness of prm is the basis of all security and privacy features. Failure to adhere to this principle can lead to serious problems [61].

Table 4.5: Secrecy score summary.

Aspect	Score
Vote Privacy	1
Everlasting Privacy	0
Vote Buying Resistance	0
Secrecy	2

Verifiability. Relying solely on a malleable commitment scheme is not sufficient to make the system verifiable for the same reasons that relying solely on a malleable public-key encryption is not sufficient either.

Table 4.6: Verifiability trust assumptions.

$\mathsf{hon}(\mathsf{RA})$ AND $\mathsf{hon}(\mathsf{T})$ AND $\mathsf{hon}(\mathsf{VD})$ AND $\mathsf{hon}(\mathsf{VS})$

This means that malleable commitments must always be used in combination with a verifiable privacy-preserving mechanism, such as homomorphic aggregation (Section 4.4) or verifiable mixing (Section 4.5) mechanisms. We evaluate the combined mechanisms in Sections 4.4.3 and 4.5.3.

Usability. We get the same result as for malleable public-key encryption (Section 4.2.3) with the same reasoning.

Table 4.7: Usability score summary.

Aspect	Score
Efficiency	5
Effectiveness	5
Usability	$\min(\cdot,\cdot) = 5$

Practicality. The evaluation from the point of view of practicality is very similar to malleable PKE (Section 4.2.3). We mention here only the aspects which differ:

- *Implementability enablers*: No software library which provides commitments as a building block as required in our context was found, which results in a score of 0 for this aspect.[7].
- *Data size*: As we use commitments instead of encryptions, the data size changes. Each commitment consists out of one group element and the encryption of an opening value $d = (m, r)$ of 4 group elements. A NIZKP of membership in $\{0, 1\}$ for a Pedersen commitment consists of 2 group elements and 4 scalars, as described, e.g., in Appendix B.2 of [48].

  When we tally the ballots with homomorphic aggregation (Section 4.4), in an election with 100 candidates where a voter can select at most one of them, a ballot consists therefore of $(1+4)*100 = 500$ values for the commitments/openings and $(2+4)*(100+1) = 606$ values for the NIZKPs. This results in $500 * 32\mathsf{B} + 606 * 32\mathsf{B} \leq 35\mathsf{KB}$.

  This remains well below our $1\mathsf{MB}$ threshold, hence the data is still considered to be *small*.

[7] Some prototype implementations of systems are taken into account when evaluating homomorphic aggregation and verifiable mixnets in Sections 4.4 and 4.5

- *Voter time*: Similarly to data size, the use of commitments changes the time to build a ballot. Each Pedersen commitment requires 2 exponentiations and the ElGamal encryption of the corresponding opening requires 4 exponentiations. The membership proof for a commitment requires 4 exponentiations (see Appendix B.2 of [48]).

  When we tally the ballots with homomorphic aggregation (Section 4.4), in an election with 100 candidates where a voter can select at most one of them, the construction of a ballot requires $(2 + 4) * 100 = 600$ exponentiations for the commitments/openings and $4 * (100 + 1) = 404$ for the membership proofs. Using the same time estimation for an exponentiation as in Section 4.2.3, i.e. 0.32ms, this results in $(600 + 404) * 0.32\text{ms} = 321\text{ms}$.

Table 4.8: Practicality score summary.

Aspect	*no threshold*	*threshold*
Skills	-1	-1
Resources	0	0
Protocol complexity	-1	-2
Software library	0	0
Implementability (I)	$\text{trim}(5 + \text{sum}(\cdot)) = 3$	2
Communication	5	3
Computation	5	5
Efficiency (E)	$\text{min}(\cdot, \cdot) = 5$	3
Practicality	$\frac{2}{3}I + \frac{1}{3}E = 3.67$	2.33

4.4 Homomorphic aggregation

Many verifiable voting systems use the malleable structures of the cryptographic algorithms to encrypt or commit to the votes to combine these ciphertexts or commitments, respectively, in such a way that the votes they contain are added together.

The concept of homomorphic aggregation to tally ballots in a verifiable manner is based on the additively homomorphic property of the underlying public-key encryption or commitment scheme. As we explained in Section 4.2 (for public-key encryption) and in Section 4.3 (for commitments), this feature guarantees that a list of such ciphertexts or a list of such commitments can be combined in a way that the result is a single ciphertext or a single commitment that contains the sum of all messages in the respective list; in particular, this function is deterministic and can be performed by anyone without any secret knowledge (e.g., of a secret key or opening values).

Idea (Figure 4.5). Let us now describe how this property is used in verifiable online voting to tally ballots in a verifiable privacy-preserving way. We illustrate the idea for the case of encrypted ballots (Section 4.2); the case of committed ballots (Section 4.3) is analogous.

The basic idea of homomorphic aggregation is that each voting device encrypts the number 1 if the voter supports the corresponding candidate, and the number 0 otherwise. After each voting device has encrypted its input vote v_i under Enc to $e_i = \mathsf{Enc}(pk, v_i)$ and cast it, the homomorphic function Hom of the public-key encryption scheme is used to implicitly add up the secret votes $v_1, \ldots, v_n$ of all voters without having to decrypt the corresponding ciphertexts $e_1, \ldots, e_n$; in short, $e = \mathsf{Hom}((e_i)_{i=1}^n) = \mathsf{Enc}(pk, \sum_{i=1}^n v_i)$.

This summation breaks the links between the individual voters and their votes and thus keeps them secret. Since the Hom operation is deterministic, any observer can independently verify the correctness of this step. Depending on the voting system, the aggregated ciphertext is then decrypted or used for other purposes.

Remark 7: Malleability warning notice

We already warned the reader in Remark 2 that exploiting homomorphic (or more generally: malleable) properties of cryptographic algorithms is a double-edged sword, as it can also be exploited by potential adversaries.

In fact, if we use the homomorphic aggregation as sketched above, a corrupted voter V_i could secretly submit more than one vote (by choosing some $v > 1$) or remove votes to manipulate the election outcome (see Remark 2 for an example).

To ensure that the homomorphic property is used only for its intended purpose, we must use a proof of set membership, as specified in Sections 4.2.1 and 4.3.1. In this way, voters prove that their encrypted votes v are from the set of valid choices.[a]

[a] Since these proofs of set membership are usually also proofs of knowledge, we do not need a separate proof of knowledge to ensure ballot independence (Remark 2).

4.4.1 Requirements

We do not need any cryptographic building blocks other than those specified in Section 4.2 (homomorphic PKE) or Section 4.3 (homomorphic commitments).

4.4.2 Description

We explain how to extend the secret ballot mechanism for encrypted ballots (Section 4.2.2) or for committed ballots (Section 4.3.2) to securely aggregate ballots homomorphically.

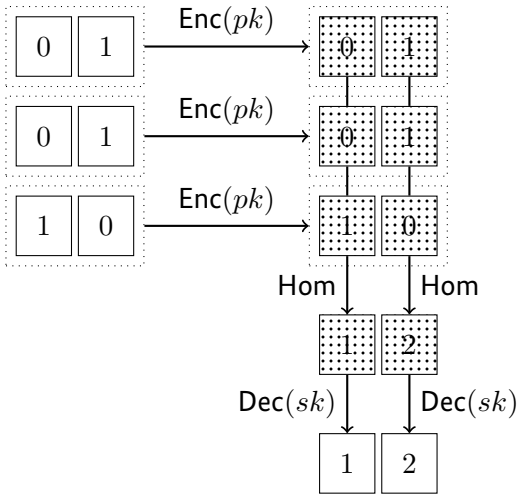

Figure 4.5: Illustration of a homomorphic aggregation of public-key encrypted votes (Section 4.4.2), with three voters (rows) and two candidates (columns).

There are two main differences between the homomorphic aggregation of public-key ciphertexts (see above) and commitments. First, in the case of public-key encryption, trust is distributed by sharing the secret key among the talliers, while in the case of commitments, trust is distributed by sharing the individual votes among the talliers. Second, in the case of public-key encryption, the talliers use their secret key (shares) to decrypt the aggregated ballots, while in the case of commitments, they use the opening values they received directly from the voters.

Encrypted ballots. In addition to the steps taken in the submission phase described in Section 4.2.2, the voting device of each voter V_i also computes a proof of set membership π_i for the secret vote v_i. The voting device appends the proof π_i to the ciphertext e_i and posts the tuple (e_i, π_i) to BB.

After the end of the submission phase, the talliers verify for each (e_i, π_i) whether π_i is valid w.r.t. e_i and whether neither e_i nor π_i are contained in any previous ballot; the first check is to guarantee the correctness of the secret vote, and the second check is to ensure that the ballots are cast independently of each other (recall Remark 2). If any of these conditions do not hold, the ballot is discarded, and otherwise accepted for the following tallying phase. Note that this preselection can be performed and thus verified by anyone with access to BB.

In the tallying phase, the preselected encrypted votes are first homomorphically aggregated; note that this step can also be performed and thus be verified by anyone with access to BB. Afterwards, the talliers check whether the preselection and the homomorphic aggregation were correct, and then secretly decrypt the aggregated result, compute a proof of correct decryption, and post the resulting plaintexts together with their proofs of decryption to BB.

To verify the correctness of the election result, an observer with access to BB verifies all proofs on BB (i.e., the tallier's proof of correct key generation, the voters' proof of set membership, and the tallier's proof of correct decryption), the correctness of the preselection, the correctness of the homomorphic aggregation, and that the result is equal to the combination of the partial decryptions. If all checks pass, the election result is accepted, otherwise it is rejected.

Committed ballots. First, the voting device secretly shares the voter's secret vote v_i among the talliers $\mathsf{T}_1, \ldots, \mathsf{T}_m$ into shares $v_i^1, \ldots, v_i^m$ and then commits to each share as $(c_i^j, d_i^j) \leftarrow \mathsf{Com}(prm, v_i^j)$. Afterwards, the voting device computes a proof of set membership π_i to show that the sum of all committed shares $c_i \leftarrow \sum_{j=1}^{m} c_i^j$ can be opened to an element in the set of valid choices. The vot-

ing device adds the proof π_i to the commitments $(c_i^j)_{j=1}^m$ and posts the tuple $((c_i^j)_{j=1}^m, \pi_i)$ to BB. In parallel, the voting device encrypts the opening value d_i^j under the public key of the corresponding tallier T_j and sends the encrypted value to T_j (for example via the voting server VS, which forwards it to T_j).

After the end of the submission phase, the talliers verify for each (c_i, π_i) whether π_i is valid w.r.t. c_i and whether neither c_i nor π_i are contained in any previous ballot; the first check is to guarantee the correctness of the secret vote, and the second check is to ensure that the ballots are cast independently of each other (see Remark 2). If any of these conditions does not hold, the ballot is discarded, and otherwise accepted for the following tallying phase. Note that this preselection can be performed and thus verified by anyone with access to BB.

In the tallying phase, the preselected committed votes are first homomorphically aggregated; note that this step can also be performed and thus be verified by anyone with access to BB. Afterwards, each tallier T_j privately aggregates its shares $d_1^j, \ldots, d_n^j$ of the secret opening values that it obtained from the voters, and posts the resulting overall opening value $d^j \leftarrow \sum_{i=1}^n d_i^j$ to BB. The final result can then be computed by opening the aggregated commitments with the aggregated opening values of all talliers.

To verify the correctness of the election result, an observer with access to BB verifies all proofs on BB (i.e., the voters' proof of set membership), the correctness of the preselection, the correctness of the homomorphic aggregation, and the correctness of the final opening. If all checks pass, the election result is accepted, otherwise it is rejected.

4.4.3 Analysis

Secrecy. We analyze the secrecy aspects (vote privacy, everlasting privacy, vote-buying resistance) of the combination *malleable PKE (Section 4.2) & homomorphic aggregation* and of the combination *malleable commitments (Section 4.3) & homomorphic aggregation*. We summarize our results in Table 4.9.

Vote privacy. We distinguish between the following two scenarios, both for the use of malleable PKE and for the use of malleable commitments. In the first scenario, there is only one tallier. In the second scenario, there are multiple talliers and they generate their joint public key distributively (in the case of malleable PKE) or voters secretly distribute their votes among the talliers (in the case of malleable commitments).

1. *Single tallier*: In this case, the voting system protects against **Level 1** attackers, but not against *Level 2* attackers. The reason is that a *Level 2* attacker can compromise the decryption machine and thus learn the decryption key to decrypt or open any ballot on the bulletin board BB.

2. *Several talliers*: In this case, the voting system protects against **Level 4** attackers, but not against *Level 5* attackers. Indeed, the attacker cannot compromise enough decryption talliers to be able to decrypt or open commitments. However, because voting device VD learns the voter's plain vote, the mechanism does not protect against *Level 5* attackers.

We would like to make the following remarks about possible shortcuts that reduce the vote privacy level:

- Implementing a distributed key generation (DKG) is challenging (see Section 4.2.3). Therefore, in practice many systems run a "fake" DKG protocol on a *single* machine. The shares are then distributed to each tallier to give a semblance of security. This approach degrades the evaluation to **Level 3** since the attacker can learn the decryption key once the machine used during the setup can be compromised.

- In many practical voting systems, the talliers do not decrypt on their own machines (mainly to control the access to the voting material). Instead, partial decryptions are all computed on a single machine on which each tallier uses its share of the decryption key. This degrades the evaluation to **Level 3**. Note that if the data received by the tally machine link voters to their ballot, then the evaluation decreases to **Level 1** since the tally machine can associate a plain vote to a voter.

Everlasting privacy. We distinguish between the following two cases: identifiable ballots and anonymous ballots (recall Remark 4).

1. *Identifiable ballots*: In this case, malleable public-key encryption combined with homomorphic aggregation does not provide practical everlasting privacy. The reason is that all encrypted ballots are posted to the bulletin board, and thus anyone with access to the bulletin board will also learn the conditionally secret ballots.

   Malleable commitments can however provide practical everlasting privacy even when the voters' commitments (ballots) are identifiable. In fact, if we use unconditionally hiding commitments such as Pedersen commitments, if all NIZKPs are unconditionally zero-knowledge (like in the implementation we consider), and if the voting server forwards (and then deletes) the encrypted opening values directly to the talliers, then the system protects against **Level 3** attackers because all data on the bulletin board keeps the individual votes unconditionally secret.

2. *Anonymous ballots*: In this case, the ballots are information-theoretically independent of the voters' identities. Therefore, this approach provides everlasting privacy. However, this technique cannot be combined with digital signatures to reduce trust in the registration authority (Section 4.6).

Vote-buying resistance. This mechanism combined with any of the secret ballot mechanisms does not protect against vote-buying, i.e., achieves only **level 0**, but not level 1. The reason is the same as described in Sections 4.2.3 and 4.3.3.

Table 4.9: Summary of secrecy evaluation. The overall score of one of the compositions here is calculated by adding together the doubled score of vote privacy, the score of everlasting privacy, and the score of vote-buying resistance (see Section 3.2).

Aspect	*with mall. PKE*	*with mall. commitment*
Vote privacy		
One tallier	1	1
Several talliers	4	4
Everlasting privacy		
Identifiable ballots	0	3
Anonymous ballots	3	3
Vote-buying resistance	0	0

Verifiability. We study the verifiability of homomorphic aggregation combined with malleable PKE or malleable commitments. To this end, we analyze under which trust assumptions the correctness of the final result can be verified.

The voters' NIZKP of set membership on BB ensures that each voter can submit only valid votes. The homomorphic aggregation of the votes is deterministic and can be executed by anyone with

access to the bulletin board BB. If public-key encryption is used, the talliers' NIZKP of correct decryption on BB ensures that the final result matches the homomorphically aggregated votes. If commitments are used, the binding property of the commitment scheme ensures that the final result matches the homomorphically aggregated votes. In combination, these mechanisms guarantee that the final result correctly matches the encrypted votes on BB, even if the talliers are corrupted.

We do, however, need to make three trust assumptions in this evaluation:

1. We assume that the voting server VS, which hosts the bulletin board BB, is honest; mitigating this assumption is usually independent of the actual voting system (see Section 4.10).
2. We assume that the voting devices of (honest) voters are honest. Otherwise, without further means the voters do not have any guarantee that their voting devices processed their votes correctly. The challenge to protect against possibly corrupted voting devices is addressed in Sections 4.7 to 4.9.
3. We assume that the voters were registered correctly by the corresponding authority RA. This assumption can, for example, be mitigated with the use of digital signatures (Section 4.6), however only if ballots are identifiable.

These assumptions are summarized in the following table. Note that we do not need to trust the tallier T.

Table 4.10: Verifiability trust assumptions.

hon(RA) AND hon(VD) AND hon(VS)

Usability. The mechanism does not affect usability of the voter casting their vote.

Practicality. We evaluate the implementability constraints (Section 3.5.1) as follows:

- *Skills*: Knowledge in cryptography is necessary to implement and integrate the mechanism correctly. We deduce one point.
- *Resources*: No specific resources are needed for the mechanism.
- *Protocol complexity*: The mechanism does not significantly increase the complexity of the underlying voting system.

