Cryptographic protocol

Assembly Voting

September 2023

Documentation of the cryptographic protocol

Version 2.1

ASSEMBLY VOTING

Protect democracy. Prove integrity.

ASSEMBLY VOTING

Abstract

End-to-end verifiable voting systems attempt to establish elections where voters receive assurance that their votes have been cast correctly. At the same time, auditors can confirm that all ballots have been processed and tallied correctly. End-to-end verifiability in itself has many challenges, as one always has to find a balance between security and usability. An accessible digital election system based on an end-to-end verifiable voting protocol can potentially restore trust in democratic processes in society. At the same time, it enables voters to exercise their democratic rights from remote locations. This paper describes the path of Assembly Voting to achieve an end-to-end verifiable election protocol.

Contents

1 Introduction

This document presents all the technical details of the cryptographic protocol used in the Assembly Voting election solution. It describes the second version of the protocol of an end-to-end verifiable digital election system.

1.1 Prerequisites

There are no hard prerequisites beyond the knowledge specified in the intended audience section. However, it is recommended to review the System Architecture[**?**] and System Configuration[**?**] documents for a more comprehensive understanding of the system. Although this document is self-contained concerning the cryptographic processes required for the election protocol, the aforementioned documents provide broader context that may aid in understanding the entire system.

1.2 Intended audience

The document is primarily targeting cryptographers or technical, mathematical readers. This document is intended as an argumentation for the security claims we make about the election protocol. It contains plain language descriptions in the body of the document and mathematical descriptions of all algorithms are provided in the appendices.

1.3 Scope and Objectives

The election protocol incorporates several core features, including remote voting by voters, encryption of votes, the utilization of threshold cryptography, cryptographic shuffling of votes to ensure anonymity, verifiability of all essential processes, and the ability to perform audits on all system components throughout the election event. Although the system cannot always prevent fraud or unauthorized access, it can detect them.

The scope of this document is limited to the cryptographic protocol and the proof of the properties listed below. Other cybersecurity concerns, relating to system architecture and configuration, are addressed in separate documents. The protocol encompasses the entire election event, including election configuration, voter authorization, vote casting, tallying, and auditing. While cryptographic algorithms are essential for the security and auditing features of the system, many non-cryptographic processes are also necessary to conduct a safe election. This document describes an online election system accessed by users, i.e., election officials and voters, through a web browser or a native application on an internet-connected device such as a PC, laptop, tablet, smartphone, etc.

Throughout the election protocol, various components are tasked with generating cryptographic key pairs. It is incumbent upon each of these components to safeguard their respective private keys, ensuring their secrecy and security. The responsibility of the components to retain and protect their private keys is of paramount importance, given the potential vulnerabilities that might arise should they be compromised. Nonetheless, while significant care has been dedicated to the manner in which the system manages its keys, the details of this process, as well as the safeguards in place, are beyond the purview of this document. Comprehensive discussions on these key management strategies and related safety precautions can be found in accompanying documentation.

The primary objective of this document is to describe, claim, and argue the achievement of the following properties of our protocol, which are elaborated in greater detail in section 3:

- Voter Eligibility
- Voters Privacy
- Votes Anonymity
- Ballot Replacement
- Data Integrity
- Receipt-Freeness
- End-to-End Verifiability
- Vote & Go
- Replay Protection

1.4 Notation conventions

The following notation conventions are used throughout the document:

- use italic font Greek and Latin characters to display variables α, σ, x, y, z,
- use 1-based indexed arrays $\{a_1, ..., a_n\}$,
- generally, use n for the length of an array and ℓ for the height of a matrix,
- use the equal symbol to denote the structure of a variable $t = (x, y, z)$,
- use an arrow symbol on top of the variable to denote a vector $\vec{a} = \{a_1, ..., a_n\}$,
- generally, use letters i and j in subscript as indexes $a_i \in \vec{a}$,
- use regular font subscript to denote the context of a variable use x_{cnf},
- use double-struck font style to denote sets of elements $\mathbb{N}$, $\mathbb{Z}$,
- use superscript to denote the size of a vector $\vec{a} \in \mathbb{Z}^n$; no superscript implies size 1,
- use subscript $\mathbb{N}_q = \{0, 1, ..., q-1\}$ to define a subset with q elements,
- use symbol $\leftarrow$ to denote variable assignment $x \leftarrow 0$,

- use symbol $\in_{\mathrm{R}}$ to denote random assignment from a set $x \in_{\mathrm{R}} \mathbb{Z}$,
- use calligraphic font style to denote an actor in the protocol $\mathcal{T}$,
- use bold calligraphic font style to denote a set of actors $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, ..., \mathcal{V}_n\}$,
- use san-serif font style to declare algorithms $\mathsf{Algorithm}(x, y)$,
- use typewriter font style to declare protocols `Protocol`,
- use symbol $\mathcal{H}$ to denote a hash function,
- in elliptic curve context, use lower case letter for scalars and upper case for point variables q, G,
- in elliptic curve context, use notation $[x]G$ as point multiplication
- generally, use the notation (x, Y) to denote private-public key pairs and mark them with subscripts for specific contexts,
- generally, assume elliptic curve domain parameters (p, a, b, G, q, h) known and available to all algorithms and protocols.

2 Solution entities

This section describes the different concepts seen throughout the document.

2.1 Actors

2.1.1 Election Administrator

The Election Administrator Component, denoted as $\mathcal{E}$, serves as a pivotal point in the cryptographic election process. Hosted by a designated organization, it is an online application through which all election officials interact for the purposes of setting up, configuring, and updating an election event.

The primary responsibility of $\mathcal{E}$ is to lay down the foundation for the election by determining its configurations. It is paramount for election officials to access this service, and this is facilitated through a pre-defined set of authentication credentials ensuring security and integrity.

To ensure the authenticity and integrity of the configurations, $\mathcal{E}$ is equipped with a key pair: a public key $Y_{\mathcal{E}}$ and a corresponding private key $x_{\mathcal{E}}$. The private key is of particular importance and is securely stored by the election administrator to prevent unauthorized access or malicious manipulations. Configurations are signed using this key pair, assuring their genuineness and credibility.

2.1.2 Digital Ballot Box

The Digital Ballot Box, $\mathcal{D}$, is the central communication unit interacting with all other parties. It is publicly accessible via the internet and contains a bulletin board with all public information about an election, categorized into configuration data, voting data, and result data. The Digital Ballot Box owns a key pair $(x_{\mathcal{D}}, Y_{\mathcal{D}})$ used for signing data on the bulletin board and is responsible for privately storing its private key $x_{\mathcal{D}}$.

2.1.3 Trustees

Trustees, denoted as $\mathcal{T}_i$, where $i \in \{1, ..., n_{\text{t}}\}$ and n_{t} is the total number of trustees, are election officials responsible for preserving the secrecy of voting data and actively participating in result computation. They use the trustee application to perform all cryptographic processes involved in the protocol and are responsible for building the election encryption key, safely storing their shares of the decryption key, and destroying the keys after the election event has ended.

2.1.4 Voters

Predefined voters, denoted as $\mathcal{V}_i$, where $i \in \{1, ..., n_\text{v}\}$ and n_v is the total number of voters, generate all voting data using the voting application. They own credentials that authorize them to cast a digital ballot and have an interest in verifying that their vote is processed correctly and included in the final tally. During the protocol, the voting application generates a key pair (x_i, Y_i), representing the cryptographic identity of voter $\mathcal{V}_i$.

2.1.5 Voter Authorizer

The Voter Authorizer, $\mathcal{A}$, is responsible for authorizing a Voter $\mathcal{V}$ after being authenticated with all Identity Providers and for preserving the election eligibility property. It owns a key pair $(x_\mathcal{A}, Y_\mathcal{A})$ used for signing voter authorization and is responsible for privately storing its private key $x_\mathcal{A}$.

2.1.6 Identity Provider

Identity Providers, denoted as $\mathcal{I}_i$, where $i \in \{1, ..., n_\text{i}\}$ and n_i is the total number of identity providers, are third-party applications responsible for authenticating a voter $\mathcal{V}$ during the election phase and must follow the OIDC protocol.

2.1.7 Credential Authority

Credential Authorities, denoted as $\mathcal{C}_i$, where $i \in \{1, ..., n_\text{c}\}$, are third-party institutions responsible for generating and distributing voter credentials, which are necesarry for the voter authorization proces. It is recommended that each Credential Authority use a different communication channel for credential distribution, e.g., e-mail, post, SMS.

2.1.8 External Verifier

The External Verifier, $\mathcal{X}$, is an auditing tool used by voters to perform ballot verification processes (section 4.8). It allows voters to check that their vote has been correctly encrypted and stored on the bulletin board. During this process, the External Verifier generates a new key pair $(x_\mathcal{X}, Y_\mathcal{X})$ and must protect its private key $x_\mathcal{X}$.

2.1.9 Auditors

Public auditors, possessing the proper auditing tools, can audit the election process without having an active role in the election process or the election

system noticing. Most of the time, auditing happens without the election system noticing.

2.2 Public bulletin board

The digital ballot box publishes all election events as items on a publicly accessible bulletin board. Each item on the board is authored by a relevant actor and describes a specific event, uniquely identified by its *address* (i.e., the *hash value* of the item). The *address* of the last item represents the *board hash value* at that moment. All events are stored in an *append-only list*, ensuring no event can be removed or replaced, and new events are appended at the end. This structure was inspired by *The Append-Only Web Bulletin Board*[1].

Our approach differs from *The Append-Only Web Bulletin Board*[1] in that to append a new item, the writer must include the address of any existing board item as part of the new item, instead of only the previous item. This reference is called the *parent* item. The digital ballot box $\mathcal{D}$ computes the new item's address by hashing its content (including the parent item reference) concatenated with the current board hash value, the address of the previous item, and a registration timestamp. $\mathcal{D}$ then signs the new item's address and delivers it to the writer as proof of the new item's acceptance on the board. This ensures the new item is linked to the previous one.

As a result, each bulletin board item references two other items:

- an existing item chosen by the writer as the parent item,
- and the previous item on the board.

This modification ensures the digital ballot box protects the *history* property described in *The Append-Only Web Bulletin Board*[1]. Additionally, a new *ancestry* property is introduced, defined by items being meaningfully related. Thus, traversing the *ancestry* line reveals a tree-like structure, while the *history* line appears linear.

We also introduce a *hidden verification track* for the ballot checking process described in section 4.8. It is:

- *hidden* because it is not publicly available on the bulletin board, but can be requested using the *address* of a specific item,
- *verification* because it is used only for ballot checking,
- *track* because it spawns an extra *history* of events injected under a specific item from the main *history*.

Consequently, any i^{th} bulletin board item b_i consists of the following structure $(m_i, c_i, \mathcal{W}, \sigma_i, t_i, p_i, h'_i, h_i)$, where:

- m_i is the item type,

- c_i is the content describing the event,
- $\mathcal{W}$ is a reference to the item writer,
- σ_i is the writer's signature,
- p_i is the parent item address, with $p_i \in \{h_1, ..., h_{i-1}\}$,
- h'_i is the previous item address in the *history* (i.e., $h'_i = h_{i-1}$),
- t_i is the registration timestamp,
- h_i is the item address.

Different item types (i.e., possible values of m_i) and the events they represent are detailed in appendix A. Rules about how items can reference parent items and which actors can write them are outlined there.

To write a new item on the bulletin board, a writer must follow the protocol described in section 4.1.1. The following actors are allowed to write on the bulletin board:

- Election Administrator $\mathcal{E}$, responsible for writing all election configuration events.
- Voter Authorizer $\mathcal{A}$, authorizing voters to interact with the digital ballot box based on successful authentication.
- Voters $\mathcal{V}_i$, with $i \in \{1, ..., n_\text{v}\}$, writing all vote-related events of an election.
- Digital Ballot Box $\mathcal{D}$, ultimately accepting all events published on the bulletin board and writing all auxiliary events supporting the voting process.
- External Verifier $\mathcal{X}$, writing events related to the ballot checking process on the hidden verification track of the bulletin board.

2.3 Voter Authorization

2.3.1 Vote Submission Authorization - identity based

During the pre-election phase, the voter authorizer service $\mathcal{A}$ lists a set of third-party identity providers $\boldsymbol{\mathcal{I}} = \{\mathcal{I}_1, ..., \mathcal{I}_{n_\text{i}}\}$, where n_i is the number of them. Then, the voter authorizer service is loaded with a list of eligible voters $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, ..., \mathcal{V}_{n_\text{v}}\}$, where n_v is the total number of voters. Each voter $\mathcal{V}_i$ is defined by a set of identities supported by each of the identity providers $\boldsymbol{\mathcal{I}}$.

To be authorized to cast a vote on the bulletin board, a user has to authenticate to all identity providers $\boldsymbol{\mathcal{I}}$ and receive identity tokens. Then, the user submits them to the voter authorizer service which checks whether all identities match a voter $\mathcal{V}_i \in \boldsymbol{\mathcal{V}}$. If successful, the voter $\mathcal{V}_i$ is authorized to cast a ballot on the bulletin board. The process is further described in section 4.7.1.

Once authenticated and authorized, voter $\mathcal{V}_i$ can interact directly with the digital ballot box in the voting protocol as described in section 4.7.4.

2.3.2 Vote Submission Authorization - credential based

During the pre-election phase, the voter authorizer service $\mathcal{A}$ interacts with a set of Credential Authorities $\boldsymbol{\mathcal{C}} = \{\mathcal{C}_1, ..., \mathcal{C}_{n_c}\}$ to generate and distribute voter credentials, as described in section 4.5. The voter authorizer is loaded with the eligible voters $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, ..., \mathcal{V}_{n_v}\}$, where each voter $\mathcal{V}_i$ is defined by contact information for each communication channel used by all credential authorities.

To be authorized to cast a vote on the bulletin board, a user has to provide all credentials received from each authority and convert them into proofs of credentials. Then, the user submits the proofs to the voter authorizer service which checks whether they match a voter $\mathcal{V}_i \in \boldsymbol{\mathcal{V}}$. If successful, the voter $\mathcal{V}_i$ is authorized to cast a ballot on the bulletin board. The process is further described in section 4.7.1.

Once authenticated and authorized, voter $\mathcal{V}_i$ can interact directly with the digital ballot box in the voting protocol as described in section 4.7.4.

2.4 Threshold cryptography

Threshold cryptography offers an innovative approach to cryptographic key management and sensitive operations. In a traditional cryptographic system, a singular entity possesses the cryptographic key, which if compromised, can jeopardize the entire system. Threshold cryptography, however, divides a secret (such as a private key) among multiple parties, termed trustees. In order for a sensitive operation to be executed or the secret to be reconstructed, a predefined number of these trustees, the threshold, must cooperate. This ensures that no single trustee has absolute power or knowledge of the entire secret. By distributing the trust among several entities, the vulnerability associated with a single point of failure is greatly reduced.

In the context of our election system, threshold cryptography offers numerous advantages. First and foremost, it ensures that no singular entity has control over sensitive election operations, thereby enhancing security and trustworthiness. Additionally, it provides flexibility and resilience; for instance, if one or more trustees are unavailable or their components are compromised, the election process can still proceed as long as the threshold number of trustees are operational. This not only improves the robustness of the election system but also allows election officials to manage risks more effectively by setting an appropriate threshold.

3 Properties

The protocol exhibits the following properties, which hold true under current known conditions. However, these properties depend on certain assumptions about the system configuration and the capabilities of potential attackers or malicious actors. General assumptions that apply to all properties:

- An attacker computational power is assumed to be polynomially bounded.
- The elliptic curve discrete logarithm problem is assumed to be infeasible to break, as described in *Guide to Elliptic Curve Cryptography* [2].

3.1 Voter Eligibility

The voter eligibility property is defined as the fact that only a limited number of predefined voters can cast a valid vote. The property requires the following assumptions:

- There is at least one honest third-party identity provider that generates genuine identity tokens upon successful voter authentication.
- The administration auditing process (section 5.2) is trustworthy, i.e., an honest election official runs genuine auditing tools on real election data.

3.2 Votes Privacy

The votes privacy property implies that no entity can read a partial result or any votes before the intended time. This is to prevent influencing subsequent voters throughout the election period, as voters' initial intentions may change if the current results were publicized. The property requires the assumption that:

- There are no more than t malicious trustees or compromised trustee applications, where t is the decryption threshold configured during the threshold ceremony (section 4.6).

3.3 Votes Anonymity

The votes anonymity property implies that no single entity can determine how a particular voter voted. The property is reached on the following assumptions:

- The ballot marking application does not leak voter secret information.
- There is at least one honest trustee that participate in the mixing phase of the result computation process.

3.4 Ballot Replacement

Ballot replacement allows voters to overwrite their votes, enabling them to vote multiple times. However, only the most recent vote submitted by a voter will be taken into account in the final tally.

3.5 Data Integrity

The data integrity property implies mechanisms for verifying the origin and authenticity of the data published on the bulletin board. The property requires the assumption that:

- The integrity audit (section 5.3.1) is trustworthy, i.e., an honest election official runs genuine auditing tools against real election data.

3.6 Receipt-Freeness

The receipt-freeness property is defined as the fact that voters cannot prove to a third party how they voted after they submitted the encrypted ballot.