We evaluate the implementability enablers (Section 3.5.1) as follows:

- *Software library*: When homomorphic aggregation is combined with malleable PKE, the libraries listed in Section 4.2.3 can be used. We grant three points.

  When homomorphic aggregation is combined with malleable commitments, to our knowledge the only public implementation is the prototype of the Koinonia voting system [49][8]. We grant one point.

We evaluate the communication efficiency (Section 3.5.2) as follows:

- *Data size*: The data size is small (see Sections 4.2.3 and 4.3.3).
- *Complexity*: The mechanism does not add communication complexity to the system.

[8] https://github.com/gehuangyi20/Koinonia, accessed on 2024-05-13

We evaluate the computational efficiency (Section 3.5.2) as follows:

- *Server time*: As part of the mechanism, the NIZKPs of ballot validity are verified, the ballots are aggregated and the result is then verifiably decrypted.

  Verifying a proof of set membership in $\{0, 1\}$ for an ElGamal ciphertext requires 8 exponentiations (see 5.4.1 and 5.4.2 in [58]), corresponding to 808 exponentiations per ballot for 100 candidates with one selected candidate. For an election with 100'000 voters, this results in $8.08 \cdot 10^7$ exponentiations. To aggregate the ballots, they need to be multiplied together, resulting in $100'000 \cdot 200 = 2 \times 10^7$ multiplications. Then, the aggregation is decrypted, and a corresponding proof of correct decryption is generated. This requires one exponentiation for the decryption, as well as two exponentiations for the proof. Each tallier performs all of these steps, possibly in parallel.

  We reuse the same time estimation as in Section 4.2.3, i.e. 0.11ms. This results in $(8.08 \times 10^7) \cdot 0.11\text{ms} \approx 8'888\text{s} \approx$ **2:28 h** for verifying ballots, $(2 \times 10^7) \cdot 0.11\text{ms} \approx 2200\text{s} \approx$ **0:37 h** for aggregation and $3 \cdot 0.11\text{ms} =$ **0.33 ms** for verifiable decryption, amounting to a total of **3:05 h**.

  As the algorithm, notably the verification, is highly parallelizable, we grant four points.

- *Voter time*: There are no computations to be performed on the voter's side.

Table 4.11: Practicality score summary.

Aspect	*with mall. PKE*	*with mall. commitment*
Skills	-1	-1
Resources	0	0
Protocol complexity	0	0
Software library	+3	+1
Implementability (I)	$\text{trim}(5 + \text{sum}(\cdot)) = 5$	5
Communication	5	5
Computation	4	4
Efficiency (E)	$\min(\cdot, \cdot) = 4$	4
Practicality	$\frac{2}{3}I + \frac{1}{3}E = 4.67$	4.67

4.5 Verifiable mixing networks

Verifiable mixing networks (*mix nets*) are a common technique to tally secret ballots in verifiable online voting. In this section, we describe how to verifiably mix encrypted or committed secret votes (Sections 4.2 and 4.3) and evaluate the properties of this approach.

There are two types of mix nets: *decryption mix nets* (originally proposed in [27], see Section 4.10.2) and *re-randomization mix nets* (originally proposed in [123]). In this study, we focus on re-randomization mix nets, which work as follows (for public-key encrypted messages).

Re-randomization mix nets (Figure 4.6). Mix nets are distributed protocols for anonymizing sensitive data. A mix net is run between a set of senders $\mathsf{S}_1, \ldots, \mathsf{S}_m$ (in our case the voters) and a set of independent mix servers $\mathsf{M}_1, \ldots, \mathsf{M}_n$ (in our case the talliers). The senders and the mix servers know the public key pk of a IND-CPA-secure public key encryption scheme (as in Section 4.2), whose secret key sk is secretly shared among some designated talliers (e.g., the mix servers themselves). The public key encryption scheme must be re-randomizable, which guarantees that it is possible to derive from any ciphertext $e = \mathsf{Enc}(pk, m)$ a computationally indistinguishable ciphertext $e' = \mathsf{Enc}(pk, m)$ without knowing the corresponding secret key sk (see the definition in Section 4.2.1).

Each sender S_i encrypts its plaintext input message m_i with the public key pk and posts the ciphertext to BB. The first mix server M_1 re-randomizes all input ciphertexts, shuffles the re-randomized ciphertexts uniformly at random, and passes the resulting ciphertexts to the second mix server M_2. The second mix server re-randomizes these ciphertexts again, shuffles them, and so on. Finally, the last mix server M_n outputs a list of ciphertexts that encrypt the input messages originally chosen by the senders, but under different random coins and in a random order. Afterwards, the result can be decrypted by the talliers who hold the secret key shares.

If at least one mix server M_k is honest and does not reveal the permutation and the random coins it applied, a mix net guarantees that the individual links between senders and their messages remain secret.

Verifiability technique. If misbehaving mix servers are a threat, then the simple mix nets sketched above must be extended. In fact, for applications such as secure online voting, the mix net used must also be verifiable to ensure that it can be detected if something goes wrong (e.g., the final result does not match the submitted ballots). Many different methods have been developed to make mix nets verifiable [62]. The most popular and common technique to transform a re-randomization mix net into a verifiable one is to have each mix server prove in zero-knowledge that it correctly re-randomized and shuffled its input ciphertexts (see Section 4.10.2 for more techniques for verifiable mix nets). Using such *proofs of shuffle* is also the approach that we describe and analyze in the following.

4.5.1 Requirements

We now explain which additional cryptographic building blocks are required to securely shuffle encrypted ballots (Section 4.2) or committed ballots (Section 4.3). The building blocks of the corresponding secret ballot approaches are described in Sections 4.2.1 and 4.3.1, respectively.

Proof of shuffle. As described above, we consider the approach that each mix server proves with a proof of shuffle that the output ciphertext or commitment vector contains the same messages as the vector it received. This proof of shuffle must not reveal any further information, such as the random coins used for re-randomization or the private permutation.

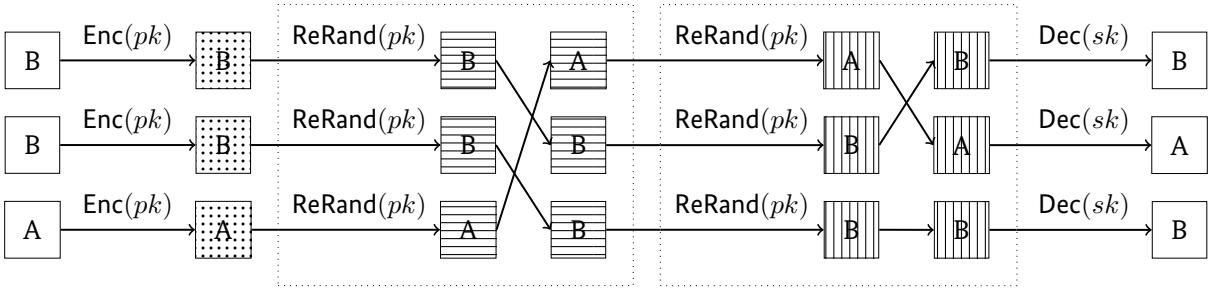

Figure 4.6: Illustration of a re-randomization mix net for public-key encrypted votes (Section 4.4.2), with three voters (rows) and two mix servers (dotted boxes: re-randomization and shuffle).

Implementation. We consider the following implementations of this approach in our analysis.

- *Public-key encryption*: We use the NIZKP of shuffle originally presented in [135] that was later implemented in *Verificatum* [139]. We note that in [59, 60] a machine-checked verifier for this proof of shuffle was proposed.

  A notable alternative to this proof of shuffle is the one proposed in [15], which is more efficient but also more complicated and therefore harder to implement, as evidenced, for example, in [61].

- *Commitments*: When commitments are used to keep the voters' vote secret, verifiable shuffling is more difficult. The challenge here is that the re-randomized shuffled commitments must be opened at the end, but without revealing any of the secret data of the individual mix servers (private re-randomization values and private permutations). Below we describe how to solve this challenge.

  This solution requires a special type of malleable public-key encryption that is called *commitment consistent encryption* [38]. This property makes it possible to derive (without any secret knowledge) from a public-key ciphertext $e = \mathsf{Enc}(pk, m)$ a commitment c that can be opened to the same message m; the corresponding opening values d to open c to m can be derived from e with the secret key sk.

  Now, the idea is to publish only the unconditionally hiding commitments on the part of the bulletin board that the auditor A can read, while the conditionally secret public-key ciphertexts are only shared on a secret part of the bulletin board BB that the mix servers can access.

  An implementation of this special public-key encryption scheme and a corresponding NIZKP of plaintext knowledge have been proposed in [38], and it has been proved in [51] that a modification of the Verificatum NIZKP of shuffle (see above) can also be used to implement a proof of shuffle for this special scheme. We will choose these implementations for our evaluation as they are technically close to the state-of-the-art mix net Verificatum.

4.5.2 Description

We now describe how the secret ballot protocols in Section 4.2.2 (for encrypted ballots) or in Section 4.3.2 (for committed ballots) are extended so that the ballots can be verifiably shuffled.

Encrypted ballots. In addition to the steps taken in the submission phase described in Section 4.2.2, the voting device of voter V_i computes a proof of knowledge π_i for the secret vote v_i. The voting device appends the proof π_i to the ciphertext e_i and posts (e_i, π_i) to BB.

After the end of the submission phase, the mix servers and talliers verify for each (e_i, π_i) whether π_i is valid w.r.t. e_i and whether neither e_i nor π_i are contained in any previous ballot; see Remark 2.

If any of these conditions does not hold, the ballot is discarded, and otherwise it is accepted for the following tallying phase. Note that this preselection can be performed and thus verified by anyone with access to BB.

In the tallying phase, the preselected encrypted votes are sent through a mix net of designated mix servers $\mathsf{M}_1, \ldots, \mathsf{M}_l$. Each of these mix servers computes a proof of shuffle and posts it together with its outcome ciphertext vector to BB.

The talliers check that the preselection phase was correct and that all mix servers posted valid proofs of shuffle. If so, the talliers secretly decrypt the aggregated result, compute a proof of correct decryption, and post the resulting plaintexts along with the proof of decryption to BB.

To verify the correctness of the election result, an observer with access to BB verifies all proofs on BB (i.e., the tallier's proof of correct key generation, the voters' proof of knowledge, the mix servers' proofs of shuffle, and the tallier's proof of correct decryption) and the correctness of the preselection. If all checks pass, the election result is accepted, otherwise it is rejected.

Committed ballots. The main idea behind this approach is to have two trails of votes: a private one to tally the ballots and a public one to audit the correctness of the tallying process.

- *Private*: The *private trail* is the same as the encrypted ballot approach described earlier. The data of this verifiable mix network is shared on a secret part of the bulletin board, called SBB, which only the mix servers and the talliers have access to, but not the auditors.
- *Public*: The commitment consistency property of the special public-key encryption scheme and of the NIZKPs (of knowledge and of shuffle) is then used to derive the *public trail*. More precisely, committed messages are derived from the corresponding encrypted messages, and NIZKPs for the commitments are derived from the corresponding NIZKPs for the encrypted messages. This data, which can be used to verify the correctness of the mixing and opening process, is published on the part of the bulletin board BB that the auditor can read.

This results in two parallel verifiable mix nets, a public one with unconditionally hiding[9] committed messages and NIZKPs that are unconditionally (aka perfectly) zero-knowledge, and a private one with computationally secret encrypted messages.

After the mixing phase, as before, the talliers check that all previous steps (preselection, NIZKPs) were correct, and, if so, they decrypt the final ciphertext vector of the private mix net. They then post the resulting messages to the public part of the bulletin board BB so that everyone can open the final commitment vector of the public mix net.

When this approach is realized with the above implementation [38, 51], the public part of the bulletin board BB is essentially a (strict) subset of the data on the secret bulletin board SBB.

4.5.3 Analysis

Secrecy. We analyze the secrecy aspects (vote privacy, everlasting privacy, vote-buying resistance) of the combination "malleable PKE (Section 4.2) & verifiable mixing" and of the combination "malleable commitments (Section 4.3) & verifiable mixing". We summarize our results in Table 4.12. We get essentially the same result as for homomorphic aggregation (Section 4.4.3) with a similar reasoning.

Vote privacy. We distinguish between the following two scenarios, both for the use of malleable PKE and for the use of malleable commitments. In the first scenario, there is only one tallier and one mix server. In the second scenario, there are multiple talliers and they generate their joint public key distributively (in the case of malleable PKE) or voters secretly distribute their votes among the talliers (in the case of malleable commitments), and there are multiple mix servers.

[9] If implemented, for example, with Pedersen's commitment scheme.

1. *Single tallier and single mix server*: In this case, the voting system protects against *Level 1* attackers, but not against *Level 2* attackers. The reason is that a *Level 2* attacker can compromise the decryption machine and thus learn the decryption key to decrypt or open any ballot on the bulletin board BB.

2. *Multiple talliers and multiple mix servers*: In this case, the voting system protects against *Level 4* attackers, but not against *Level 5* attackers. Indeed, the attacker cannot compromise enough decryption talliers to be able to decrypt or open commitments. Similarly for the mix servers who jointly realize the mix net. However, because voting device VD learns the voter's plain vote, the mechanism does not protect against *Level 5* attackers.

We would like to point to our remarks in the vote privacy analysis in Section 4.4.3 about possible shortcuts.

Everlasting privacy. We obtain essentially the same result as for homomorphic aggregation (recall Section 4.4.3). When the ballots are identifiable, then using malleable public-key encryption does not provide practical everlasting privacy, while using unconditionally hiding malleable commitments in combination with perfect zero-knowledge proofs guarantees practical everlasting privacy with *Level 3*. When the ballots are anonymous, then the scheme provides practical everlasting privacy no matter whether public-key encryption or commitments are used.

Vote buying resistance. This mechanism combined with any of the secret ballot mechanisms does not protect against vote-buying, i.e., achieves only **level 0**, but not level 1. The reason is the same as described in Sections 4.2.3 and 4.3.3.

Table 4.12: Summary of secrecy evaluation. The overall score of one of the compositions here is calculated by adding together the doubled score of vote privacy, the score of everlasting privacy, and the score of vote-buying resistance (see Section 3.2).

Aspect	*with mall. PKE*	*with mall. commitment*
Vote privacy		
One tallier	1	1
Several talliers	4	4
Everlasting privacy		
Identifiable ballots	0	3
Anonymous ballots	3	3
Vote-buying resistance	0	0

Verifiability. We study the verifiability of verifiable mixing combined with malleable PKE or malleable commitments. To this end, we analyze under which trust assumptions the correctness of the final result can be verified. We get essentially the same result as for homomorphic aggregation (Section 4.4.3) with a similar reasoning.

The correctness of each mixing step can be verified with the mix server's individual proofs of shuffle posted to BB. If public-key encryption is used, the correctness of the final decryption step can be verified with the tallier's individual proofs of correct decryption on the bulletin board BB. If a commitment scheme is used, then the correctness of the final opening step can be verified with the opening algorithm of the commitment scheme applied to all entries of the final shuffled vector. In combination, these mechanisms guarantee that the final result correctly matches the voters' submitted votes, even if the mix servers and the talliers are corrupted.

We do, however, need to make the same trust assumptions as for homomorphic aggregation (see Section 4.4.3).

Table 4.13: Verifiability trust assumptions.

hon(RA) AND hon(VD) AND hon(VS)

Usability. The mechanism does not affect usability of the voter casting their vote.

Practicality. We evaluate the implementability constraints (Section 3.5.1) as follows:

- *Skills*: NIZKPs of correct shuffling are more complicated to implement than the NIZKPs of set membership and correct decryption. Thus, compared to homomorphic aggregation (Section 4.4), stronger cryptographic skills are required. We deduce two points.
- *Resources*: No specific resources are needed for the mechanism.
- *Protocol complexity*: During the tally phase, the mix servers have to perform their part of the shuffle in a serial manner. This process implies involved interactions, which makes the protocol more complex. We deduce one point.

We evaluate the implementability enablers (Section 3.5.1) as follows:

- *Software library*: For encrypted ballots, there are libraries and systems available, notably a subset of the resources listed in Section 4.2. In particular, Verificatum [139] provides a well specified and documented mix net implementation for the entire mechanism. We grant a score of 3.

  For committed ballots, we found no equivalent library. However, the Verificatum mix net provides shuffling of arbitrary ballots, and therefore may be adapted to work over committed ballots (see [52]). We grant a score of 1.

We evaluate the communication efficiency (Section 3.5.2) as follows:

- *Data size*: The data size is small (see Sections 4.2.3 and 4.3.3).
- *Complexity*: During the tally phase, the mix servers perform their shuffle in a serial manner and block the process during their turn, see Protocol 7.7 in [58] for more details. As there is typically more than one tallier, we count this as *some* rounds, and award three points.

We evaluate the computational efficiency (Section 3.5.2) as follows:

- *Server time*: As part of the mechanism, the NIZKPs of ballot validity are verified, the ballots are shuffled and the result is verifiably decrypted.

  Verifying a proof of plaintext knowledge requires 4 exponentiations (see 5.4.1 in [58]). Then, each ballot needs to be re-randomized, requiring 2 exponentiations per ballot (see Algorithm 8.46 in [58]). Further, generating a shuffle proof requires 7 exponentiations per ballot (see Section 5.5 of [58]). For an election with 100'000 voters, this results in $100'000 * (4+2+7) = 13 * 10^5$ exponentiations. Each tallier performs the verified shuffle one-after-the-other.

  After the verified shuffle, each ballot is decrypted, and a corresponding proof of correct decryption is generated. This requires one exponentiation for the decryption, as well as two exponentiations for the proof. For 100,000 ballots, we obtain $3 \cdot 10^5$ exponentiations per trustee. Each tallier performs the verifiable decryption using their part of the private key, possibly in parallel.

  We reuse the same time estimation as in Section 4.2.3, i.e. 0.11ms. This results in $4 \times 10^5 \cdot 0.11\text{ms} =$ **44 s** for verifying the ballots, $(7+2) \times 10^5 \cdot 0.11\text{ms} =$ **99 s** for re-randomizing the

ballots and generating the shuffle proof, and $3 \times 10^5 \cdot 0.11\text{ms} =$ **33 s** for verifiable decryption, amounting to a total of **2:56 min**.

According to [51], the time needed to verifiably shuffle malleable commitments is in a similar range.

- *Voter time*: There are no computations to be performed on the voter's side.

Table 4.14: Practicality score summary.

Aspect	*with mal. PKE*	*with mal. commitment*
Skills	-1	-2
Resources	0	0
Protocol complexity	-1	-1
Software library	+3	+1
Implementability (I)	$\mathsf{trim}(5 + \mathsf{sum}(\cdot)) = 5$	3
Communication	3	3
Computation	3	3
Efficiency (E)	$\mathsf{min}(\cdot, \cdot) = 3$	3
Practicality	$\frac{2}{3}I + \frac{1}{3}E = 4.34$	3

4.6 Digital signatures

Digital signatures are a central cryptographic building block of verifiable online voting systems. While the building block itself (usually) does not require any election specific properties, its usage can help to resolve or mitigate possible security issues in online voting. For example, it can resolve possible disputes between different participants, or reduce trust in the election authority regarding the eligibility verifiability of voters.

4.6.1 Requirements

We do not need any special requirements for the digital signature scheme besides its EUF-CMA security (Section 2.2.3). There are numerous possible options to implement this cryptographic building block, for example Schnorr signatures [130] and many more (see below).

4.6.2 Description

We describe two applications of digital signatures for verifiable online voting.

Signatures of election authorities. Each election authority generates a key pair for signing messages and verifying signed messages. The election authority distributes the verification key among the participants of the voting system and keeps the signing key secret. The election authority then signs each message with its private key and the recipients only accept messages from this authority if they arrive with a signature that has been verified as correct using the authority's verification key. In particular, the voting server (as an election authority) signs valid incoming ballots and returns the signature to the voter as a receipt.

Signatures of voters. In this application, there are two ways in which voters can obtain a secret signing key. Either they generate the signing key and the corresponding verification key themselves, or they receive this key pair from a trusted party. In the second case, for example, electronic identities (eIDs) distributed by an external trust service provider could be used. In any case, the (election specific) registration authority is not involved in the creation and the distribution of the key material. Whatever approach is taken, it is important to ensure that it is implemented correctly. Since the exact method should be independent of or outside the rest of the voting system, we do not investigate it in this paper.

When casting their vote, the voter also enters their secret key into the voting device, which signs the generated ballot (Sections 4.4 and 4.5) and then sends it to the bulletin board BB. Only those ballots are then counted that have been provided with a valid signature of a voter eligible to vote.

4.6.3 Analysis

Secrecy. Signatures of election authorities do not affect secrecy. Signatures of voters (that can be verified without having to trust the registration authority), do not affect vote privacy and vote-buying resistance. However, they do affect everlasting privacy:

- *Malleable PKE*: If we extend a voting protocol that uses malleable PKE and anonymous ballots, then the ballots become identifiable due to the signatures. Therefore, the everlasting score for such schemes decreases from 3 (when combined with one of the tallying techniques from Sections 4.4 and 4.5) to 0 (recall Tables 4.9 and 4.12).
- *Malleable commitments*: If we extend a voting protocol that uses malleable commitments in combination with one of the tallying techniques described in Sections 4.4 and 4.5, then everlasting privacy remains at level 3.

Verifiability. We distinguish between the two possible uses of digital signatures that we have described above. In both cases we assume that one of the two secret ballot methods (Sections 4.2 and 4.3) is combined with one of the two verifiable tallying methods (Sections 4.4 and 4.5).

Signatures of election authorities. In this case, verifiability is strengthened by accountability (see Section 3.3.2), because the signatures guarantee that the election authorities cannot deny signed messages that they sent.

Signatures of voters. Assuming that the voters' key pairs were generated correctly and the signing keys were distributed or computed secretly, the trust in the election specific registration authority RA is reduced. This gives us the following trust assumption (if we combine a secret ballot method with a verifiable tallying method as mentioned above).

Table 4.15: Verifiability trust assumptions.

hon(VD) AND hon(VS)

Otherwise, if the PKI is set up by RA, then trust in RA is still required and we obtain the same trust assumptions as in Sections 4.4.3 and 4.5.3.

Usability. We distinguish between the two possible uses of digital signatures that we have described above.

Signatures of election authorities. Some systems provide signatures which need to be validated by the voter (e.g. a signature which confirms the ballot has been received by the system). We consider here the case where this is done (only) on the voting device VD. This case has no impact on the voter's procedure, which is why we do not evaluate it further.[10]

Signatures of voters. The signatures of votes have different user interactions depending on their implementation. Signature keys may be generated on the voting device, with the public key then being authenticated through conventional authentication procedures (e.g. see the identity-based voter authentication mode in Electa [10]). Further, signature keys may be generated by the system, and then delivered to the voter. In the latter case, the system may provide a string to be entered (e.g. in the Swiss systems [58, 134]), or a dedicated signature device (e.g. an eID card as in the Estonian setting [73]).

- *Efficiency*: In case that the signature keys are generated on the voting device, the voter then needs to authenticate the public key, which we count as one step. In case the system provides the signature key, the voter either needs to enter it, or operate the dedicated signature device, which we each count as one step. All three approaches therefore need one step, which results in an efficiency score of 5.
- *Effectiveness*: The dedicated signature device counts as an additional device, which reduces the effectiveness of this approach by one to 4. The other approaches have none of the characteristics which would reduce their effectiveness, hence both achieve an effectiveness of 5.

[10] These signatures may (additionally) be validated on an audit device, or may even be forwarded to independent parties (e.g. receipts in the POLYAS system may be delivered to auditors, which then verify the signed vote is part of the tally [119]). The precise voter's procedure depends heavily on the design of the voting system.

Table 4.16: Usability score summary.

Aspect	*Device-generated*	*Entered to device*	*Signature device*
Efficiency	5	5	5
Effectiveness	5	5	4
Usability	$\min(\cdot,\cdot) = 5$	$\min(\cdot,\cdot) = 5$	$\min(\cdot,\cdot) = 4$

Practicality. We evaluate both possible uses of digital signatures together, as their implementation is similar, and as they achieve the same score. However, we mention crucial differences, and explain why these do not affect the scoring.

We consider the Schnorr signature , which is one of the popular, lightweight and well-studied schemes. See, for example, Section 5.6 of [58]. We use the same elliptic curves as in Section 4.2.3. Each signature needs two exponentiations to generate, and is composed out of two values with each having a size of 32B.

We evaluate the implementability constraints (Section 3.5.1) as follows:

- *Skills*: Knowledge in cryptography is necessary to implement and integrate the mechanism correctly. We deduce one point.
- *Resources*: For the signatures of election authorities, no dedicated resources are needed. For the signatures of voters, some approaches may require distribution of key material to the voters (e.g. over postal mail), or a dedicated signature device. We expect however that voting systems would reuse an already established system, which is why we consider this to be outside of the actual voting system. We therefore deduce no points.
- *Protocol complexity*: The mechanism does not significantly increase the complexity of the underlying voting system.

We evaluate the implementability enablers (Section 3.5.1) as follows:

- *Software library*: Digital signatures are not specific to online voting, and a plethora of high quality and ubiquitous libraries for different languages and signature schemes exist. For example, OpenSSL[11] provides ECDSA functionalities in C, and schnorrkel[12] provides Schnorr signatures in Rust. We grant three points.

We evaluate the communication efficiency (Section 3.5.2) as follows:

- *Data size*: The size of a signature is only 64B, which is well below 1MB.
- *Complexity*: The mechanism does not add communication complexity to the system.

We evaluate the computational efficiency (Section 3.5.2) as follows:

- *Server time*: Assuming an election with 100 000 ballots, with the same number of signatures to be generated, a single server would need $10^5 \cdot 2 \cdot 0.11\text{ms} = 22\text{s}$. The process is additionally easy to parallelize.
- *Voter time*: The time to generate a signature is 0.32ms, which is well below 1s.

[11] https://www.openssl.org/docs/man3.3/man3/ECDSA_sign.html
[12] https://docs.rs/schnorrkel/0.11.4/schnorrkel/index.html

Table 4.17: Practicality score summary.

Aspect	**Score**
Skills	-1
Resources	0
Protocol complexity	0
Software library	+3
Implementability (I)	$\mathsf{trim}(5 + \mathsf{sum}(\cdot)) = 5$
Communication	5
Computation	5
Efficiency (E)	$\mathsf{min}(\cdot, \cdot) = 5$
Practicality	$\frac{2}{3}I + \frac{1}{3}E = 5$

4.7 Audit-or-cast

The *audit-or-cast* approach, sometimes called *Benaloh challenge* after its inventor [16], is a prominent technique for enabling (human) voters to provide some evidence that their voting devices VD have processed their secret votes as intended. The goal of this and the other cast-as-intended mechanisms (Sections 4.8 and 4.9) is to protect against corrupted voting devices that do not process the voters' choices correctly. The audit-or-cast mechanisms is, for example, integrated in Helios [3] and Electa [10].

Idea. In the original and most widely implemented version, the audit-or-cast approach works as follows. After the voter's choice has been encrypted (or committed to) by the voting device VD, the voter has two options: they can *either* (1) cast that encrypted (or committed) ballot *or* (2) challenge (i.e., audit) the ballot. If the latter option is chosen, the voting client enables auditing by revealing the randomness used to encrypt (or commit to) the secret vote, so that the voter (typically using an additional computer or application) can verify that this ciphertext contains the intended choice. To avoid that the audit data can serve as a trivial receipt for the voter's choice, the spoiled ballot is discarded and the voter has to restart the voting process, possibly selecting a different choice.

The security of this approach is based on the assumption that the voting device VD (possibly controlled by an adversary) does not know in advance whether the encrypted/committed vote will be audited or cast. Therefore, if the adversary controls VD and tries to manipulate the ballot, it risks being detected. Note, however, that the ballot that is actually cast is not the one that is audited. This, unlike other approaches (Sections 4.8 and 4.9), gives the voter some (probabilistic) assurance, but not fully effective guarantees.

4.7.1 Requirements

While the audit-or-cast technique can be applied to encrypted (Section 4.2) and to committed (Section 4.3) ballots without any additional cryptographic requirements, it requires an independent audit device to verify the actual ballot.

Audit device. The human voter needs an additional device, called the *audit device* AD, during the casting process. There must be a communication channel to send messages from the voting device to the audit device.

4.7.2 Description

The basic ballot submission phase (as described in Section 4.2.2) is changed as follows; all other phases remain unchanged.

First, as in the basic scheme, the voter V_i enters their vote v_i into the voting device VD, which computes the ciphertext $e_i \leftarrow \mathsf{Enc}(pk, v_i)$. Then, the voting device displays the ciphertext e_i (or a shorter cryptographic hash of e_i) to the voter. In this way, the voting device commits to the encrypted vote. Now, the voter can choose whether they want to cast or audit this ciphertext, as described next.

If the voter decides to audit the ciphertext, the voting device VD sends the ciphertext e_i, the choice v', and the random coins r' that it used to encrypt v_i to the audit device AD. The audit device AD then verifies whether $e_i = \mathsf{Enc}(pk, v'; r')$. Afterwards, the audit device displays v' to the voter, who can verify whether this v' matches the input choice v_i. If it does not match, the voter can file a complaint; the way in which the voter can do this, depends on the specification of the real voting system. In any case, the voter must restart the voting process to cast or audit a new choice (to protect against trivial vote selling, as mentioned above).

If the voter decides to cast the ciphertext, the voting device VD sends the ciphertext e_i to the voting server as in the basic secret ballot protocol. In this case, the voting device does not reveal the

random coins that were used to encrypt the cast vote, but deletes them. The voting server responds with a signature on the submitted ballot to confirm it has received the vote. The voter then again uses their audit device to check that the acknowledged ballot matches the one that the voting device displayed at the start.

4.7.3 Analysis

Secrecy. We analyze the secrecy aspects (vote privacy, everlasting privacy, vote-buying resistance) of the audit-or-cast approach. We assume that one of the two secret ballot methods (Sections 4.2 and 4.3) is combined with one of the two verifiable tallying methods (Sections 4.4 and 4.5).

- *Vote privacy*: If the voter always audits random choices, then the information that the audit device obtains is independent of the choice that the voter casts. Under this assumption, the audit-or-cast technique does not change the vote privacy results in Sections 4.4.3 and 4.5.3.

  However, if we cannot assume that human voters enter random choices to be audited, then the individual data, which is a receipt, reveals some information about how the voter actually voted and can therefore reduce the vote privacy level to 1.

- *Everlasting privacy*: The scheme with audit-or-cast provides the same level of everlasting privacy as the underlying scheme without audit-or-cast (see Sections 4.4.3 and 4.5.3).

- *Vote-buying resistance*: Audit-or-cast does not change the vote-buying resistance of the underlying system it extends. In particular, it does not weaken vote-buying resistance because audited/spoiled votes cannot be cast.

Verifiability. To evaluate the verifiability of the audit-or-cast technique, it is important to distinguish whether the voters' ballots are anonymous or identifiable (Remark 4).

If the ballots are anonymous, then neither the audit-or-cast technique we study in this section nor the cast-and-audit technique we study in Section 4.8 protect against several corrupted voting devices joining forces to secretly manipulate the votes of some of the voters using them. The reason for this is the possibility of *clash attacks*; extra protections must be implemented to avoid these attacks (see, e.g., [103]).

Remark 8: Clash attacks [103]

The general idea of a *clash attack* is to exploit situations, in which two or more different voters create the same ballot (with or without the influence of the attacker), to maliciously exchange or remove all instances of those identical ballots except one. At the same time, depending on the individual verification method, each affected voter can still be convinced that "their own" ballot was recorded.[a]

[a] There are individual verification mechanisms that still achieve their purpose even when anonymous ballots are used, for example return codes (Section 4.9) and verification codes [100, 127] (Section 4.10.2).

Let us now explain how such a clash attack can be executed in the case of anonymous ballots, even if the audit-or-cast technique is used. Assume that two voters V_1 and V_2 vote for the same candidate v, and that their voting devices VD_1 and VD_2 are controlled by an attacker. The attacker now instructs the two voting devices to use the same randomness to compute the encrypted ballots, which results in identical ballots for both voters. Further, the attacker instructs the voting device of the voter who casts second, to *not* send the ballot to the server, but instead to reuse the acknowledgment of the voting server for the first voter. Since both ballots are identical and anonymous, the second voter is fooled into believing that their own ballot has actually been recorded, while the adversary was

successfully able to drop one of the two votes.[13] This shows that in the case of anonymous ballots, the audit-or-cast mechanism we described in Section 4.7.2 does not protect against dishonest voting devices. Thus, we will now assume that identifiable ballots are used.

The audit-or-cast technique by design does not audit the cast vote, the mechanism can only provide some assurance of the correctness of the cast vote. Therefore, it is not only necessary that the voter is able to detect a possible manipulation, but also that the voting device cannot predict whether the voter will decide to audit the just-encrypted vote. This means that the audit-or-cast approach (with identifiable ballots) only provides a probabilistic level of protection against corrupted voting devices. Due to the complexity of this analysis, we refer to [103, 80] for formal analyses.

Usability. When the voter starts the voter's procedure, they must decide whether they intend to audit or to cast. If they want to audit, they enter a candidate at random; otherwise, they enter their preferred candidate. The voting device then shows a commitment to what it claims to be an encryption of that candidate, which the voter initializes their audit device with.