3.7 End-to-End Verifiability

End-to-end verifiability is a set of mechanisms for verifying the correct processing of votes, consisting of three verification steps: cast as intended, registered as cast, and counted as registered. The property is reached on the following assumptions:

- A voter uses at least one trusted device, e.g., either the voting application device or the external verifier device.
- There are multiple external verifier deployments, out of which at least one is considered trustworthy by the voter.

3.8 Vote & Go

The Vote & Go approach stipulates that voters need to actively participate only during the voting phase, while the computation of results can be conducted independently without voter interaction.

3.9 Replay Protection

Replay protection is a security measure that prevents an adversary from capturing and replaying a valid set of actions or data to compromise a system.

It ensures that any captured information or actions cannot be reused to gain unauthorized access or perform malicious activities.

4 Election protocol

This section describes the entire election protocol which is split into three main phases:

- the pre-election phase, where the election context is created and all components are configured,
- the election phase, where the actual votes are being generated and stored,
- and the post-election phase, where the election result is finalized and audits are performed on system components.

All of these phases consist of different processes that are triggered by specific stakeholders. A map of all processes is presented in figure 1, where the leftmost label lists the process name, the circled label defines the actor that triggers the process (EO for election official, T for trustee, V for voter and PA for public auditor), and the following empty circles indicate the system components that are involved in the process.

The election process is started by an election official initializing a digital ballot box, as presented in section 4.2, and setting up the election configuration as in section 4.3. Then, the trustees perform the threshold ceremony, as described in section 4.6. In case voter authorization mode is credential-based, during the pre-election phase, voters receive their credentials as described in section 4.5.

During the election phase, voters can get authorized to cast a vote and then perform the voting process as described in section 4.7. Optionally, voters can perform a verification of their encrypted ballots, as described in section 4.8. More about voter-specific auditing is presented in section 5.1.

In the post-election phase, election officials run an administration auditing process to check that the election system behaved correctly, as described in section 5.2. Any public auditor can run a public auditing process on the entire election process, as presented in section 5.3. Also, during this time, trustees can compute the election result, as presented in section 4.9.

All the processes listed above involve appending data on the public bulletin board in the form of board items. The protocol for writing an item on the bulletin board is described in section 4.1.1.

Figure 1: Processes map

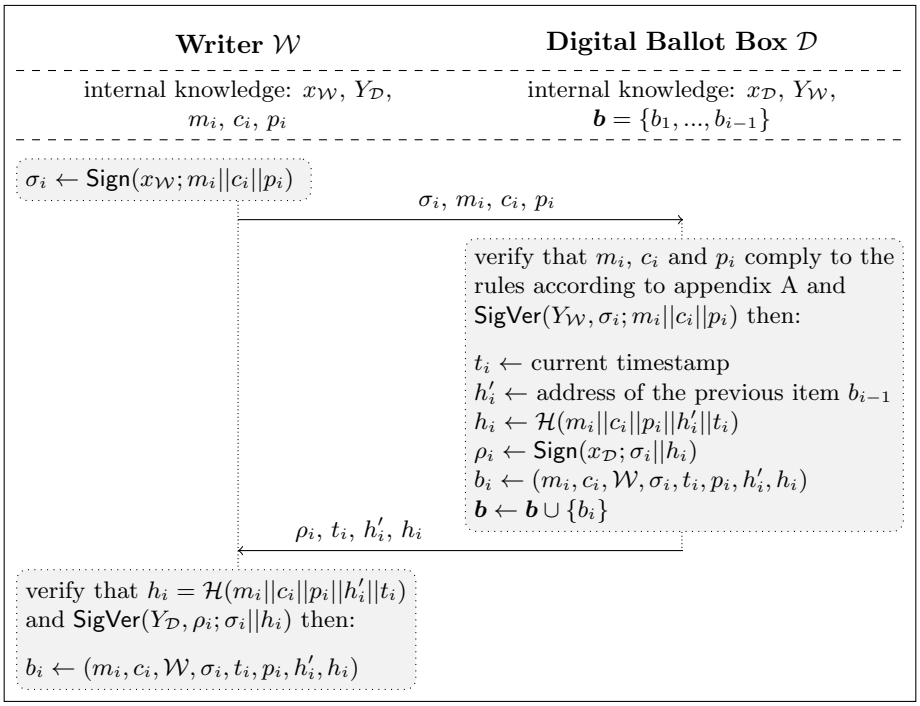
4.1 Interactions with the Digital Ballot Box

4.1.1 Writing on the bulletin board

The election protocol allows a predefined set of actors to write events on the bulletin board. They are described in section 2.1 and consist of:

- the Election Administrator $\mathcal{E}$,
- the Digital Ballot Box $\mathcal{D}$,
- the Voter Authorizer $\mathcal{A}$,
- each Voter $\mathcal{V}_i$,
- and the External Verifier $\mathcal{X}$.

For the sake of generalization, protocol 2 presents the interaction between a generic writer $\mathcal{W}$ and the digital ballot box $\mathcal{D}$ necessary for publishing the i^{th} item on the bulletin board. We define $(b_i, \rho_i) \leftarrow \texttt{WriteOnBoard}(\mathcal{W}, m_i, c_i, p_i)$ as the interaction of $\mathcal{W}$ and $\mathcal{D}$ that outputs a new item b_i and its receipt ρ_i.



Protocol 2: $\texttt{WriteOnBoard}(\mathcal{W}, m_i, c_i, p_i)$

The publicly available information consists of: the public key of the writer $Y_{\mathcal{W}}$, the public key of the digital ballot box $Y_{\mathcal{D}}$ and all of the existing items on the

bulletin board $\boldsymbol{b} = \{b_1, ...b_{i-1}\}$. The writer has his private key $x_{\mathcal{W}}$, while the digital ballot box knows its private key $x_{\mathcal{D}}$.

The protocol starts with the writer actively choosing the event type m_i and the content c_i to be appended on the bulletin board as the i^{th} item. Next, the writer chooses a pre-existing item on the bulletin board as the parent of the new item. The parent item is referenced by its address $p_i \in \boldsymbol{h}$, where $\boldsymbol{h}$ is the set of all addresses of the pre-existing board items $\boldsymbol{b}$. All event types, the contained data, and the choice of their parent item are described in the appendix A.

The writer signs with his private key $x_{\mathcal{W}}$ the concatenation of the new item type, the content, and the parent address. The signature $\sigma_i \leftarrow \mathsf{Sign}(x_{\mathcal{W}}; m_i||c_i||p_i)$ (algorithm 28) is sent with the item type m_i, content c_i and parent address p_i to the digital ballot box as a request to append a new item on the board.

The digital ballot box verifies whether m_i, c_i, and p_i are chosen according to the rules specified in appendix A and whether the request has a valid signature. If all validations succeed, it computes the address of the new item h_i by hashing a concatenation of the type of the new item m_i, its content c_i, its parent address p_i, the current board hash value $h'_i = h_{i-1}$, and the registration timestamp t_i. It then stores the new item on the bulletin board as item $b_i = (m_i, c_i, \mathcal{W}, \sigma_i, t_i, p_i, h'_i, h_i)$, where $\mathcal{W}$ is a reference to the writer.

The digital ballot box signs with its private key $x_{\mathcal{D}}$ the concatenation of the writer's signature σ_i and the address of the new item h_i. The resulting signature ρ_i is sent together with the registration timestamp t_i, the new board hash value h_i, and the previous board hash value h'_i to the writer as proof that the item has been appended on the board. Finally, the writer verifies that the address of the new item is computed correctly and that the response has a valid signature.

Note that, when the protocol is performed by a specific writer, e.g., voter $\mathcal{V}_i$, the writer's key pair $(x_{\mathcal{W}}, Y_{\mathcal{W}})$ will be replaced by the voter's key pair (x_i, Y_i).

4.1.2 Verifying bulletin board items

Because of the two bulletin board properties listed in section 2.2, namely *history* and *ancestry*, there exist two auditing algorithms. Given a list of items $\boldsymbol{b} = \{b_1, ..., b_n\}$, an auditor can run $\mathsf{AncestryVer}(\boldsymbol{b}, h_0)$ (algorithm 1) to check the *ancestry* of $\boldsymbol{b}$, where h_0 is the parent of b_1. Similarly, the auditor can run $\mathsf{HistoryVer}(\boldsymbol{b}, h_0)$ (algorithm 2) to check the *history* property of $\boldsymbol{b}$, where h_0 is the address of the previous item of b_1.

In addition, we define $\mathsf{ItemVer}(b, Y_{\mathcal{W}})$ (algorithm 3) as a publicly available auditing algorithm to check the integrity of any bulletin board item b against its writer's public key $Y_{\mathcal{W}}$.

Algorithm 1: AncestryVer($\boldsymbol{b}, h_0$)

Data: The ancestry of board items $\boldsymbol{b} = \{b_1, ..., b_n\}$, with
$b_i = (m_i, c_i, \mathcal{W}, \sigma_i, t_i, p_i, h'_i, h_i)$ and $p_i, h'_i, h_i \in \mathbb{B}^{256}$, where $i \in \{1, ..., n\}$
The address of the parent of the ancestry $h_0 \in \mathbb{B}^{256}$

```
for i ← 1 to n by 1 do
    if h_i ≠ H(m_i||c_i||p_i||h'_i||t_i)
    or p_i ≠ h_{i−1} then
        return 0                                   // ancestry is invalid
    end
end
return 1                                           // ancestry is valid
```

Algorithm 2: HistoryVer($\boldsymbol{b}, h_0$)

Data: The history of board items $\boldsymbol{b} = \{b_1, ..., b_n\}$, with
$b_i = (m_i, c_i, \mathcal{W}, \sigma_i, t_i, p_i, h'_i, h_i)$ and $p_i, h'_i, h_i \in \mathbb{B}^{256}$, where $i \in \{1, ..., n\}$
The address of the previous item in the history $h_0 \in \mathbb{B}^{256}$

```
for i ← 1 to n by 1 do
    if h_i ≠ H(m_i||c_i||p_i||h'_i||t_i)
    or h'_i ≠ h_{i−1} then
        return 0                                   // history is invalid
    end
end
return 1                                           // history is valid
```

Algorithm 3: ItemVer($b, Y_{\mathcal{W}}$)

Data: The board item $b = (m, c, \mathcal{W}, \sigma, t, p, h', h)$
The public key of the writer $Y_{\mathcal{W}}$

```
if h = H(m||c||p||h'||t)
and SigVer(Y_W, σ; m||c||p)                        // algorithm 29
 then
    return 1                                       // item is valid
else
    return 0                                       // item is invalid
end
```

4.2 Bulletin board initialization

An election official selects the elliptic curve domain parameters (p, a, b, G, q, h) from a predefined set, listed in appendix B.1.1. Based on these parameters, the election administrator service generates a new key pair $(x_{\mathcal{E}}, Y_{\mathcal{E}}) \leftarrow \mathsf{KeyGen}()$ (algorithm 16), where $x_{\mathcal{E}}$ is its signing key and will be kept secret throughout the election, while $Y_{\mathcal{E}}$ is its public signature verification key. Next, the election administrator requests the digital ballot box to initialize a new bulletin board with the initial election meta-data configuration (including the elliptic curve domain parameters and the public key $Y_{\mathcal{E}}$).

On this request, the digital ballot box generates a new key pair $(x_{\mathcal{D}}, Y_{\mathcal{D}}) \leftarrow \mathsf{KeyGen}()$ (algorithm 16), where $x_{\mathcal{D}}$ is its signing key and will be kept secret throughout the election period, and $Y_{\mathcal{D}}$ is its public signature verification key. From this, it spawns a new bulletin board by generating a genesis item as the first item of the board $(b_1, \rho_1) \leftarrow \texttt{WriteOnBoard}(\mathcal{D}, m_1, c_1, p_1)$ (protocol 2), where $m_1 =$ "genesis", $p_1 = \varnothing$, and the content c_1 is constructed according to the rules specified in appendix A. Next, the digital ballot box returns to the election administrator with the freshly created genesis item b_1. From this point on, the election administrator service and the digital ballot box represent identities $\mathcal{E}$ and $\mathcal{D}$ respectively on the bulletin board.

4.3 Election configuration

4.3.1 Election configuration

Once a bulletin board exists, an election official can interact with the election administrator $\mathcal{E}$ to write all of the configuration items on the bulletin board by following $\texttt{WriteOnBoard}(\mathcal{E}, m_i, c_i, p_i)$ (protocol 2). All items are computed and published one by one, based on the rules defined in appendix A. All items are signed with the election administrator signing key $x_{\mathcal{E}}$ and they reference the address of the previous configuration item ($p_i = h_{i-1}$) as a parent. The items which make up the initial configuration are:

- the election configuration item, containing
  - election title
- contest configuration items for each contest containing
  - the contest identifier,
  - marking rules,
  - result rules,
  - list of candidates $\{m_1, ..., m_{n_c}\}$, with each $m_i \in \mathbb{B}^*$.

4.3.2 Voting rounds configuration

An election official interacts with the election administrator service to define when the election phase will take place, namely by setting a start and end date. Then, the election administrator $\mathcal{E}$ writes a voting round item on the bulletin board by following $\texttt{WriteOnBoard}(\mathcal{E}, m_i, c_i, p_i)$ (protocol 2) based on the rules from appendix A, specifying the start and end date, and the enabled contests.

For a regular election, a single voting round is sufficient. However, multiple voting rounds can be configured to start at different times, and various contests could be enabled in each voting round.

4.4 Actor authorization

For each actor, the Election Administrator service interacts with the other services, e.g., the voter authorizer, to generate its own key pair $(x_{\mathcal{A}}, Y_{\mathcal{A}}) \leftarrow \mathsf{KeyGen}()$ (algorithm 16). Value $x_{\mathcal{A}}$ is the voter authorizer signing key and will be kept secret throughout the election period, while $Y_{\mathcal{A}}$ is its public signature verification key and is shared with the election administrator. Next, the election administrator $\mathcal{E}$ writes an actor configuration item on the bulletin board by following $\texttt{WriteOnBoard}(\mathcal{E}, m_i, c_i, p_i)$ (protocol 2) based on the rules from appendix A. The item contains: the actor identifier, the actor's public key $Y_{\mathcal{A}}$, and the actor role, e.g. "voter authorizer".

Once the actor configuration item is published on the bulletin board, the voter authorizer becomes identity $\mathcal{A}$ and can interact with the digital ballot box.

4.4.1 Voter authorization configuration

An election official interacts with the Voter Authorizer service to configure the voter authentication process and to provide the list of eligible voters $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, ..., \mathcal{V}_{n_\mathcal{V}}\}$.

Additionally, if the voter authorization mode is identity-based, the election official selects a list of third-party identity providers $\boldsymbol{\mathcal{I}} = \{\mathcal{I}_1, ..., \mathcal{I}_{n_\mathcal{I}}\}$ used for voter authentication. Note that each voter $\mathcal{V}_i$ is defined by a unique identifier and a list of identities supported by all of the identity providers.

Otherwise, if the voter authorization mode is credential-based, each voter $\mathcal{V}_i$ is defined by a unique identifier and a list of contact information supported by each Credential Authority in $\boldsymbol{\mathcal{C}}$.

Finally, the voter authorizer writes the voter authorization configuration item on the bulletin board by following $\texttt{WriteOnBoard}(\mathcal{A}, m_i, c_i, p_i)$ (protocol 2) based on the rules defined in appendix A. The item is signed by the voter authorizer signing key $x_{\mathcal{A}}$. If the voter athorization mode is identity-based, the item

contains the list of identity providers $\mathcal{I}_j$, with $j \in \{1, ..., n_\text{i}\}$, each defined by their public key $Y_{\mathcal{I}_j}$.

4.5 Voter credentials distribution process

In case the voter authentication mode is **credential-based**, each credentials authority $\mathcal{C}_j \in \boldsymbol{\mathcal{C}}$, receives a list of voters consisting of contact details for each voter $\{a_1, ..., a_{n_\text{v}}\}$ in the form of e-mail addresses, postal addresses or phone numbers, depending on the credentials authority's communication channel. The authority generates random credentials $c_{i,j} \in_\text{R} \mathbb{B}^\ell$ for each voter, with $i \in \{1, ..., n_\text{v}\}$. The authority distributes the credential $c_{i,j}$ to a specific voter $\mathcal{V}_i$ (using that voter's contact details a_i) and appends the corresponding public authentication key $Y_{\text{auth};i,j}$ in the list of voters next to $\mathcal{V}_i$, where $(x_{\text{auth};i,j}, Y_{\text{auth};i,j}) \leftarrow \mathsf{Pass2Key}(c_{i,j})$ (algorithm 26).

Credentials can be generated as a random string of alphanumeric characters, bound by the level of entropy ℓ. It is recommended that credentials are based on at least 80 bits of entropy ($\ell \geq 80$). That corresponds to a 14-character alphanumeric code that is sent to the voter as credentials.

All credentials authorities $\mathcal{C}_j \in \boldsymbol{\mathcal{C}}$ return the lists with voters' contact details and public authentication keys $(a_i, Y_{\text{auth};i,j})$ to the voter authorizer. The voter authorizer then combines all keys received from all credentials authorities for each voter to form the voter's public authentication key $Y_{\text{auth};i} = \sum_{j=1}^{n_\text{c}} Y_{\text{auth};i,j}$.

For authenticating to the voter authorizer, the voter $\mathcal{V}_i \in \boldsymbol{\mathcal{V}}$ must input all credentials $\{c_{i,1}, ..., c_{i,n_\text{c}}\}$ received from all credentials authorities in the voting application. The application will thereafter derive keys from each credential $(x_{\text{auth};i,j}, Y_{\text{auth};i,j}) \leftarrow \mathsf{Pass2Key}(c_{i,j})$ (algorithm 26) and aggregate all of them to form the voter's private authentication key $x_{\text{auth};i} = \sum_{j=1}^{n_\text{c}} x_{\text{auth};i,j} \pmod q$. The private authentication key is used to compute a proof of credentials PK_auth, as described in section 4.7.1, which is used to authenticate the voter.

4.6 Threshold ceremony

An election official defines the lists of trustees $\boldsymbol{\mathcal{T}} = \{\mathcal{T}_1, ..., \mathcal{T}_{n_\text{t}}\}$. Then, the election official coordinates the threshold ceremony during which all trustees $\mathcal{T}_i \in \boldsymbol{\mathcal{T}}$ participate in the protocol from figure 12 described in appendix B.5. The protocol will generate the election encryption key Y_enc and each trustee's share of the decryption key sx_i. The election official sets the threshold value t, such that any t out of the n_t trustees can perform the decryption of ballots.

At the end of the ceremony, the election official interacts with the election administrator service which writes the threshold configuration item on the bulletin board by following $\texttt{WriteOnBoard}(\mathcal{E}, m_i, c_i, p_i)$ (protocol 2) based on the rules defined in appendix A. The content of the item c_i consists of:

- election encryption key Y_{enc} and the threshold setup t-out-of-n_{t},
- public keys of each trustee $Y_{\mathcal{T}_i}$, with $i \in \{1, ..., n_{\text{t}}\}$,
- public polynomial coefficients of each trustee $P_{\mathcal{T}_i,j}$, with $j \in \{1, ..., t-1\}$.

Finally, trustees validate that the threshold configuration item published on the bulletin board corresponds with the data generated by them during the threshold ceremony.

4.7 Voting

During the election phase, any voter $\mathcal{V}_i \in \boldsymbol{\mathcal{V}}$ can cast a valid digital ballot by performing the following steps:

- obtain the ancestry of configuration items α_{cnf} for the digital ballot box,
- authenticate and become authorized to cast a digital ballot on the bulletin board as described in section 4.7.1 or section 4.7.2 ,
- select and encode vote choices as described in section 4.7.3,
- encrypt the ballot following the process from section 4.7.4,
- cast the ballot and obtain a vote confirmation receipt as in section 4.7.5.

4.7.1 Voter authorization procedure - identity based

When the authorization procedure is identity based, a voter $\mathcal{V}_i$ must follow the protocol from figure 3 to get authorized to submit a digital ballot. Specifically, the voter must authenticate and receive identity tokens $\sigma_{\text{id},j}$ from all identity providers $\mathcal{I}_j \in \boldsymbol{\mathcal{I}}$ which the voter authorizer has configured in the pre-election phase. The voting application then generates a key pair $(x_i, Y_i) \leftarrow \mathsf{KeyGen}()$ (algorithm 16) and forwards all identity tokens $\{\sigma_{\text{id},1}, ..., \sigma_{\text{id},n_{\text{i}}}\}$ and the public key Y_i to the voter authorizer service $\mathcal{A}$ proving the identity of the voter $\mathcal{V}_i$.

If the voter authorizer service can validate all identity tokens and the voter is eligible, i.e., $\mathcal{V}_i \in \boldsymbol{\mathcal{V}}$, it will authorize the use of the public key Y_i for the voter $\mathcal{V}_i$. This is done by the voter authorizer $\mathcal{A}$ interacting with the digital ballot box $\mathcal{D}$ in the protocol 2 $\texttt{WriteOnBoard}(\mathcal{A}, m_{\text{vs}}, c_{\text{vs}}, p_{\text{vs}})$ to write a voter session item b_{vs} on the bulletin board as the next item. This occurs according to the rules specified in appendix A, where m_{vs} = "voter session", the parent p_{vs} is the address of the latest configuration item, and the content c_{vs} consists of the voter identifier, the public key Y_i, and the authentication fingerprint computed by hashing all identity tokens received from the voter.

The voter authorizer returns the voter session item b_{vs} to the voter as received from the digital ballot box. The voting application checks the item according

to the validations of protocol 2. Additionally, it verifies that the item is consistent according to the configuration ancestry α_{cnf}, i.e., $\mathsf{AncestryVer}(\{b_{\text{vs}}\}, h_{\text{cnf}})$ (algorithm 1), where h_{cnf} is the address of the last item in α_{cnf}. From this point on, the voter can interact directly with the digital ballot box as the identity $\mathcal{V}_i$.

Note that the voter has extracted from the configuration items α_{cnf} all the necessary values, such as the public keys of the voter authorizer $Y_{\mathcal{A}}$, of the digital ballot box $Y_{\mathcal{D}}$, and of each identity provider $Y_{\mathcal{I}_i}$.

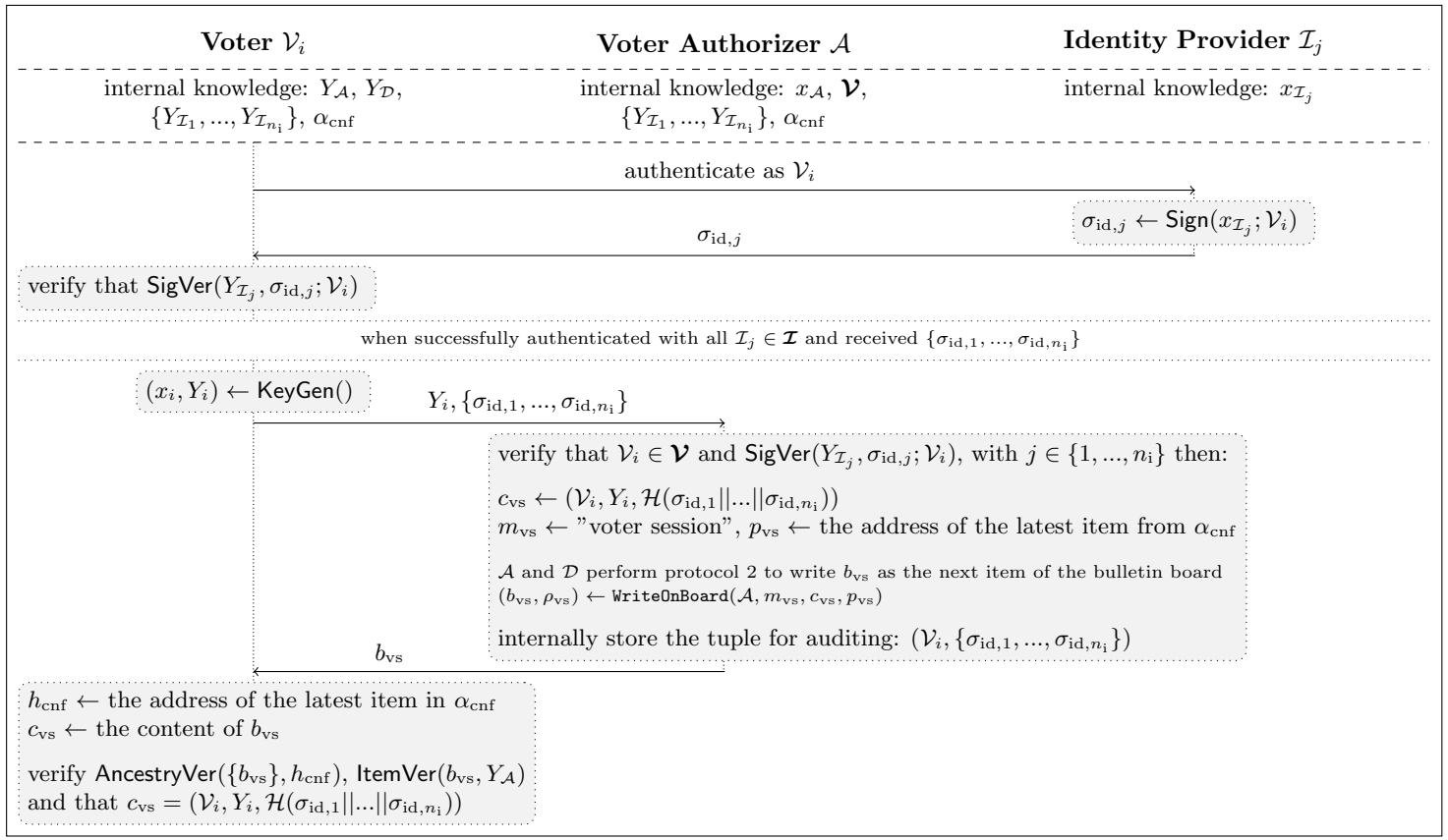


Figure 3: Identity-based voter authentication protocol

The voter authorizer service stores a link between the voter identity $\mathcal{V}_i$ and all related identity tokens for the administrative auditing process in the post-election phase, as described in section 5.2. This link is stored privately by the voter authorizer service since the identity tokens likely contain personal information that must not be disclosed on the public bulletin board.

4.7.2 Voter authorization procedure - credential based

When the authorization procedure is credential based, a voter $\mathcal{V}_i$ has to follow the protocol from figure 4 to get authorized to cast a digital ballot on the bulletin board. Specifically, the voter must prove possession of credentials associated with the voter's authentication public key $Y_{\text{auth};i}$.

Voter inputs the credentials received from each credentials authority $\{c_1, ..., c_{n_c}\}$ into the voting application. All credentials get converted into the voter's authentication key pair $(x_{\text{auth};i}, Y_{\text{auth};i})$, where the private key $x_{\text{auth};i}$ is computed by adding together all keys derived from each credential c_j (by using algorithm 26 $\mathsf{Pass2Key}(c_j)$), with $j \in \{1, ..., n_c\}$. The public key is computed by $Y_{\text{auth};i} \leftarrow [x_{\text{auth};i}]G$. Based on the private key, the voting application computes $PK_{\text{auth}} \leftarrow \mathsf{DLProve}(x_{\text{auth};i}, \{G\})$ (algorithm 14) as the proof of credentials.

Then, the voting application generates a new key pair (x_i, Y_i) to be used as the signing/signature verification keys in the upcoming voter session. The voting application sends the proof PK_{auth} and the public key Y_i to the voter authorizer proving possession of credentials of voter $\mathcal{V}_i$. The voter authorizer checks that the proof is valid and whether it was generated by an eligible voter from $\boldsymbol{\mathcal{V}}$.

If the authentication succeeds, the voter authorizer service will authorize the use of public key Y_i for the voter $\mathcal{V}_i$ by interacting with the digital ballot box $\mathcal{D}$ in $\texttt{WriteOnBoard}(\mathcal{A}, m_{\text{vs}}, c_{\text{vs}}, p_{\text{vs}})$ (protocol 2) to write a voter session item b_{vs} as the next item on the bulletin board, according to the rules specified in appendix A, where m_{vs} = "voter session", the parent p_{vs} is the address of the latest configuration item and the content c_{vs} consists of the voter identifier, the public key Y_i, and a digest of the proof PK_{auth}.

The voter authorizer returns to the voter with the voter session item b_{vs} as received from the digital ballot box. The voting application validates the item according to protocol 2. Additionally, it checks that the item is consistent according to the configuration ancestry α_{cnf}, i.e., $\mathsf{AncestryVer}(\{b_{\text{vs}}\}, h_{\text{cnf}})$ (algorithm 1), where h_{cnf} is the address of the last item in α_{cnf}. From this point on, the voter can interact directly with the digital ballot box as the identity $\mathcal{V}_i$.

The voter authorizer service stores a link between the voter identity $\mathcal{V}_i$ and the proof of credentials PK_{auth} for the administration auditing process in the post-election phase as described in section 5.2.

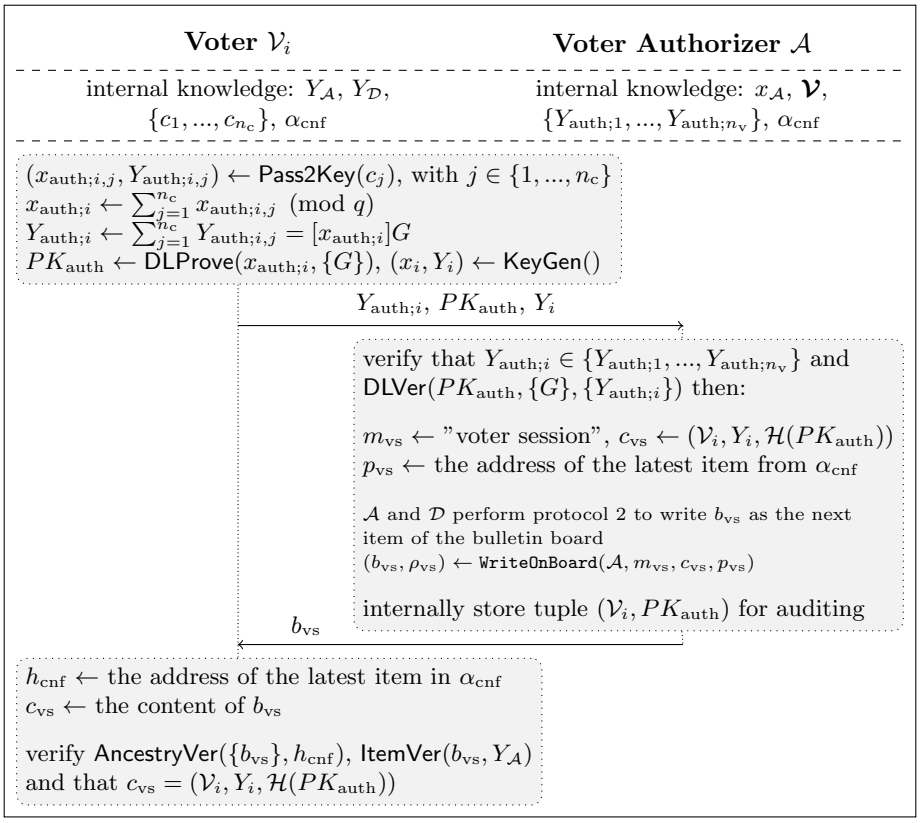


Figure 4: Credential-based voter authentication protocol

4.7.3 Mapping vote options on the Elliptic Curve

An expressed vote (i.e., a vote in plain text) must be able to be converted deterministically into elliptic curve points to be used in our cryptographic protocols. Additionally, a series of points from the elliptic curve must be able to be converted back into a plain-text vote if said points have been constructed from a plain-text vote. Depending on the election type (referendum, simple election, multiple choice election, STV election), the plain text vote can be constructed in different ways, such as a simple string, an array of integers, or even a complex data structure. Regardless of the election type and marking rules, we represent a plain-text vote as a byte array $\vec{b} \in \mathbb{B}^*$.

Next, $\vec{b}$ is converted into elliptic curve points $\vec{V} \leftarrow \mathsf{EncodeVote}(\vec{b})$ (algorithm 4), which can be used in the encryption mechanism described in section 4.7.4. Thus, the set of points $\vec{V}$ represents the voter's choices in cryptographic form.

Recovering a byte array from $\vec{V}$ is done by $\vec{b} \leftarrow \mathsf{DecodeVote}(\vec{V})$ (algorithm 5).

Then, $\vec{b}$ can be interpreted as a plain-text vote, depending on encoding rules.

Algorithm 4: $\mathsf{EncodeVote}(\vec{b})$

Data: The plain-text vote $\vec{b} = \{b_1, ..., b_n\} \in \mathbb{B}^n$
$m \leftarrow \mathsf{ByteLengthOf}(p) - 1$ `// algorithm 11`
$\ell \leftarrow \lceil n/m \rceil$
for $i \leftarrow 0$ **to** $\ell - 1$ **by** 1 **do**
  $V_{i+1} \leftarrow \mathsf{Bytes2Point}(\{b_{i*m+1}, ..., b_{i*m+m}\})$ `// algorithm 12`
end
$\vec{V} \leftarrow \{V_1, ..., V_\ell\}$
return $\vec{V}$ `//` $\vec{V} \in \mathbb{P}^*$

Algorithm 5: $\mathsf{DecodeVote}(\vec{V})$

Data: The list of points $\vec{V} = \{V_1, ..., V_\ell\} \in \mathbb{P}^\ell$
$\vec{b} \leftarrow \{\}$
for $i \leftarrow 1$ **to** ℓ **by** 1 **do**
  $\vec{b} \leftarrow \vec{b} \cup \mathsf{Point2Bytes}(V_i)$ `// algorithm 13`
end
return $\vec{b}$ `//` $\vec{b} \in \mathbb{B}^*$

4.7.4 Vote cryptogram generation process

During the vote cryptogram generation process, the voting application collaborates with the digital ballot box $\mathcal{D}$ for generating cryptograms $\vec{e}$ that represent the encryption of the vote $\vec{V}$. This process results in neither the voter $\mathcal{V}_i$ nor the digital ballot box $\mathcal{D}$ having the whole randomizer value r used in the generation process of each cryptogram e (as explained in appendix B.4.1 a cryptogram $e = \mathsf{Enc}(Y_{\text{enc}}, V; r)$). That is achieved by the voter and the digital ballot box building up the randomizer, while neither of them knowing its entire value. It is important that neither the voter nor the digital ballot box know the value of r so they cannot produce cryptographic evidence of the way they voted. The entire process consists of each party generating its own encryption randomizer, then committing to it (figure 5), and finally, combining them to form the encrypted ballot and submitting it (figure 6).