If the voter's intent was to audit, they advice the voting device to audit. The voting device reveals the encryption randomness to the audit device, which allows the audit device to recover the chosen candidate. If the audit device shows the correct candidate, the voter restarts the procedure. Otherwise, the voter aborts and files a complaint on a channel that is independent from their voting device.

If the voter's intent was to cast, they advice the voting device to cast, and use the audit device to check that indeed the encrypted ballot is cast in their name.

- *Efficiency*: We consider here the case when the voter audits once, and then casts their vote.

  The voter's procedure starts by entering a random candidate, and initializing the audit device. Then, the voter advises the voting device to audit, and performs the audit on the audit device (whether the displayed vote is correct). So far, the voter needed four steps.

  Afterwards, the voter restarts the vote procedure by selecting their preferred candidate, and again initializes the audit device. The voter advises the voting device to cast, and performs the audit on the audit device (whether the ballot is submitted in their name). This adds another four steps, resulting in an efficiency score of -2.[14]

  Note that for the mechanism to be effective, at least some voters need to audit at least once, while the voter's strategy must remain unpredictable to the voter's device. Some voters may even audit more than once, which further increases the number of steps required to execute the mechanism. We account for this by prefixing the score by a smaller-than-or-equal sign $\leq$.

- *Effectiveness*: Before entering the candidate, the voter needs to decide on an audit strategy, which we consider a difficult task.[15] The audit step is optional, but has a large impact, as failing to perform it correctly may allow the adversary to change the vote. Further, the audit needs an additional device. Altogether, this gives an effectiveness score of 1.

[13] Of course, this attack can be generalized to apply to more than two voters at the same time. Further, the adversary may also choose, instead of simply dropping, to *replace* ballots.

[14] We analyze here the classic version of the mechanism, however we are also aware of a variant [10] which achieves a slightly better ≤ -1 in efficiency (still resulting in the same overall score). In this variant, the voter always casts their ballot, but is only *afterwards* given the choice to audit it. In case the voter chooses to audit, the ballot is first invalidated, then the voter performs the audits (whether the vote is correct, and the ballot is submitted in their name), and then restarts the voter's procedure. In case the voter chooses to not audit, no more action is necessary. This overall saves one step, namely initializing the audit device when the vote is not audited.

[15] Studies find that this step does not align well with the mental model of voters about verification (e.g. [112, 2]).

Table 4.18: Usability score summary.

Aspect	Score
Efficiency	≤ -2
Effectiveness	1
Usability	$\min(\cdot, \cdot) = 0$

Practicality. We evaluate the implementability constraints (Section 3.5.1) as follows:

- *Skills*: Knowledge in cryptography is necessary to implement and integrate the mechanism correctly. We deduce one point.
- *Resources*: No specific resources are needed for the mechanism.
- *Protocol complexity*: The mechanism does not significantly increase the complexity of the underlying voting system.

We evaluate the implementability enablers (Section 3.5.1) as follows:

- *Software library*: No library is, to our knowledge, available for the mechanism. Helios[16] implements the mechanism. We note that the mechanism is rather simple, and access to libraries which offer just its building blocks (e.g. encryption), some of which we describe in Section 4.2.3, should be sufficient. We grant a score of 2.

We evaluate the communication efficiency (Section 3.5.2) as follows:

- *Data size*: The voting device needs to transfer the ballot and the encryption randomness to the audit device. Implemented as proposed in Section 4.2.3, this remains small.

  Further, the audit device needs to check whether the ballot is indeed stored by the server, most likely by downloading an appropriate digital signature from the server. Besides the signature, the audit device would likely download additional meta-data (e.g. the election the ballot was cast in).

  It is safe to conclude that the data size remains small and clearly under 1MB.

- *Complexity*: No blocking operation decreases the communication efficiency.

We evaluate the computational efficiency (Section 3.5.2) as follows:

- *Server time*: The mechanism does not need computation on the server side.
- *Voter time*: For each audit, the voting device needs to encrypt the vote, and the audit device needs to decrypt it. Implemented as proposed in Section 4.2, this is fast and remains clearly under 1s.

[16] `https://github.com/benadida/helios-server`, originally published in [3]

Table 4.19: Practicality score summary.

Aspect	**Score**
Skills	-1
Resources	-1
Protocol complexity	0
Software library	+2
Implementability (I)	$\mathsf{trim}(5 + \mathsf{sum}(\cdot)) = 5$
Communication	5
Computation	5
Efficiency (E)	$\mathsf{min}(\cdot, \cdot) = 5$
Practicality	$\frac{2}{3}I + \frac{1}{3}E = 5$

4.8 Cast-and-audit

The *cast-and-audit* approach is another prominent technique for enabling (human) voters to verify that their voting devices VD have processed their secret votes as intended. Such mechanisms are, for example, integrated in the Estonian [73] and in the Polyas online voting system [119].

Idea. The basic idea is that the voting device VD reveals some information data to the audit device AD that can be used to revert and thus verify the encryption of the voter's choice. However, care must be taken that this data does not serve as a receipt for how the voters voted.

We will describe and analyze two different approaches to tackle this problem. The first technique mitigates the problem on the conceptual level at the cost of public verifiability. The second technique solves the problem on the cryptographic level without affecting public verifiability.

4.8.1 Requirements

Both cast-and-audit techniques that we describe and analyze can be applied to encrypted (Section 4.2) and to committed (Section 4.3) ballots. Both techniques require an independent audit device to verify the actual ballot. The second technique additionally requires that the underlying PKE scheme (or commitment scheme) is re-randomizable and that a corresponding interactive zero-knowledge proof is available.

Audit device. The human voter needs an additional device, called the *audit device* AD, during the casting process.

Re-randomization and interactive zero-knowledge proof. For the second technique, we assume that the public-key encryption scheme is re-randomizable (Section 4.2.1). More specifically, we require that $\mathsf{ReRand}(pk, \mathsf{Enc}(pk, m, r), x) = \mathsf{Enc}(pk, m, r + x)$ holds true for all messages m and all random coins r, x. This property is guaranteed, for example, for ElGamal PKE (see Section 4.2.1) and for Pedersen commitments (Section 4.3.1).

Furthermore, we assume an *interactive* zero-knowledge proof (ZKP) to prove that a given ciphertext e^* is a re-randomization of e. The main difference to non-interactive zero-knowledge proofs (Section 2.2.4) is that interactive ones require an active verifier (in our case the audit device) and guarantee that the proof is convincing only to the active verifier, since it can be simulated/"faked" towards any other party.

To implement such an interactive ZKP, we can use essentially the same building blocks as for the NIZKP (for the same relation) that we mentioned in Section 4.5.1; however, since this proof is interactive, we must use a technique such as Protocol 6.5.1 [70, 107] (which is also implemented in [119]) to make this proof sound.

4.8.2 Description

We describe two versions of this approach:

- In the first version, the audit material can also convince a third party how the voter voted; in other words, without further means, a voter cannot deny how they voted to an observer who receives the individual audit material. This issue can, however, be mitigated under the assumption that voters can re-vote and that the auditing parties A (who can access the bulletin board) do not collude with any corrupted observer. On the downside, assuming that A is trusted in this respect means that the public (including possible vote buyers) cannot access and therefore not verify the content of the bulletin board.

- The second version is designed so that the audit material can only convince the corresponding voter that their vote was cast as intended, even if voters cannot re-vote or if the bulletin board is public. While the technique is not designed to prevent voters from willingly selling their votes, it does not reduce the level of receipt-freeness of the underlying voting system that it extends (assuming that the audit device is not corrupted).

In both versions, only the basic ballot submission phase (as described in Section 4.2.2) is changed, while all other phases remain unchanged. For simplicity, we assume that malleable PKE is used to keep individual choices secret (Section 4.2); the case of malleable commitments (Section 4.3) is analogous.

Version without cryptographic deniability. First, as in the basic scheme, the voter V_i enters their vote v_i into the voting device VD, which computes the ciphertext $e_i \leftarrow \mathsf{Enc}(pk, v_i)$ with randomness r_i (and a corresponding proof of knowledge). Then the voting device sends the encrypted ballot e_i to the voting server VS over the dedicated authenticated channel.

The voting server VS verifies the identity of the voter, creates a ballot identifier vid for e_i, and returns vid to the voting device.

If the voter decides to audit the ballot, the audit device AD queries VD to obtain a ballot identifier vid and the random coins r that VD used to compute e_i. Then AD sends vid to the voting server to obtain a ciphertext e_i. The audit device uses the randomness r to compute v such that $e_i = \mathsf{Enc}(pk, v; r)$. Finally, the audit device displays v to the voter, who can check whether v matches the cast vote v_i.

Version with cryptographic deniability. First, as in the secret ballot protocol (Section 4.2), the voter V_i enters their vote v_i into the voting device VD, which computes the ciphertext $e_i \leftarrow \mathsf{Enc}(pk, v_i)$ with randomness r_i (and a corresponding proof of knowledge). Then the voting device sends the encrypted ballot e_i to the voting server VS over the dedicated authenticated channel.

Now, to allow the voter to verify the cast vote, the voting server establishes a "blinded verification track" of e_i as follows. This does not change the ciphertext e_i, which will be tallied, but provides a way to verify that e_i encrypts the intended choice without revealing the randomness r_i to the voter. More precisely, the voting server VS first verifies the identity of the voter, chooses random coins x (from the set of possible random coins to encrypt messages) and re-randomizes e_i with x to e_i^*, i.e., $e_i^* = \mathsf{ReRand}(pk, e_i, x)$. Then, the voting server VS returns x to the voting device VD, which computes the blinded randomness $r^* \leftarrow r + x$, where r is the randomness that VD used to compute e_i.

If the voter decides to audit the ballot, the audit device AD first queries VD to obtain the blinded randomness r^* and then the voting server VS to obtain the original ciphertext e_i and the re-randomized ciphertext e_i^*. The voting server VS and the audit device AD run the interactive zero-knowledge proof to prove/verify that e_i^* is a re-randomization of e_i. If the proof is successful, the audit device decrypts the blinded ciphertext e_i^* with the blinded randomness r^* to obtain v^*. Finally, the audit device displays v^* to the voter, who can check whether v^* matches the cast vote v_i.

Let us now explain why the recomputed choice v^* matches v_i if the two devices and the voting server run their dedicated programs. Due to the re-randomization property, the re-randomized ciphertext e_i^* is of the form $e_i^* = \mathsf{Enc}(pk, v_i, r + x)$ and the randomness r^* is of the form $r^* = r + x$. Therefore, when the audit device AD decrypts e_i^* with r^*, it obtains the choice v_i that the voter cast.

4.8.3 Analysis

Secrecy. We analyze the secrecy aspects (vote privacy, everlasting privacy, vote-buying resistance) of the cast-and-audit approach. As for our evaluation of the audit-or-cast approach (Section 4.7.3), we assume that one of the two secret ballot methods (Sections 4.2 and 4.3) is combined with one of the two verifiable tallying methods (Sections 4.4 and 4.5).

- *Vote privacy*: In the version without cryptographic deniability, the receipt that the voter obtains from their voting device reveals to an attacker how the voter voted if the attacker can also access the semi-public information. Therefore, with this technique, the vote privacy level is 2.

  In the version with cryptographic deniability, the vote privacy level of the underlying scheme (Sections 4.4.3 and 4.5.3) is preserved because the receipt only reveals how the voter voted if the voting server is also corrupted.

- *Everlasting privacy*: The scheme with cast-and-audit provides the same level of everlasting privacy as the underlying scheme without cast-and-audit (see Sections 4.4.3 and 4.5.3).

- *Vote-buying resistance*: When the version *without* cryptographic deniability is used, the cast-and-audit technique makes vote buying easy if voters cannot re-vote or if a vote buyer learns the election audit data. On the other side, the version *with* cryptographic deniability does not weaken the vote-buying resistance of the underlying system in all cases.

Verifiability. As in the evaluation of the audit-or-cast technique (Section 4.7.3), we distinguish between the cases of anonymous ballots and identifiable ballots (Remark 4). In the case of anonymous ballots, the cast-and-audit technique we study in this section does not protect against corrupted voting devices because the same kind of clash attacks (Remark 8) is possible as when the audit-or-cast technique studied in Section 4.7 is used with anonymous ballots. As mentioned in Section 4.7.3, extra protections must be implemented to avoid these attacks.

In the following, we assume that the ballots are identifiable, which protects against clash attacks or similar vulnerabilities. We assume that one of the two secret ballot methods (Sections 4.2 and 4.3) is combined with one of the two verifiable tallying methods (Sections 4.4 and 4.5). If the audit device is not corrupted, then in both versions (with and without cryptographic deniability), the voting device VD does not have to be trusted for verifiability. We thus obtain the following trust assumptions.

Table 4.20: Verifiability trust assumptions.

$\mathsf{hon}(\mathsf{RA})$ AND $\mathsf{hon}(\mathsf{VS})$ AND ($\mathsf{hon}(\mathsf{VD})$ OR $\mathsf{hon}(\mathsf{AD})$)

Note that in combination with digital signatures (Section 4.6), we can also avoid the trust assumption $\mathsf{hon}(\mathsf{RA})$.

Usability. We consider both versions of the mechanism together since the version with cryptographic deniability does not impact the voting procedure.

The voter's procedure starts by entering their preferred candidate, which the voting device casts. Then, the voter may optionally audit the ballot. To perform the audit, the voter initializes the audit device to access the encrypted ballot. Then, the voter checks whether the vote displayed on the audit device is correct, and whether the vote has been cast in their name. If the vote is not correct, the voter aborts and files a complaint.

- *Efficiency*: The voter first enters their chosen candidate, which is then cast by their voting device. Then, to perform the audit, the voter initializes the audit device, checks whether the

displayed vote is correct, and has been cast in their name. The four executed steps result in an efficiency score of 2.

- *Effectiveness*: The audit step is optional, but has a large impact, as failing to perform it correctly may allow the adversary to change the vote. Further, the audit needs an additional device. Altogether, this gives an effectiveness score of 2.

Table 4.21: Usability score summary.

Aspect	**Score**
Efficiency	2
Effectiveness	2
Usability	$\min(\cdot,\cdot) = 2$

Practicality. We evaluate the implementability constraints (Section 3.5.1) as follows:

- *Skills*: Knowledge in cryptography is necessary. We deduce one point.
- *Resources*: The voter needs an audit device, which is considered an additional resource. We deduce one point.
- *Protocol complexity*: The version without cryptographic deniability does not significantly increase the complexity of the underlying voting system. However, we consider the interactive zero-knowledge proof between the voting and audit device for the version with cryptographic deniability a complex interaction, which is why we deduce one point for this version.

We evaluate the implementability enablers (Section 3.5.1) as follows:

- *Software library*: No library is, to our knowledge, available for the mechanism. The building blocks are however readily available (see Section 4.2.3). For the version without cryptographic deniability, this should enable a straightforward implementation, which is why we grant two points. The version with cryptographic deniability, which is specified in a research paper [119], is more evolved to compose out of its building blocks, which is why we grant one point.

We evaluate the communication efficiency (Section 3.5.2) as follows (total with/without cryptographic deniability: 3/5):

- *Data size*: The version without cryptographic deniability needs to transfer the ballot identifier vid, the randomness r and the ciphertext e_i between the voting device, the audit device and the voting server. If the cryptography is implemented as proposed in Section 4.2.3, all values are small.[17]

  The version with cryptographic deniability additionally transfers the second ciphertext e_i^*[18], as well as the messages part of the interactive ZKP. Overall, all values remain small, and clearly under 1MB.

- *Complexity*: In both versions, the audit device AD requests the randomness from the voting device VD and the ciphertext from the voting server, which we count as one round of small size. We grant five points.

  In the version with cryptographic deniability, additionally AD and VD perform an interactive ZKP, which adds another round. We grant three points.

[17] For the identifier, a couple of bytes are sufficient.
[18] The randomness r and the blinded randomness r^* are essentially of the same size.

We evaluate the computational efficiency (Section 3.5.2) as follows:

- *Server time*: The version without cryptographic deniability needs no computations on the server side, but only to store or forward the ballot. The version with cryptographic deniability additionally needs to perform the re-randomization of the ballot. Implemented as proposed in Section 4.2.3, this is efficient. The server time remains clearly under a few minutes.
- *Voter time*: The version without cryptographic deniability needs to decrypt the ballot. In the version with cryptographic deniability, the audit device additionally needs to perform the interactive ZKP. All operations are efficient if implemented as proposed in Section 4.2.3. The client time remains for both mechanisms under 1s.

Table 4.22: Practicality score summary.