The generation process begins with the voting application generating its encryption randomizers $\vec{r}_{\text{v}} = \{r_{\text{v};1}, ..., r_{\text{v};\ell}\} \in_{\text{R}} \mathbb{Z}_q^\ell$ and computing a commitment to them $c_{\text{v}} \leftarrow \mathsf{Com}(\vec{r}_{\text{v}}, s_{\text{v}})$ (algorithm 31), where $s_{\text{v}} \in_{\text{R}} \mathbb{Z}_q$. The voting application subsequently interacts with the digital ballot box in the protocol 2 $\texttt{WriteOnBoard}(\mathcal{V}_i, m_{\text{vec}}, c_{\text{vec}}, p_{\text{vec}})$ to append the vote encryption commitment item b_{vec} on the board, where $m_{\text{vec}} =$ "voter encryption commitment", the content c_{vec} consists of the commitment c_{v}, and the parent p_{vec} is the address of

the voter session item, received in section 4.7.1. Note that before appending the new item, the board consists of $\{b_1, ..., b_{k-1}\}$, thus b_{vec} becoming the k^{th} item.

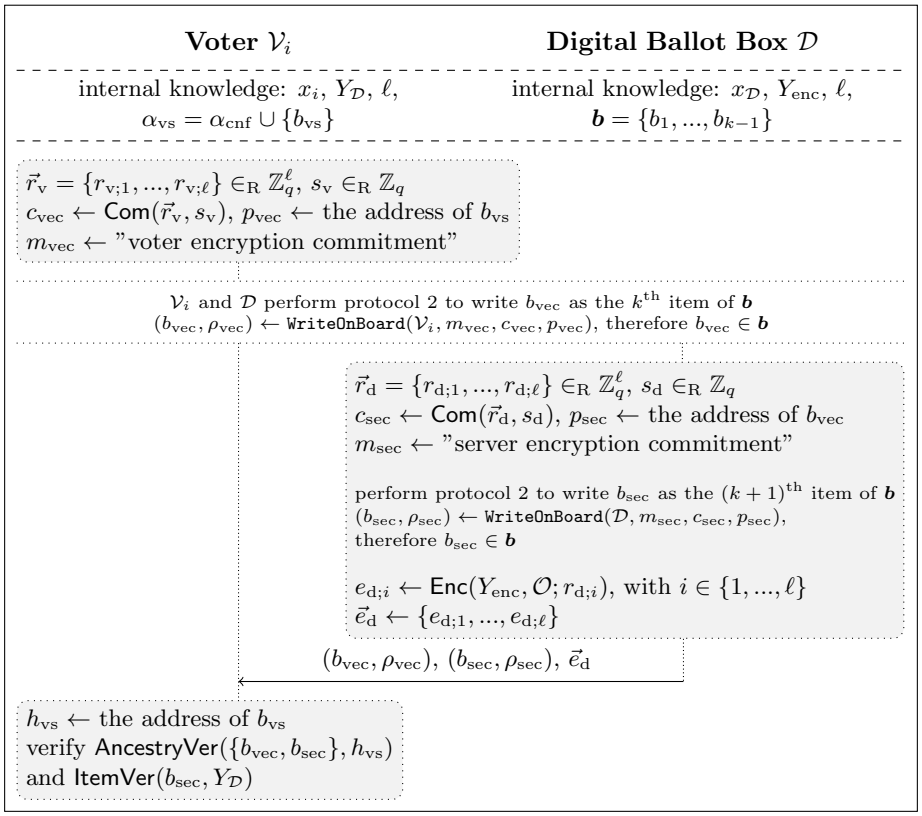


Figure 5: Encryption commitments submission protocol

After publishing the voter encryption commitment item on the bulletin board, the digital ballot box immediately generates its own set of encryption randomizers $\vec{r}_{\text{d}} = \{r_{\text{d};1}, ..., r_{\text{d};\ell}\} \in_{\text{R}} \mathbb{Z}_q^{\ell}$ and commitment $c_{\text{d}} \leftarrow \mathsf{Com}(\vec{r}_{\text{d}}, s_{\text{d}})$ (algorithm 31), where $s_{\text{d}} \in_{\text{R}} \mathbb{Z}_q$. It then self-writes a server encryption commitment item b_{sec} on the board by running protocol 2 $\texttt{WriteOnBoard}(\mathcal{D}, m_{\text{sec}}, c_{\text{sec}}, p_{\text{sec}})$, where $m_{\text{sec}} =$ "server encryption commitment", the content c_{sec} consists of its commitment c_{d}, and p_{sec} is the address of the voter encryption commitment item b_{vec}.

Next, the digital ballot box returns to the voting application both items b_{vec} and b_{sec} together with their respective receipts, according to the protocol 2 and the empty cryptograms $\vec{e}_{\text{d}} = \{e_{\text{d};1}, ..., e_{\text{d};\ell}\}$, with each $e_{\text{d};i}$ being the encryption of the neutral point $\mathcal{O}$ using the encryption randomizers $r_{\text{d};i}$. The voting application performs the validation of the board items b_{vec} and b_{sec} according to the protocol 2 and continues, if successful.

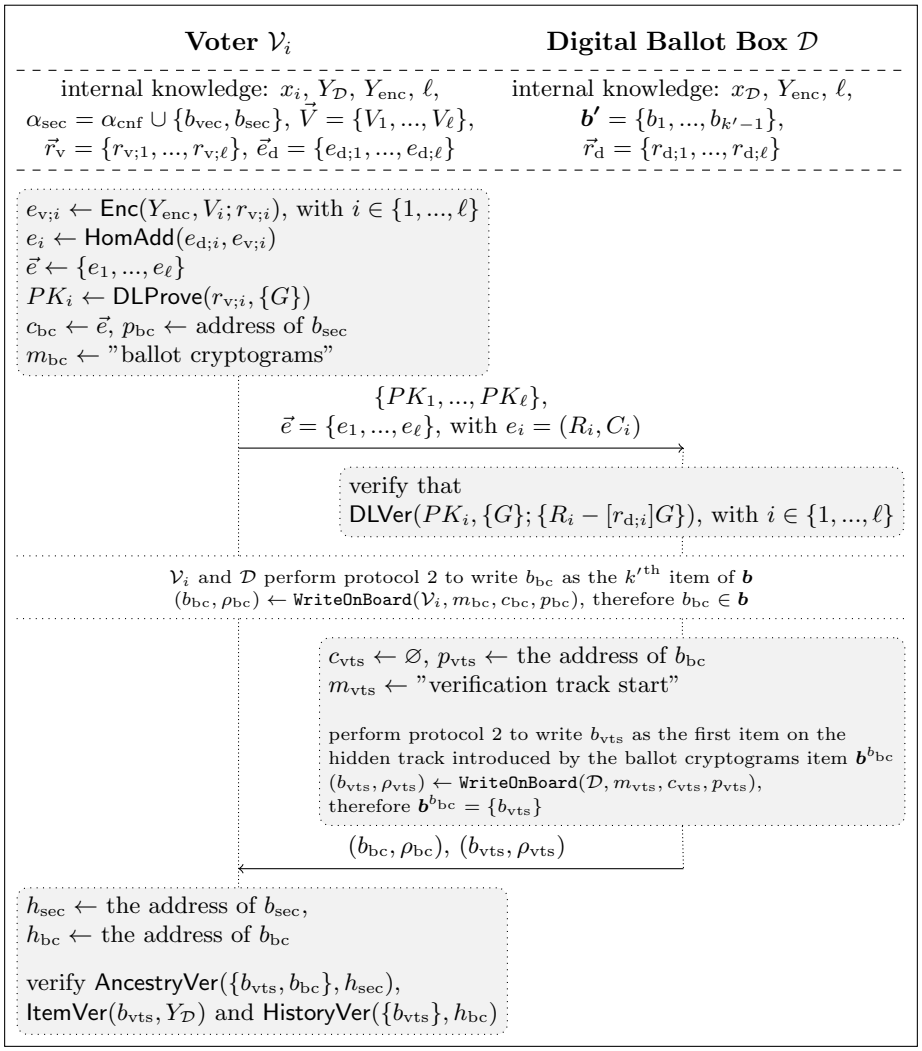


Figure 6: Encrypted ballot submission protocol

After both parties have published their encryption commitment items, as presented in figure 6, the voting application encrypts the voter's encoded vote $\vec{V}$ (as constructed in section 4.7.3) by computing $e_{\text{v};i} \leftarrow \mathsf{Enc}(Y_{\text{enc}}, V_i, r_{\text{v};i})$ (algorithm 17), with $i \in \{1, ..., \ell\}$. This is further combined with the empty cryptograms received from the digital ballot box to produce the voter's final ballot cryptograms $\vec{e} = \{e_1, ..., e_\ell\}$, where $e_i \leftarrow \mathsf{HomAdd}(e_{\text{v};i}, e_{\text{d};i})$ (algorithm 19). The voting application also computes as proof of correct encryption $PK_i \leftarrow \mathsf{DLProve}(r_{\text{v};i}, \{G\})$ (algorithm 14) to confirm that the empty cryptograms $\vec{e}_{\text{d}}$ have been used in the creation of the final ballot cryptograms $\vec{e}$.

The voting application interacts with the digital ballot box in the protocol $\texttt{WriteOnBoard}(\mathcal{V}_i, m_{\text{bc}}, c_{\text{bc}}, p_{\text{bc}})$ (protocol 2) to append the ballot cryptogram item b_{bc} on the board, where m_{bc} = "ballot cryptograms", the content c_{bc} consists of the cryptograms $\vec{e}$, and the parent p_{bc} is the address of the server encryption commitment item b_{sec}. Note that this time the bulletin board consists of items $\boldsymbol{b'} = \{b_1, ..., b_{k'-1}\}$, where $k' \geq k$ as more items could have been appended by other voters in between the protocols from figure 5 and figure 6, resulting in b_{bc} becoming the k'^{th} item.

Additionally, the voting application submits the proofs $\{PK_1, ..., PK_\ell\}$ to the digital ballot box, which performs protocol 2 if $\mathsf{DLVer}(PK_i, \{G\}, \{R_i - [r_{\text{d};i}]G\})$ (algorithm 15) succeeds, for each $i \in \{1, ..., \ell\}$, where the content of the item c_{bc} consists of $\vec{e} = \{e_1, ..., e_\ell\}$ and each $e_i = (R_i, C_i)$.

After publishing the ballot cryptograms item on the bulletin board, the digital ballot box immediately self-writes a verification track start item b_{vts} on the hidden track of the bulletin board $\boldsymbol{b}^{b_{\text{bc}}}$ by running $\texttt{WriteOnBoard}(\mathcal{D}, m_{\text{vts}}, c_{\text{vts}}, p_{\text{vts}})$ (protocol 2), where m_{vts} = "verification track start", the content c_{sec} is empty and the parent p_{vts} is the address of the ballot cryptogram item b_{bc}. Note that, at this point, the hidden track contains $\boldsymbol{b}^{b_{\text{bc}}} = \{b_{\text{vts}}\}$.

Next, the digital ballot box returns both items b_{bc} and b_{vts} to the voting application together with their respective receipts, according to the protocol 2. The voting application validates the two board items according to the protocol 2. In addition, it checks that the verification track start item is the only item on the hidden track by $\mathsf{HistoryVer}(\{b_{\text{vts}}\}, h_{\text{bc}})$ (algorithm 2), where h_{bc} is the address of the ballot cryptograms item.

Throughout the entire process, if an actor fails to validate any of the verification steps, the actor stops and aborts the protocol.

Note that each cryptogram e_i is actually equivalent to $\mathsf{Enc}(Y_{\text{enc}}, V_i; r_i)$, where $r_i = r_{\text{v};i} + r_{\text{d};i}$. Both the voter and the digital ballot box know part of the randomizer value, $r_{\text{v};i}$ and $r_{\text{d};i}$ respectively, but neither of them knows the combined value r_i, for any $i \in \{1, ..., \ell\}$.

4.7.5 Vote confirmation receipt

After encrypting a ballot, the voter $\mathcal{V}_i$ can choose whether to test or cast it. After deciding to cast the ballot, the voter receives a receipt from the digital ballot box that confirms that the ballot has been registered as cast on the public bulletin board.

The voter has to follow the protocol from figure 7 where the voting application interacts with the digital ballot box in $\texttt{WriteOnBoard}(\mathcal{V}_i, m_{\text{cr}}, c_{\text{cr}}, p_{\text{cr}})$ (protocol 2) to append the cast request item b_{cr} on the board, where m_{cr} = "cast request", the content c_{cr} is empty and the parent p_{cr} is the address of the ballot cryptograms item b_{bc}.

After publishing the cast request item on the bulletin board, the digital ballot box return to the voting app with the item b_{cr} and its receipt ρ_{cr}. The voting app checks the item according to validations of protocol 2, and if valid, the voting application presents the receipt ρ_{cr} together with σ_{cr} and h_{cr} to the voter.

The voter stores the tuple as the vote confirmation receipt (i.e., proof of the ballot being cast on the bulletin board). The voter can use it at any time to check that the vote is registered on the bulletin board as described in section 5.1.2.

Note that if a voter $\mathcal{V}_i$ has a valid vote confirmation receipt $(\rho_{\text{cr}}, \sigma_{\text{cr}}, h_{\text{cr}})$, which does not correspond with the current state of the bulletin board, i.e., h_{cr} is not an address on the bulletin board $\boldsymbol{b}$, that reveals that the integrity of the bulletin board has been broken and should be reported to the election officials.

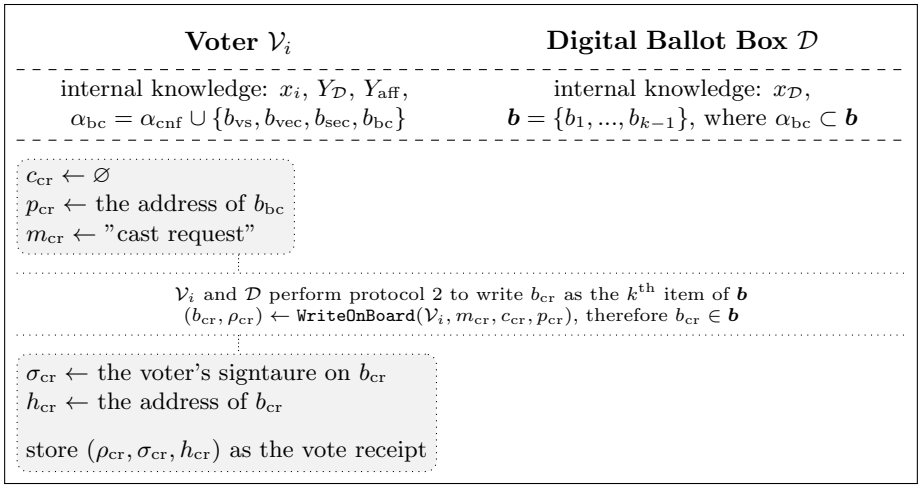


Figure 7: Ballot casting protocol

4.8 Ballot checking

After encrypting a ballot, the voter $\mathcal{V}_i$ can choose whether to test or cast it. To perform the testing process of an encrypted ballot, the voter needs to interact with the *external verifier* that will perform all the testing operations on behalf of the voter, according to the data published on the bulletin board. At the end of the testing process, the voter will be presented with the vote choices encoded in the encrypted ballot. The encrypted ballot being tested gets spoiled when doing the testing procedure. Therefore, the voter needs to redo the vote cryptogram generation process from section 4.7.4 to get a new encrypted ballot, which the voter has to choose again whether to test or to cast. This process can be repeated until the voter trusts the legitimacy of the next encrypted ballot generated by the voting application. The protocol is inspired by [3].

The first part of the protocol (figure 8) establishes a trusted connection between the voting application and the external verifier over the bulletin board. The voter inputs into the external verifier the address of the verification track start item b_{vts}, which queries the digital ballot box for the item at that address and its ancestry. The digital ballot box returns $\alpha_{\text{vts}} = \alpha_{\text{cnf}} \cup \{b_{\text{vs}}, b_{\text{vec}}, b_{\text{sec}}, b_{\text{bc}}, b_{\text{vts}}\}$, which consists of all the configuration items α_{cnf} (e.g., the genesis item, election configuration items, contest configuration items, etc.) plus all the voting items that are relevant to voter $\mathcal{V}_i$. Notice that all configuration and voting items are on the public bulletin board (i.e., $\alpha_{\text{cnf}}, b_{\text{vs}}, b_{\text{vec}}, b_{\text{sec}}, b_{\text{bc}} \in \boldsymbol{b}$), except the verification track start item b_{vts} which exists on the hidden track $\boldsymbol{b}^{b_{\text{bc}}}$ that has been spawned by the ballot cryptograms item b_{bc}.

The external verifier validates the list by running $\mathsf{AncestryVer}(\alpha_{\text{vts}}, \varnothing)$ (algorithm 1), therefore checking that α_{vts} has a consistent ancestry all the way through the genesis item, which has no parent. Thus, the parent of the entire ancestry is null or $\varnothing$. The external verifier also checks the integrity of every item by $\mathsf{ItemVer}(b_j, Y_{\mathcal{W}})$ (algorithm 3), where $b_j \in \alpha_{\text{vts}}$ and $Y_{\mathcal{W}}$ is the public key of the respective writer, according to the rules from appendix A. Note that the set of writers, as presented in section 2.2, consists of the voter $\mathcal{V}_i$, the digital ballot box $\mathcal{D}$, the election administrator $\mathcal{E}$ and the voter authorizer $\mathcal{A}$. The external verifier can extract the voter's public key Y_i from the voter session item b_{vs} and the other public keys $Y_{\mathcal{D}}$, $Y_{\mathcal{E}}$ and $Y_{\mathcal{A}}$ from the configuration items. If valid, the external verifier notifies the voter that the ballot was successfully found.

Then, the voter chooses to test the encryption of the ballot, so the voting application interacts with the digital ballot box in $\mathtt{WriteOnBoard}(\mathcal{V}_i, m_{\text{sr}}, c_{\text{sr}}, p_{\text{sr}})$ (protocol 2) to append the spoil request item b_{sr} on the board, where $m_{\text{sr}} =$ "spoil request", the content c_{sr} is empty and the parent p_{sr} is the address of the ballot cryptograms item b_{bc}.

After publishing the spoil request item b_{sr}, the digital ballot box sends the new item also to the external verifier, which verifies its integrity $\mathsf{ItemVer}(b_{\text{sr}}, Y_i)$ (algorithm 3) and that it is consistent with the ancestry $\mathsf{AncestryVer}(\{b_{\text{sr}}\}, h_{\text{bc}})$ (algorithm 1), where h_{bc} is the address of the ballot cryptograms item b_{bc}. If valid, the external verifier generates its key pair $(x_{\mathcal{X}}, Y_{\mathcal{X}}) \leftarrow \mathsf{KeyGen}()$ (algorithm 16) and interacts with the digital ballot box in $\mathtt{WriteOnBoard}(\mathcal{X}, m_{\text{v}}, c_{\text{v}}, p_{\text{v}})$ (protocol 2) to write a verifier item b_{v} on the hidden track, where $m_{\text{v}} =$ "verifier", the content c_{v} contains the external verifier's public key $Y_{\mathcal{X}}$ and the parent p_{v} is the address of the spoil request item b_{sr}. Note that the verifier item b_{v} is appended on the hidden track introduced by the ballot cryptograms item b_{bc}. Therefore, at the end of this step, the hidden track consists of $\boldsymbol{b}^{b_{\text{bc}}} = \{b_{\text{vts}}, b_{\text{v}}\}$. From this point on, the external verifier represents identity $\mathcal{X}$ on the hidden track $\boldsymbol{b}^{b_{\text{bc}}}$.

The protocol continues with figure 9 where the external verifier returns to the voter with the address of the verifier item h_{v}. The voter also receives the verifier item b_{v} from the digital ballot box. The voter checks the integrity of the item $\mathsf{ItemVer}(b_{\text{v}}, Y_{\mathcal{X}})$ (algorithm 3) and that it is consistent with the ancestry $\mathsf{AncestryVer}(\{b_{\text{v}}\}, h_{\text{vts}})$ (algorithm 1), where $Y_{\mathcal{X}}$ is extracted from the content

of the verifier item and h_{vts} is the address of the verification track start item. Then the voter checks that the address received from the external verifier is consistent with the verifier item received from the digital ballot box. If valid, the voter managed to establish a trusted connection with the external verifier over the bulletin board. Therefore the protocol can continue.

Next, both the voter and the digital ballot box collaborate to securely deliver their encryption randomizers $\vec{r}_{\text{v}}$ and $\vec{r}_{\text{d}}$ respectively to the external verifier, as generated in section 4.7.4. The external verifier will use them to decrypt the voter's ballot cryptograms and present the vote choices for assessment.

This is achieved by the voting application encrypting (using standard symmetric key encryption) the randomizers and the commitment opening $d_{\text{v}} \leftarrow \mathsf{SymEnc}(k_{\text{v}}, \vec{r}_{\text{v}}||s_{\text{v}})$ (algorithm 22), where k_{v} is a derived symmetric key based on Diffie-Hellman key exchange mechanism between the voter and the external verifier, i.e., $k_{\text{v}} \leftarrow \mathsf{DerSymKey}(x_i, Y_{\mathcal{X}})$ (algorithm 24). Then, the voter interacts with the digital ballot box to write the voter commitment opening item b_{vco} on the hidden track $\texttt{WriteOnBoard}(\mathcal{V}_i, m_{\text{vco}}, c_{\text{vco}}, p_{\text{vco}})$ (protocol 2), where $m_{\text{vco}} =$ "voter commitment opening", content c_{vco} consists of the encryption d_{v} and the parent p_{v} is the address of the verifier item b_{v}.

After publishing the voter commitment opening item, the digital ballot box immediately computes its own encryption of the randomizers and commitment opening d_{d} using the same strategy as the voter in the previous paragraph. Then, it self writes a server commitment opening item b_{sco} on the board by running $\texttt{WriteOnBoard}(\mathcal{D}, m_{\text{sco}}, c_{\text{sco}}, p_{\text{sco}})$ (protocol 2), where $m_{\text{sco}} =$ "server commitment opening", the content c_{sec} consists of the encryption d_{d} and the parent p_{sco} is the address of the voter commitment opening item b_{vco}.

Then (figure 10), the external verifier is notified about both commitment opening items, which verifies their integrity and that they are consistent with the previous ancestry. If valid, it decrypts (using standard symmetric key decryption) both commitment openings of the voter $(\vec{r}_{\text{v}}, s_{\text{v}}) \leftarrow \mathsf{SymDec}(k_{\text{v}}, d_{\text{v}})$ (algorithm 23) and of the digital ballot box $(\vec{r}_{\text{d}}, s_{\text{d}}) \leftarrow \mathsf{SymDec}(k_{\text{d}}, d_{\text{d}})$, where the encryptions d_{v} and d_{d} are extracted from the content of the voter and the server commitment opening items respectively. The symmetric keys k_{v} and k_{d} are computed based on the Diffie-Hellman key exchange mechanism between the external verifier and the voter or the digital ballot box, respectively.

Next, the external verifier checks whether the commitment openings are consistent with the commitments that were published in section 4.7.4, i.e., verification of the voter commitment $\mathsf{ComVer}(c_{\text{v}}, \vec{r}_{\text{v}}, s_{\text{v}})$ (algorithm 32) and of the server commitment $\mathsf{ComVer}(c_{\text{d}}, \vec{r}_{\text{d}}, s_{\text{d}})$, where commitments c_{v} and c_{d} are extracted from the voter and server encryption commitment items respectively. If commitments are valid, the external verifier proceeds to unpack the cryptograms $\vec{e}$, which are extracted from the ballot cryptograms items b_{bc}. If any validations fail, the external verifier informs the voter about the failure.

The external verifier unpacks vote $\vec{V}'$ by decrypting a variant of each cryptogram $e_i = (R_i, C_i)$, with $e_i \in \vec{e}$, where point R_i is substituted by the encryption key Y_{enc}, such that it can be decrypted by the randomizer $r_{\text{v};i} + r_{\text{d};i}$ instead of the decryption key. Note that the encryption key Y_{enc} can be extracted from the threshold configuration item, which is part of α_{cnf}. Formally, $\vec{V}' = \{V'_1, ..., V'_\ell\}$, with $V'_i \leftarrow \mathsf{Dec}(r_{\text{v};i} + r_{\text{d};i}, e'_i)$ (algorithm 18), where $e'_i \leftarrow (Y_{\text{enc}}, C_i)$.

Finally, the external verifier presents the vote $\vec{V}'$ to the voter, which can compare to the original vote choice $\vec{V}$, as computed in section 4.7.3. Note that $\vec{V}'$ can even be decoded into a human-readable presentation of the vote choices by decoding it to bytes $\mathsf{DecodeVote}(\vec{V}')$ (algorithm 5) and then into a plain-text vote according to the configuration from α_{cnf}. If the vote matches, then the voter is assured that the voting application behaved correctly (i.e., encrypted a genuine vote). Otherwise, the voter has evidence that the voting application has misbehaved during the process and should act accordingly.

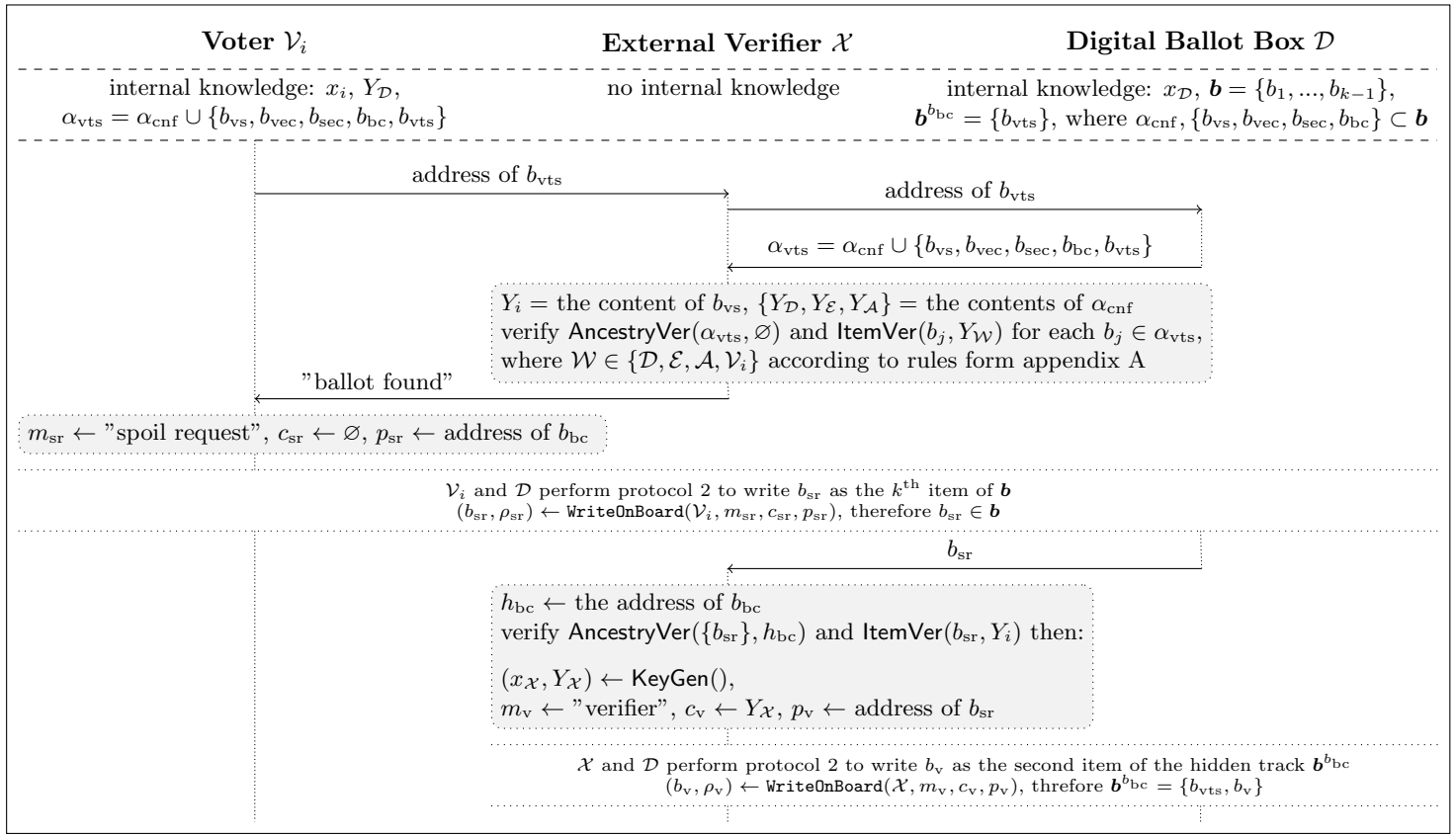


Figure 8: External verifier setup protocol

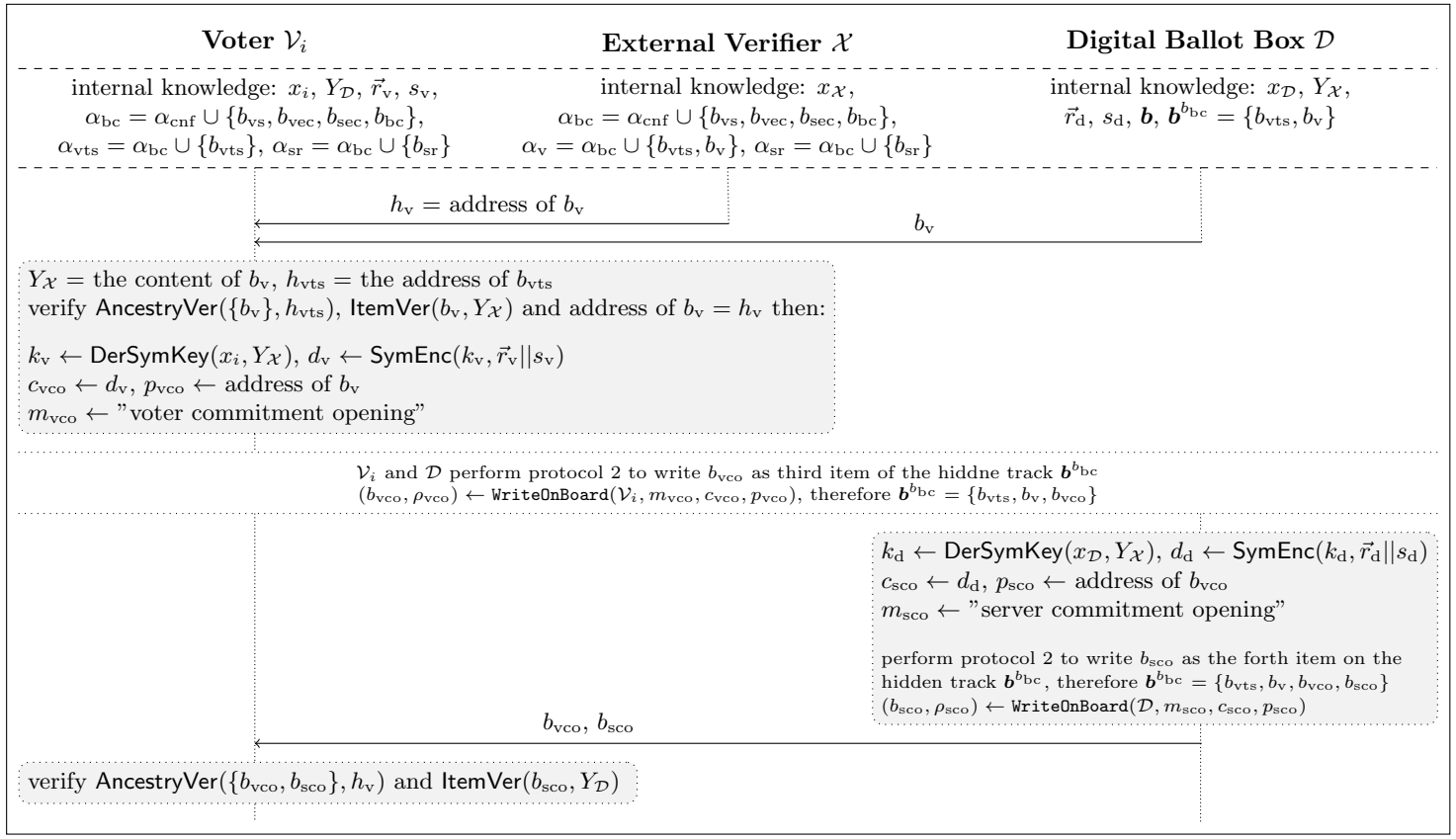


Figure 9: Commitment opening submission protocol

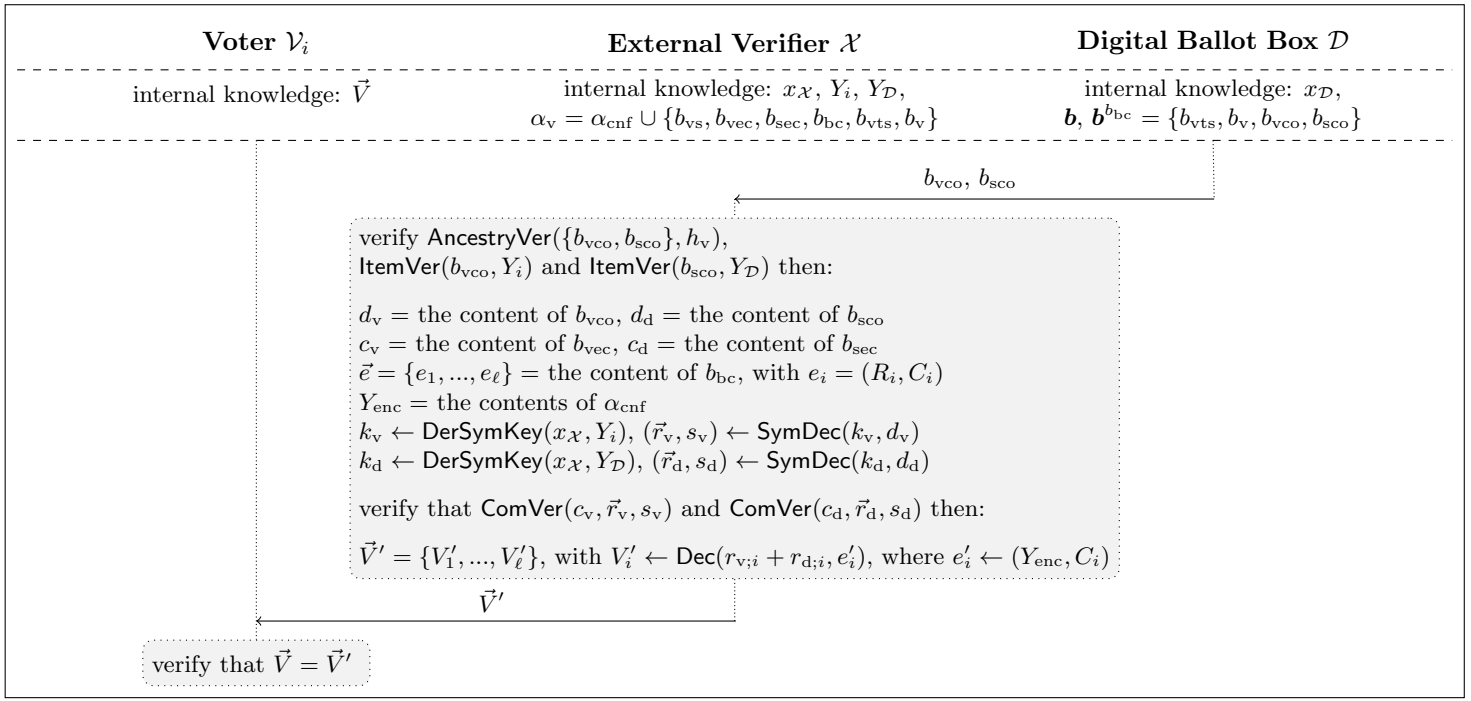


Figure 10: Unpacking the encrypted ballot protocol

4.9 Ballot extraction

After the voting phase has finished, the election proceeds to the last step, which will generate the election result. Now, the digital ballot box does not accept any new vote cryptograms. The bulletin board remains publicly available for auditing purposes.