Aspect	*without*	*with cr. deniability*
Skills	-1	-1
Resources	-1	-1
Protocol complexity	0	-1
Software library	+2	+1
Implementability (I)	$\mathsf{trim}(5 + \mathsf{sum}(\cdot)) = 5$	3
Communication	5	3
Computation	5	5
Efficiency (E)	$\mathsf{min}(\cdot, \cdot) = 5$	3
Practicality	$\frac{2}{3}I + \frac{1}{3}E = 5$	3

4.9 Return codes

Return codes are another way to enable (human) voters to verify that their voting devices VD have processed their secret votes as intended. While the mechanism does not need the use of an audit device, such as the audit-or-cast and the cast-and-audit techniques (Sections 4.7 and 4.8), it does require a setup component which processes the correct return codes, and a secure channel through which the voters can learn them (e.g., by postal mail). This approach is integrated, for example, in the Swiss setting, in the deployed Swiss Post System [134] and in the research-oriented CHVote [58].

Idea. Before the voting phase begins, each voter receives a *code sheet* (e.g., by mail) listing all possible voting choices along with the corresponding *return codes*. These codes are unique to each voter and each vote, and need to be kept secret. During the voting phase, after voting, the voter receives (via the voting application or another dedicated channel) the return code corresponding to the selected choice. The voter compares this received code with the one listed on the code sheet next to the intended choice.

4.9.1 Requirements

The return codes can be applied to both encrypted (Section 4.2) as well as committed (Section 4.3) ballots. It requires a setup component SC to produce the code sheet, with the sheet then sent over a trusted channel to the voter. Further, the voting server VS must be able to (re-)generate the return codes from the cast ballot. Finally, for the system to be able to create a complete code sheet, all voting choices must be known in advance.

Setup component. The setup component SC constructs the code sheet for the voter. As during this step it learns the return codes, it needs to be trusted. SC is further responsible to send the code sheet over a trusted channel (both secret and authenticated) to the voter.

Note that the mechanism needs no complex computations to be performed on the voter side (as opposed to audit-or-cast and cast-and-audit, which both therefore need an audit device): Comparing the correct return codes with the actual return codes displayed in the voting application can be done directly by the human voter. This eliminates the need for a trusted device to perform computations on the voter's side. However, in turn it requires this trusted SC, as well as the secure channel from SC to the voter.

Computation of return codes. Given the (encrypted or committed) ballot, the server must be able to derive the return codes sent back to the voter. In the simplest case for the mechanism, all possible ballots are formed in advance, and for each of these pre-formed ballots, a return code is stored next to it. Calculating the return codes is then a simple lookup of what is stored next to the pre-formed ballot. However, ballots may not always be known in advance, e.g. because the voter's device performs randomized ballot encryption based on the voter's plaintext choice. Extracting return codes is then more complicated, but still possible, e.g., using *plaintext equivalent tests (PETs)* [22]. In this study, we only consider pre-formed ballots, which is sufficient to show the two fundamentally different approaches from the voter's perspective.

4.9.2 Description

We adopt the setting described in Section 4.2.2, add code sheet generation after the generation of the talliers' public/private key pair (pk, sk), and modify the ballot submission phase. For simplicity, we assume that malleable PKE is used (Section 4.2); the case of malleable commitments (Section 4.3) is analogous.

We distinguish between the following two versions, which both use pre-formed ballots, but differ mainly in their user interactions:

- *Entering pre-formed ballots*: The voter enters the pre-formed ballot (i.e., their encrypted choice) directly.
- *Entering plaintext ballots*: The voter selects their candidate in the voting application, which then recomputes the pre-formed ballot corresponding to that candidate.

In the first version, the voting application (and thus the voting device) never learns the voter's choice, but it may be cumbersome to use for the voter, who can no longer select their candidate directly in the voting application. In the second version, the voting application learns the voter's choice, but it is potentially easier to use for the voters.

Enter pre-formed ballot. To generate the code sheet for voter V, the setup component SC generates for each possible vote v_i (each candidate) a triplet (v_i, e_i, rc_i), where $e_i \leftarrow \mathsf{Enc}(pk, v_i)$ is an encryption of the vote v_i and rc_i is a randomly chosen return code. SC then secretly shuffles all tuples (e_i, rc_i) and sends the result to the voting server VS. Furthermore, SC sends all (v_i, e_i, rc_i) (the code sheet) to voter V via the secure channel.

To cast the vote v_i, the voter then enters the corresponding e_i into their voting device VD, which forwards it to the voting server VS. VS responds with the corresponding return code rc'_i. VD shows rc'_i to the voter, who checks if $rc'_i = rc_i$.

Enter the plaintext ballot. To generate the code sheet for voter V, the setup component SC generates for each possible vote v_i (each candidate) a triplet (v_i, e_i, rc_i), where $e_i \leftarrow \mathsf{Enc}(pk, v_i)$ is an encryption of the vote v_i and rc_i is a randomly chosen return code. SC then secretly shuffles all tuples (e_i, rc_i) and sends the result to the voting server VS. Additionally, SC symmetrically encrypts each (v_i, r_i), where r_i will later be used to encrypt v_i, under a randomly chosen key k into a store s, and posts s to the bulletin board. Finally, SC sends k and all (v_i, rc_i) (the code sheet) to the voter V via the secure channel.

To cast the vote v_i, the voter enters v_i and k into their voting device VD. VD downloads s and then uses k to decrypt s and retrieve the corresponding r_i. VD (re-)computes $e_i \leftarrow \mathsf{Enc}(pk, v_i, r_i)$ and sends e_i to the voting server VS. VS responds with the corresponding return code rc'_i. VD shows rc'_i to the voter, who checks if $rc'_i = rc_i$.

4.9.3 Analysis

Secrecy. We analyze the secrecy aspects (vote privacy, everlasting privacy, vote-buying resistance) of the return code approach. As for our evaluation of the audit-or-cast approach (Section 4.7.3) and cast-and-audit approach (Section 4.8.3), we assume that one of the two secret ballot methods (Sections 4.2 and 4.3) is combined with one of the two verifiable tallying methods (Sections 4.4 and 4.5).

- *Vote privacy*: When entering pre-formed ballots, the privacy increases to **level 5** if the roles of the registration authority and the talliers are distributed. Otherwise, the privacy level does not change.

  When entering plaintext ballots, the privacy level of the underlying scheme without return codes does not change.

- *Everlasting privacy*: The scheme with return codes provides the same level of everlasting privacy as the underlying scheme without return codes (see Sections 4.4.3 and 4.5.3).

- *Vote-buying resistance*: Neither version improves the vote-buying resistance of the underlying system that it extends. For both pre-formed and plaintext ballots, it holds that if the adversary has access to the short-term voting material (which includes the code sheet), then the adversary can check how the voter voted. Therefore, a system using such a mechanism cannot reach level 3 or higher.

Verifiability. We assume that one of the two secret ballot methods (Sections 4.2 and 4.3) is combined with one of the two verifiable tallying methods (Sections 4.4 and 4.5). If the setup component is not corrupted, then in both versions (entering pre-formed ballots or entering plaintext ballots) the voting device VD does not have to be trusted for verifiability. We thus obtain the following trust verifiability result.

Table 4.23: Verifiability trust assumptions.

hon(RA) AND hon(VS) AND (hon(VD) OR hon(SC))

Usability. For pre-formed ballots, the voter's procedure starts by the voter entering the pre-formed ballot (essentially a long string) corresponding to their preferred candidate. For plaintext ballots, the voter enters directly their preferred candidate, and a key (essentially a long string, too). After casting the pre-formed or plaintext ballot, the voter may optionally compare the return code.[19] If the return codes are not correct, the voter files a complaint.

- *Efficiency*: For pre-formed ballots, the voter enters directly the ballot. This is a non-trivial effort for each candidate, for which we deduce an efficiency point. After casting the ballot, the voter compares the return code. This is again a non-trivial effort for each candidate, which leads us to deduce another point. Overall, the two executed steps minus two point deductions lead to an efficiency score of 2.

  For plaintext ballots, the voter enters their preferred candidate. Further, the voter enters the key. After casting the ballot, the voter compares the return codes, which is a non-trivial effort for each candidate. Overall, the three executed steps minus one point deduction lead to an efficiency score of 2.

- *Effectiveness*: While entering long strings may be tedious, we do not consider it to be difficult. Further, we do not consider comparing return codes to be difficult, notably we observe that other common applications rely on similar mechanisms, such as some types of second factor authentication. However, the audit step is optional and has a large impact, as failing to perform it correctly may allow the adversary to change the vote. Altogether, this gives an effectiveness score of 3.

Table 4.24: Usability score summary.

Aspect	*pre-formed*	*plaintext*
Efficiency	2	2
Effectiveness	3	3
Usability	$\min(\cdot,\cdot) = 2$	$\min(\cdot,\cdot) = 2$

[19] Comparing the return codes may, depending on the implementation, involve an additional device or induce a media break (e.g. in the Swiss setting, the return codes are on a sheet of paper delivered by mail [134]).

Practicality. We evaluate the implementability constraints (Section 3.5.1) as follows:

- *Skills*: Knowledge in cryptography is necessary to implement and integrate the mechanism correctly. We deduce one point.
- *Resources*: The secure channel required for the mechanism from the system to the voter represents a resource hard to obtain. For example in the Swiss Post System, the channel is implemented by sending a letter over postal mail. We deduce two points.
- *Protocol complexity*: The addition of the setup component increases the complexity of the interactions. We deduce one point.

We evaluate the implementability enablers (Section 3.5.1) as follows:

- *Software library*: No library is, to our knowledge, available for the mechanism. For both the Swiss Post system [134] as well as the CHVote system [58], which both implement their own flavour of return codes, the specification and code is however available. Further, some versions of the mechanism are rather simple (e.g. the versions we describe here), and access to libraries which offer just its building blocks (e.g. encryption), of which multiple exist (see Section 4.2.3), should be sufficient. We grant two points.

We evaluate the communication efficiency (Section 3.5.2) as follows:

- *Data size*: The mechanism needs to form a ciphertext for every possible vote per voter. Further, these ciphertexts need to be sent to the servers and the voter (either on the voting sheet for pre-formed ballots, or within the store for plaintext ballots). Even for 100 candidates, as seen in Section 4.2.3, the data size per voter remains well under 1MB.
- *Complexity*: Generating the code sheet, and the corresponding communication with the server and the voter, needs only non-blocking communication. After casting the vote, the voter however needs to wait for the server to respond with the return code, which counts as one round of small size. For the mechanism variant using plaintext ballots, the voting device additionally needs to download the store, which we however do not consider blocking communication, as it can be done asynchronously. We grant five points.

We evaluate the computational efficiency (Section 3.5.2) as follows:

- *Server time*: For both versions of the mechanism, the server computation is an efficient lookup (e.g. a database query), which relates the ciphertext to its correct return code. We do not evaluate here the computation of the setup component, for which performance is less relevant, as execution can start long before the voting phase starts.[20] We remain well under a few minutes.
- *Voter time*: For pre-formed ballots, the mechanism does not need any computation on the side of the voter. For plaintext ballots, the voting device needs to decrypt the store, for which only (generally very efficient) symmetric encryption is necessary.[21] Further, the device needs to recompute the encryption of the vote, which is efficient as seen in Section 4.2.3. We remain well under 1s.

[20] Observe how the setup component primarily executes encryptions, which are efficient, as seen in Section 4.2.3.

[21] Symmetric encryption usually needs a low number of *cycles* per byte, hence it is by factors faster than asymmetric encryption.

Table 4.25: Practicality score summary.

Aspect	**Score**
Skills	-1
Resources	-2
Protocol complexity	-1
Software library	+2
Implementability (I)	$\mathsf{trim}(5 + \mathsf{sum}(\cdot)) = 3$
Communication	5
Computation	5
Efficiency (E)	$\mathsf{min}(\cdot, \cdot) = 5$
Practicality	$\frac{2}{3}I + \frac{1}{3}E = 3.67$

4.10 More methods

In this section, we list additional methods and approaches for verifiable online voting that we have identified through our literature and market analysis. Some of them (such as the implementation of secure bulletin boards) are complementary to the methods we described and evaluated before, while others represent alternatives (such as alternative verifiable mix nets). In contrast to the eight mechanisms in the main part, we describe these additional methods only briefly and refer to the respective publications for more details.

4.10.1 Secure bulletin board

The bulletin board BB is a critical component of verifiable voting protocols because this broadcast channel with memory guarantees that all information to verify an election is provided accurately and consistently to the auditors and other participants. If BB does not work properly, then all verifiability and secrecy properties collapse.

While many voting protocols in the literature include bulletin boards, most work abstracts away from these central protocol components and considers them as black boxes. The implementation and security analysis of secure bulletin boards has been studied in the following publications [71, 86, 37, 69, 74, 56]. We are, however, not aware of any real online voting system that uses such an approach to realize a secure bulletin board.

4.10.2 Verifiable mix nets

In Section 4.5, we described and evaluated re-randomization mix nets that are made verifiable by proofs of shuffle. While this combination net has become the most established way to implement verifiable mix nets in online voting, there is also another type of mix net that can be used, and several other techniques to make mix nets verifiable. In the following, we briefly describe these alternative solutions and refer to [62] for a systematic and more detailed review of them.

Types of mix nets. There are two types of mix nets: *decryption mix nets (DMNs)* and *re-randomization mix nets (RMNs)*. Originally, the concept of a DMN was proposed by Chaum [27] in 1981, and the one of a RMN by Park, Itoh, and Kurosawa [123] in 1993. We already described how a RMN works in Section 4.5. A DMN works as follows.

Decryption mix nets. Each mix server M_k holds a public/private key pair (pk_k, sk_k) of an IND-CCA-secure public key encryption scheme, and the senders know the public keys $pk_1, \ldots, pk_n$.

Each sender S_i iteratively encrypts its plaintext input message msg_i using the public keys $pk_1, \ldots, pk_n$ of the mix servers $\mathsf{M}_1, \ldots, \mathsf{M}_n$ in reverse order. The sender S_i submits the resulting nested ciphertext e_i to the first mix server M_1.

The first mix server M_1 uses its secret key sk_1 to decrypt the outermost encryption layer of all input ciphertexts, shuffles the decrypted messages, and forwards them to the second mix server M_2. The second mix server M_2 uses its secret key sk_2 to decrypt the next encryption layer, shuffles the result, and so on. Finally, the last mix server M_n outputs the plaintext messages initially selected by the senders in random order.

Techniques for verifiable mix nets. There exist several alternatives to using proofs of shuffles to make a mix net verifiable. In the following, we describe their main ideas.

- *Randomized Partial Checking (RPC)* was originally proposed in [79] (for both RMNs and DMNs). RPC has since been used in many e-voting systems since, such as Prêt à Voter [128] for real-world on-site elections in Australia [24]. However, researchers observed that the verifiability

and privacy level of RPC is lower than originally claimed [85]; these results were later formally proven in [101] for re-randomization mix nets, and in [104] for decryption mix nets. A simple way to fix the problems of the original RPC technique for decryption mix nets was proposed in [64].

RPC can be regarded as a relaxed version of a proof of shuffle, trading weaker security and privacy for higher efficiency. The basic idea is to ask each mix server to reveal the traces of a randomly selected subset of the ciphertexts it has processed. For this purpose, a mix server in a decryption mix net provides a proof of decryption for these messages, and a mix server in a re-randomization mix net reveals the random coins that it used to re-randomize these messages.

- *Message tracing* is a simple technique to provide a certain level of verifiability in a decryption mix net. The idea is that each sender S_i knows the trace of its own input through the mix network, and can therefore look up the output of the mix network (which includes the intermediate results of the mix servers) to verify that this trace has not been broken. Message tracing was proposed in [100] and formally analyzed in [100, 42, 116].

- *Verification codes* are another simple mechanism to provide a certain level of verifiability in a decryption mix net. In such a mix net, each sender chooses a random verification code and appends it to its message; the sender can then verify whether its message/code pair is in the final outcome of the mix net. The idea of verification codes was originally proposed in [129], and later used in [100] where it was also formally analyzed.

- The concept of the *trip wire* technique was originally used in a specific variant in the replication technique (see below) as a subroutine, and generalized in [20] so that it could be used as an independent verifiability technique. The basic idea of the trip wire technique is to hide the senders' inputs by a set of indistinguishable *dummy inputs* that act as trip wires. The trip wires are injected by a set of auditors, one of which must be temporarily trusted (similar to the RPC technique, see above). Now the mix net is run with this extended set of inputs. Once mixing is complete, the auditors publicly reveal the trip wires' traces through the mix net. If a mix server M_k manipulated one of the dummy traces, then M_k can be identified and be held accountable.

- The *message replication* technique was originally proposed in [84] for decryption mix nets and formally analyzed in [62]. The basic idea of the replication is to let each sender S_i replicate its input multiple times, so that all of its replications are also part of its input. Now, if a malicious mix server M_k tries to manipulate S_i's input, then M_k must simultaneously manipulate S_i's replicated inputs *in the same way* as well.

- Finally, there also exist alternative proofs of shuffle to [135], which we used to implement and evaluate verifiable mix nets in Section 4.5. These include, for example, [63, 6, 5, 135, 15, 108, 44, 46, 45, 88, 72], which were all designed for re-randomization mix nets.

4.10.3 Cast-as-intended

In the main part, we described and evaluated three cast-as-intended techniques, namely audit-or-cast (Section 4.7), cast-and-audit (Section 4.8), and return codes (Section 4.9). In addition to these techniques, there exist alternative proposals to enable voters to verify whether their voting devices have processed their votes correctly.

- *Pre-authenticated voting codes*: In this approach, the voter receives a voting sheet with pre-authenticated voting codes. To vote, the voter enters the code (or scans a QR code) corresponding to their choice. By construction, the voting client is then only able to prepare a valid

ballot for the selected choice, and no others. The voter however still needs to check that the voting client actually forwarded the ballot to the server.

This approach is used, for example, in [30], where the voting codes are used not only to provide verifiability, but also to protect ballot privacy against dishonest voting clients.

- *Custom hardware tokens*: Another option for verifying voting devices is to use special hardware tokens during the voting process. However, it has been shown in [87] that the only existing implementation of this approach [57] suffers from several security problems and is therefore not yet ready for deployment.

- *Verification codes*: The sElect system [100] provides cast-as-intended in a different way by implementing the *verification code* for mix nets (see Section 4.10.2). Voters choose random verification codes as they cast their ballots and attach them to their votes. After the tally, voters can check that their verification codes appear next to their votes. With this technique, voters can verify the entire voting process end-to-end.

  However, the simple verification code technique in sElect makes vote buying trivial. The reason is that the voter can simply give the verification code to a vote buyer, who can then look up the election result to check whether the voter obeyed. To address this issue, the Selene system [127] implements a more complex cryptographic machinery to use verification codes without enabling trivial vote buying.

4.10.4 Freedom of choice

Many voting protocols have been proposed to guarantee by technical means that voters can vote freely without being influenced by vote-selling or coercion:

- *Fake credentials* [81, 28, 92, 9, 7, 8]: By using fake credentials for authentication, voters who are coerced to vote for a particular candidate, can make up an invalid credential that they use to submit their coerced votes. The usability of this approach was empirically studied in [120] and turned out to be very low.

- *Deniable vote updating* [109, 65, 118, 95, 97, 73]: Voters can secretly update their (possibly coerced or sold) votes so that only their intended votes are tallied. Although the usability of this approach has not yet been studied, the voters' ceremonies seem to be much easier than in the (practically ineffective) fake credential approach.

- *Re-randomizable signatures* [26, 30]: The voters' signed ballots are re-randomized before being posted to the bulletin board in order to render the random coins used for encryption useless for vote-selling.

- *Selene* [127, 77]: This protocol uses verification codes that cannot be sold as trivial receipts of how the voters voted to provide some level of protection against corrupted voting devices.

- *Masking* [11, 138]: This technique enables voters to blind their choices with cryptographic masks so that their intended votes are counted, while being able to compute a valid-looking mask for any coerced choice.

4.10.5 Post-quantum voting

The security of the cryptographic building blocks currently used in real online voting systems, but also in most academic proposals, is based on so-called "classical" hardness assumptions. For example, the security of ElGamal PKE (Section 4.2.1), the binding property of Pedersen commitments

(Section 4.3.1),[22] and the soundness property of the most common NIZKP for these primitives (Sections 4.2.1 and 4.3.1) are based on the hardness of computing the discrete logarithm efficiently in certain groups. These (and related) assumptions are currently justified because even extremely powerful conventional computers cannot solve these problems with the best known algorithms.

However, hardness assumptions such as the discrete logarithm problem (or prime factorization), and thus the security of systems based on such assumptions, are threatened by the development of ever more powerful quantum computers. The reason is that thirty years ago Peter Shor [133] showed that quantum computers working with a sufficient number of controllable qubits could solve these problems efficiently.

Although the strength of existing quantum computers, as far as is publicly known, still seems to be far from solving these problems, it is necessary to develop and use alternative cryptographic schemes to prevent a future potential security and privacy disaster. For more than 20 years, so-called *post-quantum* cryptographic schemes have been developed, for example for PKE, commitments and NIZKPs, whose underlying hardness assumptions cannot be broken by known quantum algorithms.

There are several approaches to post-quantum cryptography, which differ in the type of hardness assumption they are based on. The most prominent approaches are lattice-based, code-based, hash-based, isogeny-based, and multivariate cryptography. The U.S. National Institute of Standards and Technology (NIST) launched a standardization program for post-quantum cryptography a few years ago and, after several rounds of evaluation, identified the first post-quantum methods (for key encapsulation mechanisms from which PKE schemes can be built and for digital signatures) to be standardized.

In the course of these developments, research has also been conducted on how to design verifiable online voting systems with post-quantum security. The challenge here is that the post-quantum cryptographic building blocks provide a different balance between security, speed, and data size than the conventional quantum-insecure schemes. Several solutions have been developed in the academic literature over the last 10 years:

- *Homomorphic aggregation*: There are two proposals for post-quantum online voting [21, 39] that combine malleable commitments (Section 4.3) with homomorphic aggregation (Section 4.4) and implement the corresponding cryptographic requirements using only lattice-based blocks. The voting protocol proposed in [21] is based on [39] and extends it to provide end-to-end verifiability.

- *Verifiable mix nets*: There are several proposals for post-quantum online voting [6, 5, 43, 75] that combine malleable PKE (Section 4.2) with verifiable mixing based on proofs of shuffle (Section 4.5) and implement the corresponding cryptographic requirements using only lattice-based blocks.

  Another solution was proposed in [20], in which the authors implement a decryption mix net made verifiable with the trip wire technique (Section 4.10.2) using only lattice-based PKE.

- *Special setting*: For elections in which the nodes of a peer-to-peer network want to hold an anonymous veto vote, a lattice-based solution was presented in [40].

Although some of these approaches are practicable, the development of post-quantum online voting systems is currently still in the basic research phase. It is therefore an open challenge to implement a complete, deployable post-quantum online voting system.

[22] The hiding property of Pedersen commitments is not affected as it is guaranteed unconditionally.

Remark 9: Post-quantum security vs everlasting privacy

At this point, we would like to clarify the relationship between post-quantum security and everlasting privacy (Section 3.2.2):

1. *Privacy*: Any voting system with everlasting privacy also guarantees post-quantum privacy, since everlasting privacy implies that even a computationally unbounded attacker cannot derive any information about the individual votes of the voters from the audit data. Conversely, post-quantum privacy does not imply everlasting privacy, since even post-quantum hardness assumptions can be broken by (theoretically) computationally unbounded attackers.

2. *Verifiability*: One cannot make a general statement about whether a voting system with everlasting privacy also provides post-quantum verifiability, and vice versa. Although it is in principle possible to combine these two properties, they are generally independent.

4.10.6 Tally-hiding voting

To reduce the amount of information published during an election, so-called *tally-hiding* voting protocols implement the desired result function exactly and securely without any detours. For example, if the result function is to announce the winner, then the data published during the election (including zero-knowledge proofs etc.) does not reveal any information other than which candidate received the most votes; in particular, it does not reveal the number of votes per candidate.

There are essentially two types of tally-hiding voting, which differ in terms of towards which parties the full tally (i.e., the number of votes per candidate) remains secret:

- *Fully tally-hiding*: In these systems, even the individual talliers who tally the election only learn the tally-hiding final result (e.g., only the winner). There are several proposals in the literature to realize verifiable fully tally-hiding online voting [33, 99, 137, 125, 66, 25]. While these solutions differ in the result functions they target, their efficiency, and their cryptographic components, they all have in common that they implement some kind of *secure multiparty computation (MPC)* protocol to verifiably compute the tally-hiding voting result in a distributed fashion.[23]

- *Publicly tally-hiding*: In these systems, the full tally remains hidden from anyone who has access to the bulletin board (e.g., to verify the election), while the individual talliers can learn the full tally (i.e., the number of votes per candidate), but not the individual votes of the voters. By relaxing the tally-hiding property in this way, it is possible to realize more complex voting methods, including ranked voting, as demonstrated in [76], which is the only verifiable publicly tally-hiding voting protocol in the literature to date.

There are two works in the literature that are somewhat related to tally-hiding voting, but have different goals:

- In [34], a secure MPC protocol was used to hide potentially sensitive information generated in the tallying phase of coercion-resistant voting protocols that employ fake credential systems (Section 4.10.4).

- In [126], a method was presented that reduces the level of verifiability (in a controlled manner) in order to reduce sensitive information in the final result.

[23] In *multiparty computation (MPC)* protocols, several parties jointly compute a function f whose input values $x_1, \ldots, x_n$ and all intermediate computation values remain secret, so that even the individual processing parties only learn the final function value $y = f(x_1, \ldots, x_n)$. *Secure* MPC protocols additionally guarantee that the correctness of the secret calculation can be verified, even when the processing parties are actively corrupted.

4.10.7 Special settings

Many voting protocols have been proposed for *peer-to-peer* (aka *boardroom*) online voting, in which the voters themselves tally their ballots [68, 83, 40, 13, 12, 14, 114, 131, 67, 122, 94, 93]. While such protocols can be interesting for these specific election types, they require that all (or most) voters actively participate in the tallying phase.

Some voting protocols enable voters to choose designated parties, so called *proxies*, to vote on their behalf (e.g., [91, 97]).

5 Conclusion

5.1 Summary

Secret ballots. Malleable public-key encryption (Section 4.2) and malleable commitments (Section 4.3) are efficient methods for encrypting voters' votes such that the ciphertexts keep these votes secret and such that they are compatible with any of the mechanisms for verifiably tallying the secret votes in a privacy-preserving manner (see below).

The main advantage of malleable public-key encryption over malleable commitments is that there are more high-quality reference implementations that have also been hardened by numerous real-world applications. The main advantage of malleable commitments over malleable public-key encryption is that they allow votes to be kept unconditionally private from any auditor, providing everlasting privacy without compromising verifiability (Remark 4).

Verifiable privacy-preserving tallying. Homomorphic aggregation (Section 4.4) and verifiable mixnets (Section 4.5) are efficient methods to tally secret ballots in such a way that the individual connections between voters and their votes remain secret in the final result, while at the same time it can be independently verified that the tallying was done correctly. There are high-quality reference implementations for both approaches, which also offer the possibility to distribute trust in terms of privacy among different tallying parties.

The main advantage of verifiable mixnets over homomorphic aggregation is that their efficiency is independent of the complexity of the ballots and can therefore be used for all types of voting methods. The main advantages of homomorphic aggregation over mixnets are that its efficiency is independent of the number of talliers and that there are no time dependencies between the tallying authorities.

Authenticated communication. Digital signatures (Section 4.6) are a simple and effective way to ensure that votes are cast by eligible voters. In particular, it can prevent votes from being "stuffed" (e.g., by unauthorized persons or in the name of abstaining voters). When digital signatures are used by voting authorities, this helps to strengthen accountability. Since digital signatures provide a strong form of individual identification of each encrypted ballot, caution is advised when using them if everlasting privacy is a concern. One cryptographic possibility to combine both properties is to use commitments (see paragraph secret ballots).

Voting device verification. There are different methods for verifying that the voting devices have processed the votes correctly, which offer diverse trade-offs between the assumptions they make about the infrastructure, the voters' possessions and their capabilities, and the specific verifiability and privacy features they provide. The main technical features of the three techniques that we have described and evaluated in more detail are as follows:

- *Audit-or-cast* (Section 4.7) is the simplest one to plug into an existing voting scheme, but provides only some probabilistic assurance of correctness. It requires an audit device.
- *Cast-and-audit* (Section 4.8) guarantees that the encrypted vote cast by the voting device is in fact the one that the voter chose, but to avoid that the individual audit data can serve as trivial

receipts it requires a cryptographically more involved interaction with the voting server. It requires an audit device.

- *Return codes* (Section 4.9) avoid the need for the voter to use an audit device and for the voting device to learn the voter's choice, but they require a more complex voting infrastructure.

Although there have been some studies on the usability of these methods, more research is needed to better understand under what practical circumstances these methods meet the desired usability properties and how they compare.

Combinations. As a basic building block of a verifiable voting system, any secret ballot method can be combined with any verifiable privacy-preserving tallying method that we investigated as part of this study. In this way, the correctness of the result can be independently verified while keeping the voters' choices secret towards the auditors.

The resulting basic building block can then be independently extended by the use of digital signatures and/or a method for verifying the voting devices. In this way, the authenticity of the communication can be improved and/or the correct functioning of the voting devices can be checked.

5.2 Key findings

Key Finding 1: The whole is more than the sum of its parts

Verifiable online voting systems combine many different methods and components. In order to assess whether an online voting system meets the desired characteristics, it is important to know what each method does, but ultimately it is key to look at the voting system as a whole. In particular, even if the individual methods work properly, they must be linked together correctly to provide end-to-end verifiability.

To illustrate Key finding 1, we recall that the effectiveness of voting device verification methods depends on the overall protocol in which they are embedded. For example, the methods we studied in Sections 4.7 and 4.8 can be ineffective if the voting protocol is vulnerable to clash attacks (Section 4.7.3). However, this problem cannot be detected if we only consider the respective mechanisms "locally", but we need to look at the whole system with all its specifications.

We would like to take this key finding as an opportunity to emphasize, as we did in Section 4.1, that the focus of our study was on individual (cryptographic) methods that can be combined to form the backend of verifiable online voting systems. We have explained which of these methods can be combined and how this works in principle. However, we have not investigated how to embed these methods into a complete online voting system and how to correctly and securely connect them at the implementation level. With this in mind, we would like to remind the reader of our warning from Section 4.1: *When analyzing the security of an online voting system, it is crucial to consider the entire system and not to draw any shortcuts!*

Key Finding 2: The goal is to reduce trust assumptions effectively

The ultimate goal of verifiable online voting systems with vote secrecy is to reduce the required trust in the various system components as much as possible. With respect to some properties (e.g., correct tallying), it is possible to completely avoid any trust in the responsible party, while for other properties (e.g., vote privacy), this is impossible. In the latter case, to avoid a single party being responsible for such properties, it is useful to distribute the role of that part among several entities, so that only some of these parties need to be trusted with respect to that property. For the distribution of trust to be effective in practice, it must be ensured that these parties are truly independent of each other.

To evaluate under which circumstances an election system is secure, one must first and foremost understand which components must be trusted in terms of verifiability and vote secrecy. One should keep in mind that in secure online elections, as in traditional paper ballot elections, it is impossible to completely avoid trusting any system component.

If an online voting system aims to distribute trust among multiple entities, then special care must be taken to ensure that these different entities are *truly independent*. Distributing trust for the sake of appearances is not enough. Depending on the role, this may mean, for example, that the parties are physically separated (e.g., in distributed mix nets), that their software comes from independent sources (e.g., in voting device verification) and that the providers are independent (e.g., economically, politically).

Key Finding 3: Distributed verifiable tallying of secret ballots is practical

We can expect any state-of-the-art verifiable online voting system to combine a secret ballot technique with a verifiable privacy-preserving tallying technique. In this way, we can allow independent auditors to verify the correctness of the election result, without having to trust the tallying authority.

Furthermore, it can be expected that the trust regarding vote privacy is distributed among multiple parties; in this way, only an (arbitrarily determinable) number of talliers must be trusted to ensure that the individual links between voters' encrypted votes and their choices in the final result remain secret. We remind the reader of Key finding 2, which here implies that the talliers must be effectively distributed.

With this key finding, we make clear that the combination of a secret ballot and a verifiable privacy-preserving tallying method should be the minimum standard for any verifiable online voting system. The methods for this purpose that we investigated are practical to implement and operate. Appropriate standardization of suitable building blocks and corresponding verification algorithms could help ensure that more voting systems meet this minimum standard.

Key Finding 4: No 'one-size-fits-all' solution for voting device verification

It is complicated to choose a voting device verification mechanism that is appropriate for a given election. The decision depends on what infrastructure is available, what assumptions we can make about the voters, what threat scenario we are considering, and what properties we want to achieve. Based on these conditions, we then have to decide on a voting device verification method that satisfies these conditions or, more realistically, at least provides a good balance between them.

Although choosing a method to verify voting devices is more difficult than choosing a privacy-preserving tallying method (see above), there are reasonable methods to solve this challenge in principle. Therefore, it is appropriate to expect that one of these methods will be used if there is a risk that the voting devices may be corrupted. Following Key finding 2, if a verification method is used that requires audit devices or other trusted parties, it is important to ensure that they and the voting device are provided by different entities and can be executed independently.

Key Finding 5: Everlasting privacy is feasible without compromising verifiability

In many elections, it is necessary to protect privacy not only in the foreseeable future, but also in the long run (for example against future adversaries that use quantum computers). There are feasible approaches to guarantee this property, called everlasting privacy, towards anyone who wants to verify the election, without compromising verifiability.