The process of computing a result consists of the following:

- an election official uses the election administrator service to request a result to be computed by interacting with the digital ballot box in protocol 2 `WriteOnBoard`$(\mathcal{E}, m_{\text{ei}}, c_{\text{ei}}, p_{\text{ei}})$ to publish an extraction intent item b_{ei} on the bulletin board, according to the rules from appendix A,
- the digital ballot box identifies the ballots to be included in the tally, according to section 4.9.1,
- a subset of all trustees $\mathcal{T}_i$, with $i \in \tau$ and $\tau \subset \{1, ..., n_{\text{t}}\}$, collaborate in the mixing process to anonymize the encrypted ballots, as described in section 4.9.2,
- the same subset of trustees collaborate in the decryption process (section 4.9.3) of the anonymized votes,
- finally, an election official publishes the result confirmation item on the bulletin board as described in section 4.9.4.

This process of computing a result is executed separately for each voting round configured in the pre-election phase (described in section 4.3.2).

4.9.1 Extraction procedure

Triggered by the extraction intent item b_{ei} being published, the digital ballot box $\mathcal{D}$ bundles the list of encrypted ballots to be extracted. They are represented by the cryptograms that exist in the content of the ballot cryptograms items that conform to the following rules:

- it is the latest ballot cryptograms items submitted by a particular voter
- it is followed by a cast request item

All other ballots are considered overwritten or rejected and, therefore, discarded.

We denote the extracted ballots as the matrix of cryptograms $\vec{\vec{e}}_0 = \{\vec{e}_1, ..., \vec{e}_{n_{\text{e}}}\}$. $\vec{\vec{e}}_0$ is called the *initial mixed board* and it is used as input to the mixing phase.

Next, the digital ballot box $\mathcal{D}$ self-publishes an extraction data item b_{ed} on the bulletin board by running `WriteOnBoard`$(\mathcal{D}, m_{\text{ed}}, c_{\text{ed}}, p_{\text{ed}})$ (protocol 2), where m_{ed} = "extraction data", the content c_{ed} consists of the initial mixed board $\vec{\vec{e}}_0$ and the parent p_{ed} is the address of the extraction intent item b_{ei}.

The extraction procedure is publicly auditable as both the list of vote cryptograms and the *initial mixed board* are publicly available.

4.9.2 Mixing Phase

During the mixing phase, the board of cryptograms will change its appearance several times, being shuffled in an indistinguishable way. Each trustee, $\mathcal{T}_i$ with $i \in \tau$, applies its mixing algorithm in sequential order (the output from $\mathcal{T}_{i-1}$ is the input to $\mathcal{T}_i$). The first trustee applies its algorithm on *the initial mixed board*, and the output of the last trustee is used as *the final mixed board*. The election administrator facilitates the mixing phase and decides the order of trustees.

Formally, trustee $\mathcal{T}_i$ computes the mixed board of cryptograms by applying $\vec{\vec{e}}_i \leftarrow \mathsf{Shuffle}(Y_{\text{enc}}, \vec{\vec{e}}_{i-1}, \vec{\vec{r}}_i, \psi_i)$ (algorithm 33), where Y_{enc} is the encryption key, $\vec{\vec{r}}_i \in_{\text{R}} \mathbb{Z}_q^{n_\text{e} \times \ell}$ and ψ_i is a permutation of n_e elements. Next, as described in appendix B.10, trustee $\mathcal{T}_i$ computes a proof of correct mixing $(PM_i, AS_i) \leftarrow \mathsf{MixProve}(\psi_i, Y_{\text{enc}}, \vec{\vec{r}}_i, \vec{\vec{e}}_{i-1}, \vec{\vec{e}}_i)$ (algorithm 36). Then, trustee $\mathcal{T}_i$ submits to the election administrator the mixed board $\vec{\vec{e}}_i$ and the mixing proof (PM_i, AS_i).

For a mixing step to be accepted, the validity of the mixing proof has to be checked by running $\mathsf{MixVer}(PM_i, AS_i, Y_{\text{enc}}, \vec{\vec{e}}_{i-1}, \vec{\vec{e}}_i)$ (algorithm 37). If the proof fails, either that trustee recomputes the mixing step or is removed, and the process continues without that trustee.

Obviously, each trustee $\mathcal{T}_i$ knows the shuffling coefficients ($\vec{\vec{r}}_i$ and ψ_i) of its own mixing algorithm, and it can link the votes on the previous mixed board $\vec{\vec{e}}_{i-1}$ with the ones on the mixing board at i^{th} step $\vec{\vec{e}}_i$. However, $\mathcal{T}_i$ does not know the shuffling coefficients of the other trustees, so it cannot create a complete link between the votes on the *final mixed board* and the ones on the *initial mixed board*, unless all trustees are corrupt and collude against the election.

Assuming at least one honest trustee will not reveal its shuffling coefficients, the *final mixed board* of cryptograms represents the anonymized version of the *initial mixed board* of cryptograms. The *final mixed board* of cryptograms is used in the decryption phase to compute the election results.

4.9.3 Decryption Phase

Because the link between a vote cryptogram and its voter has been broken during the mixing phase, it is safe to decrypt all the cryptograms from the *final mixed board* as it does not violate the secrecy of the election. Furthermore, decrypting this list of cryptograms would lead to accurate and correct results as it contains the exact same votes as the initial mixed board, a fact proven by the mixing proofs. In this section, the *final mixed board* is referred to as $\vec{\vec{e}}$.

During the decryption phase, trustees must collaborate again to perform the threshold decryption protocol as presented in [4]. Each trustee, $\mathcal{T}_i$ with $i \in \tau$, gets the *final mixed board* of cryptograms $\vec{\vec{e}} = \{e_{1,1}, ..., e_{n_e,\ell}\}$ then computes partial decryptions of each cryptogram together with a proof of correct decryption by applying $(\vec{\vec{S}}_i, PK_i) \leftarrow \mathsf{PartiallyDecryptAndProve}(\vec{\vec{e}}, sx_i)$ (algorithm 6). Recall that trustee $\mathcal{T}_i$ owns its share of the decryption key sx_i as it has been computed during the threshold ceremony (section 4.6).

Then, trustee $\mathcal{T}_i$ submits the partial decryption $\vec{\vec{S}}_i$ and the proof PK_i to the ceremony coordinator, which accepts the partial decryption if the proof validates according to $\mathsf{PartialDecryptionVer}(\vec{\vec{e}}, \vec{\vec{S}}_i, PK_i, sY_i)$ (algorithm 7). Note that sY_i is the public share of the trustee $\mathcal{T}_i$, which is computable based on the public polynomial coefficients generated during the threshold ceremony (section 4.6):

$$sY_i \leftarrow \sum_{j=1}^{n}(Y_j + \sum_{k=1}^{t-1}[i^k]P_{j,k}).$$

Upon receiving partial decryptions $\vec{\vec{S}}_i$ from all trustees $\mathcal{T}_i$ with $i \in \tau$, the ceremony orchestrator aggregates all partial decryptions for each cryptogram in $\vec{\vec{e}}$ to finalize the decryption and extract the votes $\vec{\vec{V}} = \{V_{1,1}, ..., V_{n_e,\ell}\} \leftarrow \mathsf{FinalizeDecryption}(\vec{\vec{e}}, \{\vec{\vec{S}}_i | i \in \tau\})$ (algorithm 8). The aggregation is done by calculating the *Lagrange Interpolation Polynomial* where each term is a partial decryption S_i received from a trustee $\mathcal{T}_i$ that needs to be multiplied by the *Lagrange Interpolation Polynomial coefficient* which is $\lambda(i) = \prod_{j \in \tau, j \neq i} \frac{-j}{i-j} \pmod q$. Note that the calculation is possible only when $t \leq |\tau| \leq n_t$, where t is the threshold value set during the threshold ceremony (section 4.6) and n_t is the total number of trustees.

After being decrypted, $\vec{\vec{V}}$ represents the *raw result* of the election, i.e., the full list of votes as elliptic curve points.

Algorithm 6: PartiallyDecryptAndProve($\vec{\vec{e}}, sx$)

Data: The matrix of cryptograms $\vec{\vec{e}} = \{e_{1,1}, ..., e_{n,\ell}\} \in \mathbb{E}^{n \times \ell}$, with $e_{i,j} = (R_{i,j}, C_{i,j})$
The share of decryption key $sx \in \mathbb{Z}_q$

for $i \leftarrow 1$ **to** n **by** 1 **do**
  for $j \leftarrow 1$ **to** ℓ **by** 1 **do**
    $S_{i,j} \leftarrow [sx]R_{i,j}$
  end
end
$\vec{\vec{S}} \leftarrow \{S_{1,1}, ..., S_{n,\ell}\}$
$\vec{R} \leftarrow \{G, R_{1,1}, ..., R_{n,\ell}\}$
$PK \leftarrow$ DLProve$(sx, \vec{R})$ `// algorithm 14`
return $(\vec{\vec{S}}, PK)$ // $\vec{\vec{S}} \in \mathbb{P}^{n \times \ell}$, $PK \in \mathbb{P} \times \mathbb{Z}_q \times \mathbb{Z}_q$

Algorithm 7: PartialDecryptionVer($\vec{\vec{e}}, \vec{\vec{S}}, PK, sY$)

Data: The matrix of cryptograms $\vec{\vec{e}} = \{e_{1,1}, ..., e_{n,\ell}\} \in \mathbb{E}^{n \times \ell}$, with $e_{i,j} = (R_{i,j}, C_{i,j})$
The partial decryptions $\vec{\vec{S}} = \{S_{1,1}, ..., S_{n,\ell}\} \in \mathbb{P}^{n \times \ell}$
The proof of correct decryption $PK \in \mathbb{P} \times \mathbb{Z}_q \times \mathbb{Z}_q$
The public share of decryption key $sY \in \mathbb{P}$

$\vec{R} \leftarrow \{G, R_{1,1}, ..., R_{n,\ell}\}$
$\vec{S} \leftarrow \{sY, S_{1,1}, ..., S_{n,\ell}\}$
$b \leftarrow$ DLVer$(PK, \vec{R}, \vec{S})$ `// algorithm 15`
return b // $b \in \mathbb{B}$

Algorithm 8: FinalizeDecryption($\vec{\vec{e}}, \vec{\vec{\vec{S}}}$)

Data: The matrix of cryptograms $\vec{\vec{e}} = \{e_{1,1}, ..., e_{n,\ell}\} \in \mathbb{E}^{n \times \ell}$, with $e_{i,j} = (R_{i,j}, C_{i,j})$
The partial decryptions $\vec{\vec{\vec{S}}} = \{\vec{\vec{S}}_k | k \in \tau\}$, with each $\vec{\vec{S}}_k = \{S_{k,1,1}, ..., S_{k,n,\ell}\} \in \mathbb{P}^{n \times \ell}$

for $i \leftarrow 1$ **to** n **by** 1 **do**
  for $j \leftarrow 1$ **to** ℓ **by** 1 **do**
    $V_{i,j} \leftarrow C_{i,j} - \sum_{k \in \tau} [\lambda(k)] S_{k,i,j}$
  end
end
$\vec{\vec{V}} \leftarrow \{V_{1,1}, ..., V_{n,\ell}\}$
return $\vec{\vec{V}}$ // $\vec{\vec{V}} \in \mathbb{P}^{n \times \ell}$

4.9.4 Result Interpretation

After ballots have been decrypted, the *raw result* can be interpreted and presented in a more readable way. The result interpretation depends on the election type (simple election, multiple choice, STV, etc.). For simplicity, we will consider the simple election case, where voters had to choose one option from a predefined set of candidates, i.e., a vote is a plain text representing a candidate's name.

First, all votes $\vec{V}_i \in \vec{\vec{V}}$ have to be decoded into bytes $\vec{b}_i \leftarrow \mathsf{DecodeVote}(\vec{V}_i)$ (algorithm 5), then interpreted as text and finally mapped to one of the candidate names. If any of these steps fail, the vote $\vec{V}_i$ is considered invalid. Tallying the votes that each candidate received is considered trivial and out of scope for this document.

Finally, all data that have been computed during the mixing and decryption phases are collected by the election administrator $\mathcal{E}$ and published on the bulletin board as the extraction confirmation item b_{ec} by performing protocol 2 $\texttt{WriteOnBoard}(\mathcal{E}, m_{\text{ec}}, c_{\text{ec}}, p_{\text{ec}})$, where $m_{\text{ec}} =$ "extraction confirmation", the parent p_{ec} is the address of the extraction data item b_{ed} and the content c_{ec} includes:

- a set of the following data from each trustee $\mathcal{T}_i$ that participated in the result ceremony, with $i \in \tau$:
  - the mixed boards of cryptograms $\vec{\vec{e}}_i$
  - the mixing proofs (PM_i, AS_i)
  - the partial decryptions $\vec{\vec{S}}_i$
  - the proofs of correct decryption PK_i
- the list of decrypted votes $\vec{\vec{V}}$
- the summarized (tallied) election result

5 Auditing

This section describes the entire auditing process of an election. It presents all the verification mechanisms, who conducts them, and what cryptographic algorithms they involve. These verification mechanisms can be split into three categories:

- voter-specific verification mechanisms that can be performed individually by voters and target their specific vote (section 5.1),
- internal auditing processes performed by election officials that target the behavior of the election system (section 5.2),
- publicly available audit processes that verify all data on the public bulletin board (section 5.3).

5.1 Individual voter verifications

During the voting process, voters can verify two aspects of their vote: that it is cast as intended and registered as cast. These verification steps help voters gain confidence that the election system behaves correctly, at least at processing their vote.

5.1.1 Vote is cast as intended

At the end of the vote cryptogram generation process (section 4.7.4), the voter is presented with a set of cryptograms $\vec{e} = \{e_1, ..., e_\ell\}$, with each $e_i = (R_i, C_i)$, where ℓ is the number of cryptograms a ballot is made out of. The set $\vec{e}$ is the encryption of vote $\vec{V}$ with the encryption key Y_{enc} and randomizers $\vec{r} = \{r_1, ..., r_\ell\}$, where each $r_i = r_{\text{v};i} + r_{\text{d};i}$. The set of randomizers $\vec{r}_{\text{v}} = \{r_{\text{v};1}, ..., r_{\text{v};\ell}\}$ is known by the voting application and $\vec{r}_{\text{d}} = \{r_{\text{d};1}, ..., r_{\text{d};\ell}\}$ is known only by the digital ballot box. Hence, it is the voting application and the digital ballot box that collectively perform the encryption of the voter's vote.

Because the vote is encrypted, the voter cannot tell whether the cryptograms $\vec{e}$ actually represent an encryption of vote $\vec{V}$ or not. Therefore, to get convinced that the voting application and the digital ballot box behaved correctly during the vote cryptogram generation process, the voter can perform the ballot checking process, as presented in section 4.8 to verify the activity of the voting application and digital ballot box.

If the voter chooses to perform the ballot checking process, a second device is used to perform all the cryptographic validations on behalf of the voter. Both randomizer sets $\vec{r}_{\text{v}}$ and $\vec{r}_{\text{d}}$ are sent securely from the voting application and the digital ballot box, respectively, to the external verifier application that runs on the secondary device. The verification application uses them to unpack the

encrypted ballot and present the vote choices to the voter. The fully detailed process is shown in section 4.8.