Everlasting privacy is especially relevant in the context of future quantum attackers. While such attackers cannot retroactively undermine the correctness of today's elections, they can retrospectively break privacy if the adversary learns the connections between the voters and their classically encrypted choices. Everlasting privacy provides effective protection against such 'store now, decrypt later' attacks (Remark 9).

In order to provide everlasting privacy towards the public, two approaches are currently used in practice. In the voting systems for political elections in Estonia or Switzerland, only a few selected auditors are given the opportunity to verify parts of the election. These auditors are then trusted not to violate everlasting privacy. Other voting systems, such as versions of Helios, use anonymous ballots, which means that the tallying process can also be verified by observers who may be able

to break the underlying hardness assumptions of the public-key encryption scheme used, but still cannot use this power to compromise vote privacy.

While these approaches have transparency drawbacks, using identifiable ballots with unconditionally hiding commitments instead can guarantee everlasting privacy towards the auditors under weaker trust assumptions for verifiability. Since we observed in our evaluation that this approach is in fact technically feasible, legal frameworks that allow this approach in their jurisdiction could help promote the market-ready development and deployment of online voting systems with everlasting privacy and fully independent end-to-end verifiability.

Bibliography

[1] C. Z. Acemyan, P. Kortum, M. D. Byrne, and D. S. Wallach. Usability of voter verifiable, end-to-end voting systems: Baseline data for Helios, Prêt à Voter, and Scantegrity II. *The USENIX Journal of Election Technology and Systems*, 2(3):26–56, 2014.

[2] C. Z. Acemyan, P. Kortum, M. D. Byrne, and D. S. Wallach. Users' Mental Models for Three End-to-End Voting Systems: Helios, Prêt à Voter, and Scantegrity II. In *International Conference on Human Aspects of Information Security, Privacy, and Trust*, pages 463–474. Springer, 2015.

[3] B. Adida. Helios: Web-based Open-Audit Voting. In P. C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 335–348. USENIX Association, 2008.

[4] M. Alsadi and S. Schneider. Verify my vote: Voter experience. 10 2020.

[5] D. F. Aranha, C. Baum, K. Gjøsteen, and T. Silde. Verifiable Mix-Nets and Distributed Decryption for Voting from Lattice-Based Assumptions. In W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 1467–1481. ACM, 2023.

[6] D. F. Aranha, C. Baum, K. Gjøsteen, T. Silde, and T. Tunge. Lattice-Based Proof of Shuffle and Applications to Electronic Voting. In K. G. Paterson, editor, *Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings*, volume 12704 of *Lecture Notes in Computer Science*, pages 227–251. Springer, 2021.

[7] R. Araújo, A. Barki, S. Brunet, and J. Traoré. Remote Electronic Voting Can Be Efficient, Verifiable and Coercion-Resistant. In J. Clark, S. Meiklejohn, P. Y. A. Ryan, D. S. Wallach, M. Brenner, and K. Rohloff, editors, *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, volume 9604 of *Lecture Notes in Computer Science*, pages 224–232. Springer, 2016.

[8] R. Araújo, N. B. Rajeb, R. Robbana, J. Traoré, and S. Yousfi. Towards Practical and Secure Coercion-Resistant Electronic Elections. In S. Heng, R. N. Wright, and B. Goi, editors, *Cryptology and Network Security - 9th International Conference, CANS 2010, Kuala Lumpur, Malaysia, December 12-14, 2010. Proceedings*, volume 6467 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 2010.

[9] R. Araújo and J. Traoré. A Practical Coercion Resistant Voting Scheme Revisited. In J. Heather, S. A. Schneider, and V. Teague, editors, *E-Voting and Identify - 4th International Conference, VoteID 2013, Guildford, UK, July 17-19, 2013. Proceedings*, volume 7985 of *Lecture Notes in Computer Science*, pages 193–209. Springer, 2013.

[10] Assembly Voting. Electa: Documentation of the cryptographic protocol (version 2.0). https://downloads.assembly-voting.com/download/marketing/electa_-_documentation_of_the_cryptographic_protocol.pdf. [Accessed 05-02-2024].

[11] M. Backes, M. Gagné, and M. Skoruppa. Using mobile device communication to strengthen e-voting protocols. In A. Sadeghi and S. Foresti, editors, *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*, pages 237–242. ACM, 2013.

[12] S. Bag, M. A. Azad, and F. Hao. E2E Verifiable Borda Count Voting System without Tallying Authorities. In *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019*, pages 11:1–11:9. ACM, 2019.

[13] S. Bag, M. A. Azad, and F. Hao. PriVeto: a fully private two-round veto protocol. *IET Inf. Secur.*, 13(4):311–320, 2019.

[14] S. Bag and F. Hao. E2E Verifiable Electronic Voting System for Shareholders. In *2019 IEEE Conference on Dependable and Secure Computing, DSC 2019, Hangzhou, China, November 18-20, 2019*, pages 1–8. IEEE, 2019.

[15] S. Bayer and J. Groth. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2012.

[16] J. Benaloh. Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. In R. Martinez and D. A. Wagner, editors, *2007 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07, Boston, MA, USA, August 6, 2007*. USENIX Association, 2007.

[17] J. C. Benaloh and M. Yung. Distributing the Power of a Government to Enhance the Privacy of Voters (Extended Abstract). In J. Y. Halpern, editor, *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing, Calgary, Alberta, Canada, August 11-13, 1986*, pages 52–62. ACM, 1986.

[18] D. J. Bernstein. Curve25519: new diffie-hellman speed records. In *Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9*, pages 207–228. Springer, 2006.

[19] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. 2023.

[20] X. Boyen, T. Haines, and J. Müller. A Verifiable and Practical Lattice-Based Decryption Mix Net with External Auditing. In L. Chen, N. Li, K. Liang, and S. A. Schneider, editors, *Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part II*, volume 12309 of *Lecture Notes in Computer Science*, pages 336–356. Springer, 2020.

[21] X. Boyen, T. Haines, and J. Müller. Epoque: Practical End-to-End Verifiable Post-Quantum-Secure E-Voting. In *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*, pages 272–291. IEEE, 2021.

[22] A. Brelle and T. Truderung. Cast-as-Intended Mechanism with Return Codes Based on PETs. In R. Krimmer, M. Volkamer, N. B. Binder, N. Kersting, O. Pereira, and C. Schürmann, editors, *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017, Bregenz, Austria, October 24-27, 2017, Proceedings*, volume 10615 of *Lecture Notes in Computer Science*, pages 264–279. Springer, 2017.

[23] Bundesverfassungsgericht (Germany). BVerfG, Urteil des Zweiten Senats vom 03. März 2009, March 2009.

[24] C. Burton, C. Culnane, J. Heather, T. Peacock, P. Y. A. Ryan, S. A. Schneider, V. Teague, R. Wen, Z. Xia, and S. Srinivasan. Using Prêt à Voter in Victoria State Elections. In J. A. Halderman and O. Pereira, editors, *2012 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '12, Bellevue, WA, USA, August 6-7, 2012*. USENIX Association, 2012.

[25] S. Canard, D. Pointcheval, Q. Santos, and J. Traoré. Practical Strategy-Resistant Privacy-Preserving Elections. In J. López, J. Zhou, and M. Soriano, editors, *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*, volume 11099 of *Lecture Notes in Computer Science*, pages 331–349. Springer, 2018.

[26] P. Chaidos, V. Cortier, G. Fuchsbauer, and D. Galindo. BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1614–1625. ACM, 2016.

[27] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[28] M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a Secure Voting System. In *2008 IEEE Symposium on Security and Privacy (SP 2008), 18-21 May 2008, Oakland, California, USA*, pages 354–368. IEEE Computer Society, 2008.

[29] J. D. Cohen and M. J. Fischer. A Robust and Verifiable Cryptographically Secure Election Scheme (Extended Abstract). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 372–382. IEEE Computer Society, 1985.

[30] V. Cortier, A. Filipiak, and J. Lallemand. BeleniosVS: Secrecy and Verifiability Against a Corrupted Voting Device. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, pages 367–381. IEEE, 2019.

[31] V. Cortier, D. Galindo, S. Glondu, and M. Izabachène. Distributed ElGamal à la Pedersen: Application to Helios. In A. Sadeghi and S. Foresti, editors, *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*, pages 131–142. ACM, 2013.

[32] V. Cortier, D. Galindo, R. Küsters, J. Müller, and T. Truderung. Sok: Verifiability notions for e-voting protocols. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 779–798. IEEE Computer Society, 2016.

[33] V. Cortier, P. Gaudry, and Q. Yang. A Toolbox for Verifiable Tally-Hiding E-Voting Systems. In V. Atluri, R. D. Pietro, C. D. Jensen, and W. Meng, editors, *Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part II*, volume 13555 of *Lecture Notes in Computer Science*, pages 631–652. Springer, 2022.

[34] V. Cortier, P. Gaudry, and Q. Yang. Is the JCJ voting system really coercion-resistant? *IACR Cryptol. ePrint Arch.*, page 430, 2022.

[35] Council of Europe. Code of good practice in electoral matters: guidelines and explanatory report, October 2002.

[36] Council of Europe. Recommendation CM/Rec(2017)51 of the Committee of Ministers to member States on standards for e-voting, June 2017.

[37] C. Culnane and S. A. Schneider. A Peered Bulletin Board for Robust Use in Verifiable Voting Systems. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 169–183. IEEE Computer Society, 2014.

[38] E. Cuvelier, O. Pereira, and T. Peters. Election Verifiability or Ballot Privacy: Do We Need to Choose? In J. Crampton, S. Jajodia, and K. Mayes, editors, *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, volume 8134 of *Lecture Notes in Computer Science*, pages 481–498. Springer, 2013.

[39] R. del Pino, V. Lyubashevsky, G. Neven, and G. Seiler. Practical Quantum-Safe Voting from Lattices. In B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1565–1581. ACM, 2017.

[40] J. Ding, D. Emery, J. Müller, P. Y. A. Ryan, and V. K. Wong. Post-Quantum Anonymous Veto Networks. *IACR Cryptol. ePrint Arch.*, page 1023, 2020.

[41] V. Distler, M.-L. Zollinger, C. Lallemand, P. Rønne, P. Ryan, and V. Koenig. Security - visible, yet unseen? how displaying security mechanisms impacts user experience and perceived security. 05 2019.

[42] C. C. Dragan, F. Dupressoir, K. Gjøsteen, T. Haines, P. B. Rønne, and M. R. Solberg. Machine-Checked Proofs of Accountability: How to sElect Who is to Blame. In G. Tsudik, M. Conti, K. Liang, and G. Smaragdakis, editors, *Computer Security - ESORICS 2023 - 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25-29, 2023, Proceedings, Part III*, volume 14346 of *Lecture Notes in Computer Science*, pages 471–491. Springer, 2023.

[43] V. Farzaliyev, J. Willemson, and J. K. Kaasik. Improved lattice-based mix-nets for electronic voting. *IET Inf. Secur.*, 17(1):18–34, 2023.

[44] P. Fauzi and H. Lipmaa. Efficient Culpably Sound NIZK Shuffle Argument Without Random Oracles. In K. Sako, editor, *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, volume 9610 of *Lecture Notes in Computer Science*, pages 200–216. Springer, 2016.

[45] P. Fauzi, H. Lipmaa, J. Siim, and M. Zajac. An Efficient Pairing-Based Shuffle Argument. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 97–127. Springer, 2017.

[46] P. Fauzi, H. Lipmaa, and M. Zajac. A Shuffle Argument Secure in the Generic Model. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 841–872, 2016.

[47] T. E. Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[48] H. Ge, S. Y. Chau, V. E. Gonsalves, H. Li, T. Wang, X. Zou, and N. Li. Koinonia: verifiable e-voting with long-term privacy. In D. Balenson, editor, *Proceedings of the 35th Annual Computer*

Security Applications Conference, ACSAC 2019, San Juan, PR, USA, December 09-13, 2019, pages 270–285. ACM, 2019.

[49] H. Ge, S. Y. Chau, V. E. Gonsalves, H. Li, T. Wang, X. Zou, and N. Li. Koinonia: verifiable e-voting with long-term privacy. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 270–285, 2019.

[50] R. Gennaro. Achieving Independence Efficiently and Securely. In J. H. Anderson, editor, *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing, Ottawa, Ontario, Canada, August 20-23, 1995*, pages 130–136. ACM, 1995.

[51] K. Gjøsteen, T. Haines, and M. R. Solberg. Efficient Mixing of Arbitrary Ballots with Everlasting Privacy: How to Verifiably Mix the PPATC Scheme. In M. Asplund and S. Nadjm-Tehrani, editors, *Secure IT Systems - 25th Nordic Conference, NordSec 2020, Virtual Event, November 23-24, 2020, Proceedings*, volume 12556 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 2020.

[52] K. Gjøsteen, T. Haines, and M. R. Solberg. Efficient mixing of arbitrary ballots with everlasting privacy: How to verifiably mix the ppatc scheme. In *Secure IT Systems: 25th Nordic Conference, NordSec 2020, Virtual Event, November 23–24, 2020, Proceedings 25*, pages 92–107. Springer, 2021.

[53] S. Glondu. Belenios specification. *Version 2.5*.

[54] O. Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001.

[55] O. Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[56] M. Graf, R. Küsters, D. Rausch, S. Egger, M. Bechtold, and M. Flinspach. Accountable Bulletin Boards: Definition and Provably Secure Implementation. *IACR Cryptol. ePrint Arch.*, page 1869, 2023.

[57] G. S. Grewal, M. D. Ryan, L. Chen, and M. R. Clarkson. Du-Vote: Remote Electronic Voting with Untrusted Computers. In C. Fournet, M. W. Hicks, and L. Viganò, editors, *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 155–169. IEEE Computer Society, 2015.

[58] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis. CHVote System Specification. *IACR Cryptol. ePrint Arch.*, page 325, 2017.

[59] T. Haines, R. Goré, and B. Sharma. Did you mix me? Formally Verifying Verifiable Mix Nets in Electronic Voting. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1748–1765. IEEE, 2021.

[60] T. Haines, R. Goré, and M. Tiwari. Machine-checking Multi-Round Proofs of Shuffle: Terelius-Wikstrom and Bayer-Groth. In J. A. Calandrino and C. Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 6471–6488. USENIX Association, 2023.

[61] T. Haines, S. J. Lewis, O. Pereira, and V. Teague. How not to prove your election outcome. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 644–660. IEEE, 2020.

[62] T. Haines and J. Müller. SoK: Techniques for Verifiable Mix Nets. In *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*, pages 49–64. IEEE, 2020.

[63] T. Haines and J. Müller. A Novel Proof of Shuffle: Exponentially Secure Cut-and-Choose. In J. Baek and S. Ruj, editors, *Information Security and Privacy - 26th Australasian Conference, ACISP 2021, Virtual Event, December 1-3, 2021, Proceedings*, volume 13083 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2021.

[64] T. Haines and J. Müller. Optimal Randomized Partial Checking for Decryption Mix Nets. In J. Baek and S. Ruj, editors, *Information Security and Privacy - 26th Australasian Conference, ACISP 2021, Virtual Event, December 1-3, 2021, Proceedings*, volume 13083 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2021.

[65] T. Haines, J. Müller, and I. Querejeta-Azurmendi. Scalable Coercion-Resistant E-Voting under Weaker Trust Assumptions. In J. Hong, M. Lanperne, J. W. Park, T. Cerný, and H. Shahriar, editors, *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, SAC 2023, Tallinn, Estonia, March 27-31, 2023*, pages 1576–1584. ACM, 2023.

[66] T. Haines, D. Pattinson, and M. Tiwari. Verifiable Homomorphic Tallying for the Schulze Vote Counting Scheme. In S. Chakraborty and J. A. Navas, editors, *Verified Software. Theories, Tools, and Experiments - 11th International Conference, VSTTE 2019, New York City, NY, USA, July 13-14, 2019, Revised Selected Papers*, volume 12031 of *Lecture Notes in Computer Science*, pages 36–53. Springer, 2019.

[67] F. Hao and P. Zielinski. A 2-Round Anonymous Veto Protocol. In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *Security Protocols, 14th International Workshop, Cambridge, UK, March 27-29, 2006, Revised Selected Papers*, volume 5087 of *Lecture Notes in Computer Science*, pages 202–211. Springer, 2006.

[68] L. Harrison, S. Bag, H. Luo, and F. Hao. VERICONDOR: End-to-End Verifiable Condorcet Voting without Tallying Authorities. In Y. Suga, K. Sakurai, X. Ding, and K. Sako, editors, *ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022*, pages 1113–1125. ACM, 2022.

[69] S. Hauser and R. Haenni. Modeling a Bulletin Board Service Based on Broadcast Channels with Memory. In A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala, editors, *Financial Cryptography and Data Security - FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers*, volume 10958 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2018.

[70] C. Hazay and Y. Lindell. Sigma protocols and efficient zero-knowledge. In *Efficient Secure Two-Party Protocols*, pages 147–175. Springer, 2010.