If the vote corresponds to the voter's intended choices, then the voter gains confidence that the voting application behaved correctly.

If the vote does not correspond to the voter's intention, the auditing process provides evidence that the encrypted ballot has not been cast as intended. Note that in this case, there is no distinction between the election system maliciously changing the voter's vote behind the scenes or the voter accidentally mischoosing the vote options.

After the voter has successfully performed the ballot checking process, the ballot gets invalidated because each of the randomizer values $r_{\mathrm{v};i} + r_{\mathrm{d};i}$ has been exposed, for each $i \in \{1, ..., \ell\}$. Now, the voter has to regenerate vote cryptograms (as presented in section 4.7.4) and choose again whether to check or submit. Voters should perform the checking process again until they have enough confidence in the system to cast their vote as intended.

5.1.2 Vote is registered as cast

When the voter submits and casts an encrypted ballot (by submitting a cast request item as described in section 4.7.5), a vote confirmation receipt (ρ, σ, h) is returned as a response from the digital ballot box. The receipt contains a digital signature of the digital ballot box, which certifies that the voter's ballot has been registered on the public bulletin board. The receipt can be validated by checking $\mathsf{SigVer}(Y_{\mathcal{D}}, \rho; \sigma || h)$ (algorithm 29), where $Y_{\mathcal{D}}$ is the public key of the digital ballot box, σ is the voter's signature on the cast request item, and h is the address of the item on the bulletin board.

Anytime after casting a ballot, the voter can check the receipt against the bulletin board, which should find the appropriate ballot submission. Thus, the voter gains confidence that the vote is registered as cast.

If a voter has a valid receipt (i.e., which validates $\mathsf{SigVer}(Y_{\mathcal{D}}, \rho; \sigma || h)$ algorithm 29) that does not correspond to any item from the public bulletin board, then the tuple (ρ, σ, h) represents evidence that the integrity of the bulletin board has been compromised. The argument is that a previously accepted item has been removed or tampered with on the bulletin board.

5.2 Administration auditing process

This section describes the auditing steps that are available only to election officials because they are based on data that is not publicly available. These auditing processes verify the activity of specific components of the election system. The administration auditing processes give confidence to the election officials that the election is run correctly. Therefore, the result is trustworthy.

5.2.1 Eligibility verifiability

This auditing process verifies that only eligible voters have submitted ballots to the bulletin board, i.e., verifying that all voter session items have been authorized by the voter authorizer based on successful voter authentication. This process must be done before computing a result, so election officials validate the eligibility of the extracted ballots.

The auditing starts by providing the public key of the voter authorizer $Y_{\mathcal{A}}$, all eligible voter identities $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, ..., \mathcal{V}_{n_\text{v}}\}$, and the list of voter session items from the bulletin board $\{b_{\text{vs};1}, ..., b_{\text{vs};n_\text{vs}}\}$, where n_vs is the total number of voter session items. Recall that items have the following structure $b_{\text{vs};i} = (m_i, c_i, \mathcal{A}, \sigma_i, t_i, p_i, h'_i, h_i)$, with $i \in \{1, ..., n_\text{vs}\}$.

The auditor checks the integrity of each voter session item by $\mathsf{ItemVer}(b_{\text{vs};i}, Y_{\mathcal{A}})$ (algorithm 3) and that its content $c_i = (\mathcal{V}_i, Y_i, H_i)$ relates to an eligible voter, i.e. $\mathcal{V}_i \in \boldsymbol{\mathcal{V}}$.

Then, for identity-based voter authorization mode, the voter authorizer has to provide all identity tokens $\sigma_{\text{id};i,j}$ generated by each identity provider $\mathcal{I}_j \in \boldsymbol{\mathcal{I}}$, that were used to create the voter session item $b_{\text{vs};i}$. The auditor verifies that all voter session items have been authorized after successful authentication. The auditor checks that the identity tokens are associated with the voter session item $H_i = \mathcal{H}(\sigma_{\text{id};i,1}||...||\sigma_{\text{id};i,n_\text{i}})$, where H_i is the authentication fingerprint from the item content. Also, it checks the validity of the identity tokens by $\mathsf{SigVer}(Y_{\mathcal{I}_j}, \sigma_{\text{id};i,j}, \mathcal{V}_i)$ (algorithm 29). In case any of the validations fail, that discovers an attempt of the voter authorizer to create a fraudulent voter session.

Voter identities used for third-party identity providers are considered personal data and cannot be publicly disclosed on the bulletin board. Therefore, eligibility verifiability is a administrative auditing step and is available only to election officials.

For credential-based voter authorization mode, the auditor is provided with all voter authentication public keys $\{Y_{\text{auth};1}, ..., Y_{\text{auth};n_\text{v}}\}$ for each of the voters in $\boldsymbol{\mathcal{V}}$. The voter authorizer provides all proofs of credentials $\{PK_1, ..., PK_{n_\text{vs}}\}$ associated with each voter session item from the bulletin board. The auditor checks that $H_i = \mathcal{H}(PK_i)$ and $\mathsf{DLVer}(PK_i, \{G\}, \{Y_{\text{auth};i}\})$ (algorithm 15), where H_i is the authentication fingerprint from the content of the voter session item $b_{\text{vs};i}$ and $Y_{\text{auth};i}$ is the authentication public key of voter $\mathcal{V}_i$. In case one of the validations fails, that discovers an attempt of the voter authorizer to create a fraudulent voter session.

Recall from section 4.5 that the proof of credentials PK_i is initiated by credentials generated based on a minimum of 80 bits of entropy. Being so low on entropy, the voter authentication public keys and proofs of credentials are not publicly disclosed to prevent a brute-force attack. Therefore, they are auditable only by the election officials.

5.3 Public auditing process

Public auditing processes are accessible to anybody. They are used to validate that the entire election is run correctly. This audit is typically run at the end of the election period by certified auditors that will validate or invalidate an election result. Nevertheless, it could be run both during the election phase or when the election has finished by any public person with access to the public bulletin board and suitable verification algorithms.

As part of the public auditing, the following verification steps are included:

- During the election phase and after the election has finished, anybody can verify the integrity of the data published on the bulletin board, as explained in section 5.3.1.
- After a result has been initiated (i.e., an extraction data item has been published as in section 4.9.1), anybody can verify that the extraction procedure has been performed correctly, as explained in section 5.3.2.
- After a result has been computed and published, any public auditor can check the correctness of the result by verifying the result computation, as presented in section 5.3.3.

5.3.1 Integrity of the bulletin board

This verification step checks that only qualified actors have published items on the bulletin board. It also checks that no items have been removed or tampered with once posted on the bulletin board. This is achieved by checking the integrity of the hash structure of the bulletin board (referred to as the *history* property of the bulletin board in section 2.2) and by checking the validity of the digital signatures of each item.

Formally, given a complete bulletin board or a portion of it, in the form of a list of items $\boldsymbol{b} = \{b_1, ..., b_n\}$, any public auditor can check the integrity of the list by running $\mathsf{HistoryVer}(\boldsymbol{b}, h'_1)$ (algorithm 2), where h'_1 is the address of the previous item in the history. When $\boldsymbol{b}$ is a complete bulletin board (i.e., b_1 is a genesis item), then h'_1 must be equal to $\varnothing$, as the genesis item has no previous address.

Additionally, the auditor checks the correctness of the chosen parameters of each item $b_i \in \boldsymbol{b}$ (i.e., that it has a properly structured content c_i and that it references a proper parent p_i both according to the rules specified in appendix A). Then the auditor validates the integrity of each item by $\mathsf{ItemVer}(b_i, Y_{\mathcal{W}_i})$, where $Y_{\mathcal{W}_i}$ is the public key of the writer of the i^{th} item. Note that, as described in appendix A, depending on the type of item, one of the following actors can be the writer of an item: the digital ballot box $\mathcal{D}$, the election administrator $\mathcal{E}$, the voter authorizer $\mathcal{A}$ or a specific voter $\mathcal{V}$. The public key of any of these actors must be retrieved from the content of the bulletin board itself, such as:

- the public keys of the election administrator $Y_{\mathcal{E}}$ and of the digital ballot box $Y_{\mathcal{D}}$ are listed in the genesis item,
- the public key of the voter authorizer $Y_{\mathcal{A}}$ is listed in an actor configuration item that defines the voter authorizer role,
- each public key Y_j of the j^{th} voter is introduced by a voter session item.

If any verification steps mentioned above fail, then $\boldsymbol{b}$ does not represent a valid bulletin board trace.

5.3.2 Verification of the extraction procedure

Given a bulletin board trace $\boldsymbol{b}$, with an extraction data item included in $\boldsymbol{b}$, any public auditor can verify the correctness of the list of cryptograms $\vec{\vec{e}}_0$ present in the extraction data item. Recall from section 4.9.1 that $\vec{\vec{e}}_0$ lists all votes that will make up the election result.

The auditor reruns the extraction procedure on the bulletin board $\boldsymbol{b}$, applying all filtering rules specified in section 4.9.1 to recompute the *initial mixed board* $\vec{\vec{e}}_0'$. If it is identical with $\vec{\vec{e}}_0$, then the extraction has been performed correctly.

5.3.3 Result verification

After a result has been published via an extraction confirmation item (as described in section 4.9.4), any public auditor can verify the correctness of the result by verifying the mixing and decryption procedures of each trustee that participated in the result ceremony. This checks that no votes have been tampered with, removed or added during mixing and decryption. The data that an auditor needs to collect comes exclusively from the bulletin board:

- the initial mixed board $\vec{\vec{e}}_0$ is listed in the extraction data item,
- the encryption keys Y_{enc} and the list of trustees $\boldsymbol{\mathcal{T}} = \{\mathcal{T}_1, ..., \mathcal{T}_{n_\text{t}}\}$, together with their public keys $\{Y_{\mathcal{T}_1}, ..., Y_{\mathcal{T}_{n_\text{t}}}\}$ and public polynomial coefficients $\{P_{\mathcal{T}_1,1}, ..., P_{\mathcal{T}_{n_\text{t}},t-1}\}$, are listed in the threshold configuration item, where n_t is the amount of trustees,
- the subset of trustees that participated in the mixing and decryption phases $\tau \subset \{1, ..., n_\text{t}\}$ is listed in the extraction confirmation item,
- all the intermediate mixed boards that trustees produced $\vec{\vec{e}}_i$, with $i \in \tau$, and their respective proofs of correct mixing (PM_i, AS_i) are listed in the extraction confirmation item,
- all the partial decryptions that trustees produced $\vec{\vec{S}}_i$, with $i \in \tau$, and their respective proofs of correct decryption PK_i are listed in the extraction confirmation item,

- the raw result $\vec{\vec{V}}$ is listed in the extraction confirmation item.

First, the auditor validates each intermediate mixed board by running algorithm 37 $\mathsf{MixVer}(PM_i, AS_i, Y_{\text{enc}}, \vec{\vec{e}}_{i-1}, \vec{\vec{e}}_i)$, for each $i \in \tau$. Next, the auditor validates the proof of each partial decryption by $\mathsf{PartialDecryptionVer}(\vec{\vec{e}}, \vec{\vec{S}}_i, PK_i, sY_i)$ (algorithm 7), for each $i \in \tau$, where sY_i is the public share of the trustee $\mathcal{T}_i$ computed as in appendix B.5, and $\vec{\vec{e}}$ is the mixed board produced by the last trustee in τ.

If all partial decryptions are valid, the auditor aggregates them together to recompute the raw result $\vec{\vec{V}} \leftarrow \mathsf{FinalizeDecryption}(\vec{\vec{e}}, \{\vec{\vec{S}}_i | i \in \tau\})$ (algorithm 8). The list of decrypted votes $\vec{\vec{V}}$ should be identical to the one published in the result confirmation item. If any validation step fails, the result is considered untrustworthy.

By having the election result auditable, the election protocol achieves one of the end-to-end verifiability properties, i.e., verification that all votes have been counted as registered.

6 Conclusion

This article presents a new election protocol designed to be verifiable from end to end, protect against certain attacks, ensure the security and privacy of election data, and confirm voter eligibility while maintaining their anonymity. The design distributes tasks and responsibilities among multiple groups to limit an attacker's influence. Although it can't defend against attackers with almost unlimited computing power or the ability to break cryptographic primitives, we are developing a subsequent version to address these limitations. Additionally, the current protocol doesn't guarantee perpetual privacy, as an attacker controlling a majority of trustees could compromise anonymity and access confidential data. This limitation represents another area for improvement.

Details on the methodologies employed to actualize each property are provided below:

Voter Eligibility: During the election phase, only a predefined set of voters are allowed to cast a ballot on the bulletin board. The list of eligible voters $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, ..., \mathcal{V}_{n_\mathrm{v}}\}$ is defined, during the pre-election phase, in the voter authorizer service, which authorizes the use of a public key Y_i correlated with voter identity $\mathcal{V}_i$. The public key authorization is done by publishing a voter session item on the bulletin board after voter $\mathcal{V}_i$ has successfully authenticated. All voter authorizations performed by the voter authorizer are auditable as described in section 5.2.

When voter athorization mode is identity-based, voters successfully authenticate to the voter authorizer by authenticating to all identity providers $\boldsymbol{\mathcal{I}} = \{\mathcal{I}_1, ..., \mathcal{I}_{n_\mathrm{i}}\}$. That means, to falsely acquire an authorized public key (to cast a vote with), one must forge successful authentication with all identity providers $\boldsymbol{\mathcal{I}}$. Therefore, our protocol has the eligibility property on the assumption that there is at least one honest identity provider.

In case an election is configured to use only one identity provider (i.e., $n_\mathrm{i} = 1$), then that identity provider could, in fact, authenticate and cast a vote on behalf of any voter. Therefore, if voter authentication is provided by a single identity provider, that must be assumed trustworthy.

When voter athorization mode is credential-based, voters successfully authenticate to the voter authorizer by submitting a proof of credentials, which is calculated based on all credentials received from all credentials authorities $\boldsymbol{\mathcal{C}} = \{\mathcal{C}_1, ..., \mathcal{C}_{n_\mathrm{c}}\}$ during the pre-election phase (as described in section 4.5). Therefore, our protocol has the eligibility property on the assumption that there is at least one honest credentials authority.

In case an election is configured to use only one credentials authority (i.e., $n_\mathrm{c} = 1$), then it could, in fact, authenticate and cast a vote on behalf of any

voter (as it knows all credentials of all voters). Therefore, if voter credentials are provided by a single credentials authority, that must be assumed trustworthy.

Votes Privacy: All votes that are posted on the bulletin board are encrypted using the ElGamal cryptosystem based on elliptic curve cryptography. Moreover, using a *t out of n* threshold decryption scheme, entails that the decryption of ballots form the bulletin board is possible only when at least t trustees are willing to collaborate in a result computation.

Therefore, we claim that the protocol reaches the *privacy* property on the assumption that there are at least t honest trustees, with $2/3 \cdot n \leq t \leq n$.

One can argue that, because the bulletin board data is public, somebody could save all the data for long enough in the eventuality that the elliptic curve cryptography might get broken or trustee keys get leaked. Then, voting data could be decrypted contrary to our protocol. This indicates that demonstrates that our protocl does not reach *everlasting privacy*. We take note of this fact and accept it. As a mitigation measure, processes that envolve the use of trustee private keys are performed in a air-gapped network, as described in **??**. This lowers the risk of keys being leaked, as they are never exposed to an internet-connected machine.

Votes Anonymity: All votes are shuffled during the mixing phase before they get decrypted, as describedin section 4.9.2. The mixing process is performed by a mix-net of trustees that sequencially rearrange the list of votes in a indistinguishable way. As a result of this process, when decrypted, there is no connection between a plain-text vote and its voter.

Obviously, each trustee knows how it shuffled the list of ballots but does not know how the other trustees shuffled it. Thus, it is crucial that trustees do not collude and communicate with each other the shuffling parameters. We claim that our protocol achieves the *anonymity* property on the assumption that there is at least one honest trustee in the process.

However, ballot anonymity is bound by the amount of ballots that are mixed together, which happens during each extraction. The most anonymity is reached when all ballots are mixed together (i.e., there is a single result extraction at the end of the election phase). Nevertheless, even if all ballots are mixed in a single extraction, but there are only two ballots in total, and both contain a vote for the same candidate, the anonymity is reduced to nothing, as both ballots are identifiable.

Therefore, it is essential to require a result extractions to happen only on a list with a substantial amount of ballots. This can be configured through the extraction threshold value t_e set during the pre-election phase.

Ballot Replacement: Voters have the ability to perform the voting protocol section 4.7 multiple times. Each time, they have to authenticate, get authorized, encrypt a ballot and cast. If a voter has cast multiple ballots, only the last submitted one is selected in the extraction process section 4.9.1 to be tallied.

Note that, ballot replacement is possible only until a partial result has extracted one of that voter's ballots. Hereafter, that voter is not allowed to cast another ballot as it will not be tallied.

Data Integrity The data published on the bulletin board is publicly available and it consists of the election configuration, voting data and result documentation. The data is stored as a *hash-chain* structure and, because of the *history* and *ancestry* properties of the bulletin board (presented in section 2.2), any attempts to tamper with the data is detectable by running the public audit process section 5.3.

Every time a new item is appended on the bulletin board, it is accompanied by a digital signature that proves the authorship of the data. During the public auditing process, each item signature is verified, thus confirming the origin of the data.