[71] J. Heather and D. Lundin. The Append-Only Web Bulletin Board. In P. Degano, J. D. Guttman, and F. Martinelli, editors, *Formal Aspects in Security and Trust, 5th International Workshop, FAST 2008, Malaga, Spain, October 9-10, 2008, Revised Selected Papers*, volume 5491 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2008.

[72] C. Hébant, D. H. Phan, and D. Pointcheval. Linearly-Homomorphic Signatures and Scalable Mix-Nets. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 597–627. Springer, 2020.

[73] S. Heiberg and J. Willemson. Verifiable internet voting in estonia. In R. Krimmer and M. Volkamer, editors, *6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014, Lochau / Bregenz, Austria, October 29-31, 2014*, pages 1–8. IEEE, 2014.

[74] L. Hirschi, L. Schmid, and D. A. Basin. Fixing the Achilles Heel of E-Voting: The Bulletin Board. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*, pages 1–17. IEEE, 2021.

[75] P. Hough, C. Sandsbråten, and T. Silde. Concrete NTRU Security and Advances in Practical Lattice-Based Electronic Voting. *IACR Cryptol. ePrint Arch.*, page 933, 2023.

[76] N. Huber, R. Küsters, T. Krips, J. Liedtke, J. Müller, D. Rausch, P. Reisert, and A. Vogt. Kryvos: Publicly Tally-Hiding Verifiable E-Voting. In H. Yin, A. Stavrou, C. Cremers, and E. Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1443–1457. ACM, 2022.

[77] V. Iovino, A. Rial, P. B. Rønne, and P. Y. A. Ryan. Using Selene to Verify Your Vote in JCJ. In M. Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Y. A. Ryan, V. Teague, A. Bracciali, M. Sala, F. Pintore, and M. Jakobsson, editors, *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*, volume 10323 of *Lecture Notes in Computer Science*, pages 385–403. Springer, 2017.

[78] Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts. Standard, International Organization for Standardization, Geneva, CH, Mar. 2018.

[79] M. Jakobsson, A. Juels, and R. L. Rivest. Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In D. Boneh, editor, *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*, pages 339–353. USENIX, 2002.

[80] W. Jamroga. Pretty Good Strategies for Benaloh Challenge. *CoRR*, abs/2307.03258, 2023.

[81] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In V. Atluri, S. D. C. di Vimercati, and R. Dingledine, editors, *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*, pages 61–70. ACM, 2005.

[82] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[83] D. Khader, B. Smyth, P. Y. A. Ryan, and F. Hao. A Fair and Robust Voting System by Broadcast. In M. J. Kripp, M. Volkamer, and R. Grimm, editors, *5th International Conference on Electronic Voting 2012, (EVOTE 2012), Co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC, July 11-14, 2012, Castle Hofen, Bregenz, Austria*, volume P-205 of *LNI*, pages 285–299. GI, 2012.

[84] S. Khazaei, T. Moran, and D. Wikström. A Mix-Net from Any CCA2 Secure Cryptosystem. In X. Wang and K. Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 607–625. Springer, 2012.

[85] S. Khazaei and D. Wikström. Randomized Partial Checking Revisited. In E. Dawson, editor, *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, volume 7779 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2013.

[86] A. Kiayias, A. Kuldmaa, H. Lipmaa, J. Siim, and T. Zacharias. On the Security Properties of e-Voting Bulletin Boards. In D. Catalano and R. D. Prisco, editors, *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 505–523. Springer, 2018.

[87] S. Kremer and P. B. Rønne. To Du or Not to Du: A Security Analysis of Du-Vote. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 473–486. IEEE, 2016.

[88] T. Krips and H. Lipmaa. More Efficient Shuffle Argument from Unique Factorization. In K. G. Paterson, editor, *Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings*, volume 12704 of *Lecture Notes in Computer Science*, pages 252–275. Springer, 2021.

[89] O. Kulyk, J. Henzel, K. Renaud, and M. Volkamer. Comparing "Challenge-Based" and "Code-Based" Internet Voting Verification Implementations. In *IFIP Conference on Human-Computer Interaction*, pages 519–538. Springer, 2019.

[90] O. Kulyk, J. Ludwig, M. Volkamer, R. E. Koenig, and P. Locher. Usable Verifiable Secrecy-Preserving E-Voting. In *6th International Joint Conference on Electronic Voting (E-Vote-ID)*. University of Tartu Press, 2021.

[91] O. Kulyk, K. Marky, S. Neumann, and M. Volkamer. Introducing Proxy Voting to Helios. In *11th International Conference on Availability, Reliability and Security, ARES 2016, Salzburg, Austria, August 31 - September 2, 2016*, pages 98–106. IEEE Computer Society, 2016.

[92] O. Kulyk, S. Neumann, K. Marky, J. Budurushi, and M. Volkamer. Coercion-Resistant Proxy Voting. In J. Hoepman and S. Katzenbeisser, editors, *ICT Systems Security and Privacy Protection - 31st IFIP TC 11 International Conference, SEC 2016, Ghent, Belgium, May 30 - June 1, 2016, Proceedings*, volume 471 of *IFIP Advances in Information and Communication Technology*, pages 3–16. Springer, 2016.

[93] O. Kulyk, S. Neumann, K. Marky, and M. Volkamer. Enabling Vote Delegation for Boardroom Voting. In M. Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Y. A. Ryan, V. Teague, A. Bracciali, M. Sala, F. Pintore, and M. Jakobsson, editors, *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*, volume 10323 of *Lecture Notes in Computer Science*, pages 419–433. Springer, 2017.

[94] O. Kulyk, S. Neumann, M. Volkamer, C. Feier, and T. Koster. Electronic voting with fully distributed trust and maximized flexibility regarding ballot design. In R. Krimmer and M. Volkamer, editors, *6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014, Lochau / Bregenz, Austria, October 29-31, 2014*, pages 1–10. IEEE, 2014.

[95] O. Kulyk, V. Teague, and M. Volkamer. Extending helios towards private eligibility verifiability. In R. Haenni, R. E. Koenig, and D. Wikström, editors, *E-Voting and Identity - 5th International Conference, VoteID 2015, Bern, Switzerland, September 2-4, 2015, Proceedings*, volume 9269 of *Lecture Notes in Computer Science*, pages 57–73. Springer, 2015.

[96] O. Kulyk and M. Volkamer. Usability is not enough: Lessons learned from'human factors in security'research for verifiability. *Cryptology ePrint Archive*, 2018.

[97] O. Kulyk and M. Volkamer. A Proxy Voting Scheme Ensuring Participation Privacy and Receipt-Freeness. In T. Bui, editor, *52nd Hawaii International Conference on System Sciences, HICSS 2019, Grand Wailea, Maui, Hawaii, USA, January 8-11, 2019*, pages 1–10. ScholarSpace, 2019.

[98] O. Kulyk, M. Volkamer, M. Müller, and K. Renaud. Towards Improving the Efficacy of Code-Based Verification in Internet Voting. In *VOTING*. Springer, 2020.

[99] R. Küsters, J. Liedtke, J. Müller, D. Rausch, and A. Vogt. Ordinos: A Verifiable Tally-Hiding E-Voting System. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 216–235. IEEE, 2020.

[100] R. Küsters, J. Müller, E. Scapin, and T. Truderung. sElect: A Lightweight Verifiable Remote Voting System. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 341–354. IEEE Computer Society, 2016.

[101] R. Küsters and T. Truderung. Security Analysis of Re-Encryption RPC Mix Nets. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 227–242. IEEE, 2016.

[102] R. Küsters, T. Truderung, and A. Vogt. Accountability: definition and relationship to verifiability. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 526–535. ACM, 2010.

[103] R. Küsters, T. Truderung, and A. Vogt. Clash Attacks on the Verifiability of E-Voting Systems. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 395–409. IEEE Computer Society, 2012.

[104] R. Küsters, T. Truderung, and A. Vogt. Formal Analysis of Chaumian Mix Nets with Randomized Partial Checking. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 343–358. IEEE Computer Society, 2014.

[105] R. Küsters and T. Wilke. *Moderne Kryptographie - Eine Einführung*. Vieweg + Teubner, 2011.

[106] A. Langley, M. Hamburg, and S. Turner. Elliptic Curves for Security. RFC 7748, Jan. 2016.

[107] Y. Lindell. Zero-Knowledge from Sigma Protocols–An Erratum, 2018. https://u.cs.biu.ac.il/~lindell/errata-zk-sigma.pdf; Online; accessed May 27, 2024.

[108] H. Lipmaa and B. Zhang. A More Efficient Computationally Sound Non-Interactive Zero-Knowledge Shuffle Argument. In I. Visconti and R. D. Prisco, editors, *Security and Cryptography for Networks - 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings*, volume 7485 of *Lecture Notes in Computer Science*, pages 477–502. Springer, 2012.

[109] W. Lueks, I. Querejeta-Azurmendi, and C. Troncoso. VoteAgain: A scalable coercion-resistant voting system. In S. Capkun and F. Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1553–1570. USENIX Association, 2020.

[110] K. Marky, O. Kulyk, and M. Volkamer. Comparative usability evaluation of cast-as-intended verification approaches in internet voting. 2018.

[111] K. Marky, V. Zimmermann, M. Funk, J. Daubert, K. Bleck, and M. Mühlhäuser. Improving the Usability and UX of the Swiss Internet Voting Interface. In *ACM CHI*, 2020.

[112] K. Marky, M. Zollinger, P. B. Roenne, P. Y. A. Ryan, T. Grube, and K. Kunze. Investigating Usability and User Experience of Individually Verifiable Internet Voting Schemes. *ACM Trans. Comput. Hum. Interact.*, 28(5):30:1–30:36, 2021.

[113] K. Marky, M.-L. Zollinger, P. Roenne, P. Y. Ryan, T. Grube, and K. Kunze. Investigating Usability and User Experience of Individually Verifiable Internet Voting Schemes. *ACM Trans. Comput.-Hum. Interact*, 28(5), 2021.

[114] P. McCorry, S. F. Shahandashti, and F. Hao. A Smart Contract for Boardroom Voting with Maximum Voter Privacy. In A. Kiayias, editor, *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*, volume 10322 of *Lecture Notes in Computer Science*, pages 357–375. Springer, 2017.

[115] D. Mestel, J. Müller, and P. Reisert. How efficient are replay attacks against vote privacy? A formal quantitative analysis. *J. Comput. Secur.*, 31(5):421–467, 2023.

[116] K. Morio and R. Künnemann. Verifying Accountability for Unbounded Sets of Participants. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*, pages 1–16. IEEE, 2021.

[117] J. Müller. Breaking and Fixing Vote Privacy of the Estonian E-Voting Protocol IVXV. In S. Matsuo, L. Gudgeon, A. Klages-Mundt, D. P. Hernandez, S. Werner, T. Haines, A. Essex, A. Bracciali, and M. Sala, editors, *Financial Cryptography and Data Security. FC 2022 International Workshops - CoDecFin, DeFi, Voting, WTSC, Grenada, May 6, 2022, Revised Selected Papers*, volume 13412 of *Lecture Notes in Computer Science*, pages 325–334. Springer, 2022.

[118] J. Müller, B. Pejó, and I. Pryvalov. DeVoS: Deniable Yet Verifiable Vote Updating. *Proc. Priv. Enhancing Technol.*, 2024(1):357–378, 2024.

[119] J. Müller and T. Truderung. A Protocol for Cast-as-Intended Verifiability with a Second Device. *CoRR*, abs/2304.09456, 2023.

[120] A. S. Neto, M. Leite, R. Araújo, M. P. Mota, N. C. S. Neto, and J. Traoré. Usability Considerations For Coercion-Resistant Election Systems. In M. Mota, B. S. Meiguins, R. O. Prates, and H. Candello, editors, *Proceedings of the 17th Brazilian Symposium on Human Factors in Computing Systems, IHC 2018, Belém, Brazil, October 22-26, 2018*, pages 40:1–40:10. ACM, 2018.

[121] S. Neumann, M. M. Olembo, K. Renaud, and M. Volkamer. Helios Verification: To Alleviate, or to Nominate: Is That the Question, or Shall we Have Both? In *International Conference on Electronic Government and the Information Systems Perspective*, pages 246–260. Springer, 2014.

[122] F. E. Orche, R. Géraud-Stewart, P. B. Rønne, G. Bana, D. Naccache, P. Y. A. Ryan, M. Biroli, M. Dervishi, and H. Waltsburger. Time, Privacy, Robustness, Accuracy: Trade-Offs for the Open Vote Network Protocol. In R. Krimmer, M. Volkamer, D. Duenas-Cid, P. B. Rønne, and M. Germann, editors, *Electronic Voting - 7th International Joint Conference, E-Vote-ID 2022, Bregenz, Austria, October 4-7, 2022, Proceedings*, volume 13553 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2022.

[123] C. Park, K. Itoh, and K. Kurosawa. Efficient Anonymous Channel and All/Nothing Election Scheme. In T. Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer, 1993.

[124] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[125] K. Ramchen, C. Culnane, O. Pereira, and V. Teague. Universally Verifiable MPC and IRV Ballot Counting. In I. Goldberg and T. Moore, editors, *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11598 of *Lecture Notes in Computer Science*, pages 301–319. Springer, 2019.

[126] P. Y. A. Ryan, P. B. Roenne, D. Ostrev, F. E. Orche, N. Soroush, and P. B. Stark. Who Was that Masked Voter? The Tally Won't Tell! In R. Krimmer, M. Volkamer, D. Duenas-Cid, O. Kulyk, P. B. Rønne, M. Solvak, and M. Germann, editors, *Electronic Voting - 6th International Joint Conference, E-Vote-ID 2021, Virtual Event, October 5-8, 2021, Proceedings*, volume 12900 of *Lecture Notes in Computer Science*, pages 106–123. Springer, 2021.

[127] P. Y. A. Ryan, P. B. Rønne, and V. Iovino. Selene: Voting with Transparent Verifiability and Coercion-Mitigation. In J. Clark, S. Meiklejohn, P. Y. A. Ryan, D. S. Wallach, M. Brenner, and K. Rohloff, editors, *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, volume 9604 of *Lecture Notes in Computer Science*, pages 176–192. Springer, 2016.

[128] P. Y. A. Ryan and S. A. Schneider. Prêt à Voter with Re-encryption Mixes. In D. Gollmann, J. Meier, and A. Sabelfeld, editors, *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings*, volume 4189 of *Lecture Notes in Computer Science*, pages 313–326. Springer, 2006.

[129] B. Schneier. *Applied cryptography - protocols, algorithms, and source code in C, 2nd Edition*. Wiley, 1996.

[130] C. Schnorr. Efficient Signature Generation by Smart Cards. *J. Cryptol.*, 4(3):161–174, 1991.

[131] S. F. Shahandashti and F. Hao. DRE-ip: A Verifiable E-Voting Scheme Without Tallying Authorities. In I. G. Askoxylakis, S. Ioannidis, S. K. Katsikas, and C. Meadows, editors, *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II*, volume 9879 of *Lecture Notes in Computer Science*, pages 223–240. Springer, 2016.

[132] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

[133] P. W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994.

[134] Swiss Post. Swiss Post voting system: System specification. version 1.4.0. Technical report, Swiss Post, Feb. 2024.

[135] B. Terelius and D. Wikström. Proofs of Restricted Shuffles. In D. J. Bernstein and T. Lange, editors, *Progress in Cryptology - AFRICACRYPT 2010, Third International Conference on Cryptology in Africa, Stellenbosch, South Africa, May 3-6, 2010. Proceedings*, volume 6055 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2010.

[136] M. Volkamer, O. Kulyk, J. Ludwig, and N. Fuhrberg. Increasing security without decreasing usability: Comparison of various verifiable voting systems. In *Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022)*, Boston, MA, Aug. 2022. USENIX Association.

[137] C. Wabartha, J. Liedtke, N. Huber, D. Rausch, and R. Küsters. Fully Tally-Hiding Verifiable E-Voting for Real-World Elections with Seat-Allocations. In G. Tsudik, M. Conti, K. Liang, and G. Smaragdakis, editors, *Computer Security - ESORICS 2023 - 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25-29, 2023, Proceedings, Part I*, volume 14344 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 2023.

[138] R. Wen and R. Buckland. Masked Ballot Voting for Receipt-Free Online Elections. In P. Y. A. Ryan and B. Schoenmakers, editors, *E-Voting and Identity, Second International Conference, VoteID 2009, Luxembourg, September 7-8, 2009. Proceedings*, volume 5767 of *Lecture Notes in Computer Science*, pages 18–36. Springer, 2009.

[139] D. Wikström. User manual for the verificatum mix-net version 3.1.0. *Verificatum AB, Stockholm, Sweden*, 2022.

[140] M. Zollinger, V. Distler, P. B. Rønne, P. Y. A. Ryan, C. Lallemand, and V. Koenig. User experience design for e-voting: How mental models align with security mechanisms. *CoRR*, abs/2105.14901, 2021.