Therefore, assuming that genuine auditing software is used, we claim that our protocol achieves the *data integrity* property. Both origin and authenticity properties of the data can be validated due to the bulletin board construction.

Receipt-Freeness During the vote cryptogram generation process, described in section 4.7.4, the voter and the digital ballot box collaborate together to build the encrypted ballot $\vec{e}$. During this process, the value of randomizer used in the encryption of the vote (required by the ElGamal cryptosystem appendix B.4) is split amongst the voter and the digital ballot box.

After the voter chooses to cast the ballot, as presented in section 4.7.5, the digital ballot box destroys its part of the randomizer. The voter is now unable to reproduce the entire value of the randomizer used to encrypt the ballot. As a consequence, the voter is not able to produce cryptographic evidence that $\vec{e}$ is an encryption of a particular vote $\vec{V}$ using only the publicly available data.

Therefore, we claim that our election protocol has the *receipt-freeness* property.

End-to-End Verifiability There are three levels of verifiability that different actors can perform. Some steps are individually verifiable (i.e., only the voter that is currently performing this step can verify that the process is happening correctly), such as:

- verify that the vote is cast as intended by performing the ballot checking process, described in section 4.8,

- verify that the vote is registered as cast by checking the vote confirmation receipt as described in section 4.7.5 and section 5.1.2.

Some aspects of the election protocol are accessible only to election officials by running the audits from section 5.2. These include validating:

- the voter eligibility,

Some other aspects of the election protocol are publicly verifiable. In other words, even an active voter that has cast a ballot can validate the following:

- the integrity of the bulletin board,
- the fact that ballots have been correctly extracted and included in a result,
- the correctness of the result computation.

Our protocol provides mechanisms to verify the correct processing of a vote throughout the protocol: form being cast and encrypted, to being registered on the bulletin bord, to being extracted in a result. Therefore, we argue that our election protocol is *end-to-end verifiable*.

Vote & Go An election result can be computed only by at least a threshold of the trustees that must collaborate in a result computation. Voters are not required in a result computation. One might say that voters can perform the audit step that verifies whether their vote has been extracted only after a result has been computed. That is, though, an optional step and it does not affect the election protocol.

Replay Protection Given the bulletin board construction, the ancestry property of the bulletin board entails that items must be appended in a particular order. Recall form section 2.2 that they reference each other through the parent address. At the same time, each address is unique, given the collision resistance property of the hashing function $\mathcal{H}$, as presented in appendix B.2.

Therefore, even if a significant amount of the activity that happens during the election protocol is published on the bulletin board, it has replay protection.

In summary the proposed protocol provides a solid foundation for creating a secure, verifiable digital voting system.

References

[1] J. Heather and D. Lundin, "The append-only web bulletin board," in *Formal Aspects in Security and Trust*, P. Degano, J. Guttman, and F. Martinelli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 242–256.

[2] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, 1st ed. New York: Springer, 2004.

[3] J. Benaloh, "Simple verifiable elections," in *Proceedings of the USENIX Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, ser. EVT'06. Berkeley, CA, USA: USENIX Association, 2006, pp. 5–5. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251003.1251008

[4] Y. G. Desmedt and Y. Frankel, "Threshold cryptosystems," in *Proceedings on Advances in Cryptology*, ser. CRYPTO '89. New York, NY, USA: Springer-Verlag New York, Inc., 1989, pp. 307–315. [Online]. Available: http://dl.acm.org/citation.cfm?id=118209.118237

[5] W. Trappe and L. C. Washington, *Introduction to Cryptography with Coding Theory (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2005.

[6] N. I. of Standards and Technology, "Secure hash standard (shs)," 2012.

[7] S. S. M. Chow, C. Ma, and J. Weng, "Zero-knowledge argument for simultaneous discrete logarithms," in *Computing and Combinatorics*, M. T. Thai and S. Sahni, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 520–529.

[8] T. P. Pedersen, "A threshold cryptosystem without a trusted party," in *Advances in Cryptology — EUROCRYPT '91*, D. W. Davies, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 522–526.

[9] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979. [Online]. Available: http://doi.acm.org/10.1145/359168.359176

[10] N. I. of Standards and Technology, "Advanced encryption standard," *NIST FIPS PUB 197*, 2001.

[11] D. H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869, May 2010. [Online]. Available: https://www.rfc-editor.org/info/rfc5869

[12] K. Moriarty, B. Kaliski, and A. Rusch, "PKCS #5: Password Based Cryptography Specification Version 2.1," RFC 8018, Jan. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8018

[13] O. O. W. A. S. P. Foundation, "OWASP cheat sheet series," https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#pbkdf2, 2023.

[14] C. P. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology — CRYPTO' 89 Proceedings*, G. Brassard, Ed. New York, NY: Springer New York, 1990, pp. 239–252.

[15] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology — CRYPTO '91*, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140.

[16] J. Bootle and J. Groth, "Efficient batch zero-knowledge arguments for low degree polynomials," Cryptology ePrint Archive, Report 2018/045, 2018, https://ia.cr/2018/045.

[17] J. Groth, "A verifiable secret shuffle of homomorphic encryptions," *IACR Cryptol. ePrint Arch.*, p. 246, 2005. [Online]. Available: http://eprint.iacr.org/2005/246

[18] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194.

A Bulletin board item types

All item types that can appear on the bulletin board are described in the following list and are grouped into the following four categories:

Item	Writer	Content	Parent type	Validation rules
genesis	$\mathcal{D}$	elliptic curve domain parameters (p, a, b, G, q, h), digital ballot box public key $Y_{\mathcal{D}}$, election admin public key $Y_{\mathcal{E}}$	none	It is the first item on the board.
election configuration	$\mathcal{E}$	election title, enabled languages	latest configuration item	The first item defines the configuration. The following items update the configuration.
contest configuration	$\mathcal{E}$	contest identifier, contest marking rules, question type, and result rules, candidate labels $\{m_1, ..., m_{n_c}\}$	latest configuration item	The first item with a contest identifier defines the configuration of that contest. The following items with the same contest identifier update the configuration of that specific contest.
threshold configuration	$\mathcal{E}$	ballot encryption key Y_{enc}, threshold setup t out-of n_{t}, trustees public keys $\{Y_{\mathcal{T}_1}, ..., Y_{\mathcal{T}_{n_{\text{t}}}}\}$, trustees public polynomial coefficients $\{P_{\mathcal{T}_1,1}, ..., P_{\mathcal{T}_{n_{\text{t}}},t-1}\}$	latest configuration item	This item cannot be updated.

Item	Writer	Content	Parent type	Validation rules
actor configuration	$\mathcal{E}$	actor identifier, actor role, actor public key	latest configuration item	The first item with an actor identifier defines the configuration of that actor. The following items with the same actor identifier update the configuration of that specific actor. The role can be: *Voter Authorizer* $\mathcal{A}$.
voter authorization configuration	$\mathcal{A}$	the voter authorization mode, configuration of all Identity Providers $\{\mathcal{I}_1, ..., \mathcal{I}_{n_i}\}$	latest configuration item	The first item defines the voter authorization configuration. The following items update the configuration.
voting round	$\mathcal{E}$	voting round identifier, start date and end date, list of enabled contest identifiers	latest configuration item	The first item with a voting round identifier defines the configuration of that voting round. The following items with the same voting round identifier update the configuration of that specific voting round.

Configuration items

Item	Writer	Content	Parent type	Validation rules
voter session	$\mathcal{A}$	voter identifier, voter public key Y_i, voter weight, voter authentication fingerprint list of assigned contest identifiers	latest configuration item	This item can be created only during the election phase. The following voter session items with the same voter identifier overwrite the previous voter sessions of that voter.
voter encryption commitment	$\mathcal{V}_i$	commitment c_v	the voter session item	The voter's public key Y_i is defined in the voter session item.
server encryption commitment	$\mathcal{D}$	commitment c_d	the voter encryption commitment item	Only one server encryption commitment item can reference the voter encryption commitment item. This item is created in response to the voter encryption commitment item being published.
ballot cryptograms	$\mathcal{V}_i$	cryptograms $\vec{e}_i$	the server encryption commitment item	Only one ballot cryptograms item can reference the server encryption commitment item. The voter's public key Y_i is defined in the voter session item.
cast request	$\mathcal{V}_i$		the ballot cryptograms item	There can be either a cast request or a spoil request item referencing the ballot cryptograms item. The voter's public key Y_i is defined in the voter session item.

Item	Writer	Content	Parent type	Validation rules
spoil request	$\mathcal{V}_i$		the ballot cryptograms item	There can be either a cast request or a spoil request item referencing the ballot cryptograms item. The voter's public key Y_i is defined in the voter session item.

Voting items

Item	Writer	Content	Parent type	Validation rules
verification track start	$\mathcal{D}$		the ballot cryptograms item	Only one verification track start item can reference the ballot cryptograms item. This item is created in response to the ballot cryptograms item being published.
verifier	$\mathcal{X}$	external verifier's public key $Y_{\mathcal{X}}$	the verification track start item	This is a self-signed item, i.e., the author's public key is defined in the item itself. Only one verifier item can reference the verification track start item.
voter commitment opening	$\mathcal{V}_i$	encrypted commitment opening d_{v}	the verifier item	Only one voter commitment opening item can reference the verifier item.
server commitment opening	$\mathcal{D}$	encrypted commitment opening d_{d}	the voter commitment opening item	Only a server commitment opening item can reference the voter commitment opening item.

Hidden items

Item	Writer	Content	Parent type	Validation rules
extraction intent	$\mathcal{E}$		latest config item	
extraction data	$\mathcal{D}$	a fingerprint of the matrix of cryptograms $\vec{\vec{e}}_0$	the extraction intent item	Only one extraction data item can reference the extraction intent item. The item provides a way of aquiring the list $\vec{\vec{e}}_0 = \{e_1, ..., e_{n_e}\}$.
extraction confirmation	$\mathcal{E}$	list of trustees that participated in the result ceremony $\tau \subset \{1, ..., n_t\}$, fingerprints of each intermediate mixed boards of cryptograms $\vec{\vec{e}}_i$ and proofs of correct mixing (PM_i, AS_i), fingerprints of each partial decryption $\vec{\vec{S}}_i$ and proofs of correct decryption PK_i, signatures from each trustee $\mathcal{T}_i$ on all the fingerprints above, where $i \in \tau$	the extraction data item	Only one extraction confirmation item can reference the extraction data item.

Result items

B Algorithms background

The following is a compilation of algorithms utilized in the protocol, which are neither developed nor maintained by our team. These are well-established third-party algorithms that have been thoroughly examined and have subsequently achieved the status of industry standard.

B.1 Elliptic Curve

B.1.1 Supported elliptic curves

The following elliptic curves are supported by our protocol. Each curve defines a different set of the domain parameters (p, a, b, G, q, h).

- secp256k1,
- secp256r1,
- secp384r1,
- secp521r1.

B.1.2 Mapping a message on the Elliptic Curve

The algorithms of encoding and decoding a message into an elliptic curve point is inspired from *Introduction to Cryptography with Coding Theory* [5]. Any message represented as a byte array $\vec{b}$ with length $|\vec{b}| \leq \ell - 1$ can be converted into an elliptic curve point by $M \leftarrow \mathsf{Bytes2Point}(\vec{b})$ (algorithm 12). The byte array $\vec{b}$ is prepended with an adjusting byte $b_0 = 00$ and appended (padded) with 00 bytes such that it has a length of ℓ, which is the elliptic curve byte size, i.e., $\ell \leftarrow \mathsf{ByteLengthOf}(p)$ (algorithm 11), where p is the prime of the elliptic curve. The resulting byte array is prepended with the flag byte 02 and decoded as an elliptic curve point $M \leftarrow \mathsf{DecodePoint}(b)$ (algorithm 9). If successful, then M is the encoding of $\vec{b}$. Otherwise, the algorithm modifies x by incrementing the adjusting byte and retries 255 times. If no valid point is found, the algorithm returns failure.

By having one *byte space* to find a valid point on the curve, [5] shows that the probability of all 256 x coordinates to generate non-valid points is $1/2^{256}$, which is considered acceptable. Formally, $M \leftarrow \mathsf{Bytes2Point}(\vec{b})$ (algorithm 12) converts any message $\vec{b}$ of legal size, i.e., $|\vec{b}| \leq \ell - 1$ into a valid elliptic curve point M with a negligible failure rate.

Recovering the message from an elliptic curve point M can be done by calling $\vec{b} \leftarrow \mathsf{Point2Bytes}(M)$ (algorithm 13). It decodes point M as a byte array $b \leftarrow \mathsf{EncodePoint}(M)$ (algorithm 10), disregards the rightmost 00 bytes and the adjusting byte b_0 and returns the rest of b as the message.

Note that we use standard algorithms (algorithms 9 and 10) from [5] for encoding and decoding elliptic curve points as byte arrays in compact form. Therefore, the inner implementation is absent.

Algorithm 9: $\mathsf{DecodePoint}(\vec{b}) \to M$

Data: The byte array $\vec{b} \in \mathbb{B}^{\ell+1}$
Result: The point $M \in \mathbb{P}$ or failure

Algorithm 10: $\mathsf{EncodePoint}(M) \to \vec{b}$

Data: The point $M \in \mathbb{P}$
Result: The encoded byte array $\vec{b} \in \mathbb{B}^{\ell+1}$

Algorithm 11: $\mathsf{ByteLengthOf}(x) \to n$

Data: The scalar $x \in \mathbb{Z}$
Result: The byte length $n \in \mathbb{N}$

Having these two algorithms, mapping a message on the Elliptic Curve is a sound procedure as $\vec{b} = \mathsf{Point2Bytes}(\mathsf{Bytes2Point}(\vec{b}))$, for all $\vec{b} \in \mathbb{B}^*$, with $|\vec{b}| \leq \ell - 1$.

Examples of ℓ values, depending on elliptic curves, are:

- $\ell = 32$ for **Secp256k1**,
- $\ell = 32$ for **Secp256r1**,
- $\ell = 48$ for **Secp384r1**,
- $\ell = 65$ for **Secp521r1**.

B.2 Hash functions

A *cryptographic hash function* is an algorithm used for mapping data of arbitrary size to data of fixed size, also called the *hash value*. A hash function is known as a *one-way function*, i.e., one can easily verify that some input data maps to a given hash value, but if the input data is unknown, it is infeasible to calculate it given only a hash value. Another property of a cryptographic hash function is *collision resistance*, which means that it is infeasible to find two different input data with the same hash value.

We use 3 standard hash function as described in *Secure Hash Standards* [6], with 3 output lengths, that match the byte size of the supported elliptic curves:

- SHA256 for **Secp256k1** and **Secp256r1**,
- SHA384 for **Secp384r1**,
- SHA512 for **Secp521r1**.

Algorithm 12: Bytes2Point($\vec{b}$)

Data: The byte array $\vec{b} = \{b_1, ..., b_n\} \in \mathbb{B}^n$
$\ell \leftarrow$ ByteLengthOf(p) // algorithm 11
if $n > \ell - 1$ **then**
  return *failure*
end
$m \leftarrow \ell - n - 1$
for $i \leftarrow 0$ **to** 255 **by** 1 **do**
  $b_0 \leftarrow i$
  $\vec{b}' \leftarrow \{02, b_0, b_1, ..., b_n, \underbrace{00, ..., 00}_{m \text{ times}}\}$
  if $M \leftarrow$ DecodePoint($\vec{b}'$) *is successful* **then**
    return M // $M \in \mathbb{P}$
  end
end
return *failure*

Algorithm 13: Point2Bytes(M)

Data: The point $M \in \mathbb{P}$
$\ell \leftarrow$ ByteLengthOf(p) // algorithm 11
$\vec{b}' = \{02|03, b_0, b_1, ..., b_n, \underbrace{00, ..., 00}_{m \text{ times}}\} \leftarrow$ EncodePoint(M) // algorithm 10
$\vec{b} \leftarrow \{b_1, ..., b_n\}$ // $\ell = m + n + 1$
return $\vec{b}$ // $\vec{b} \in \mathbb{B}^*$

We abuse notation and define the hash function $\mathcal{H}(data) \rightarrow x$, where $data \in \mathbb{B}^*$ and $x \in \mathbb{Z}_p$ is the output of the hash function interpreted as a field element of the elliptic curve in use.

B.3 Discrete Logarithm Proof

A discrete logarithm proof is a zero knowledge proof that proves knowledge of value x, such that $Y = [x]G$, without revealing any other information about x. Formally $PK[(x) : Y = [x]G]$. The most intuitive application of this could be to prove the possession of the private key associated with a public key.

A bit more complex ZKP is the *discrete logarithm equality proof* that proves that two different elliptic curve points $Y, P \in \mathbb{P}$ have the same elliptic curve discrete logarithm $x \in \mathbb{Z}_q$ in regards to two distinct generators $G, H \in \mathbb{P}$, formally $PK[(x) : Y = [x]G \wedge P = [x]H]$.

An optimization in proving the discrete logarithm equality between multiple points regarding their generators has been described in [7]. Using the optimized

algorithm to prove that

$$PK[(x) : \bigwedge_{i=0}^{n} Y_i = [x]G_i]$$

one can generate the proof $PK = (K, c, r)$ by following the protocol described in figure 11. The optimization consists of the fact that the commitment K is just one point regardless of the value of n.

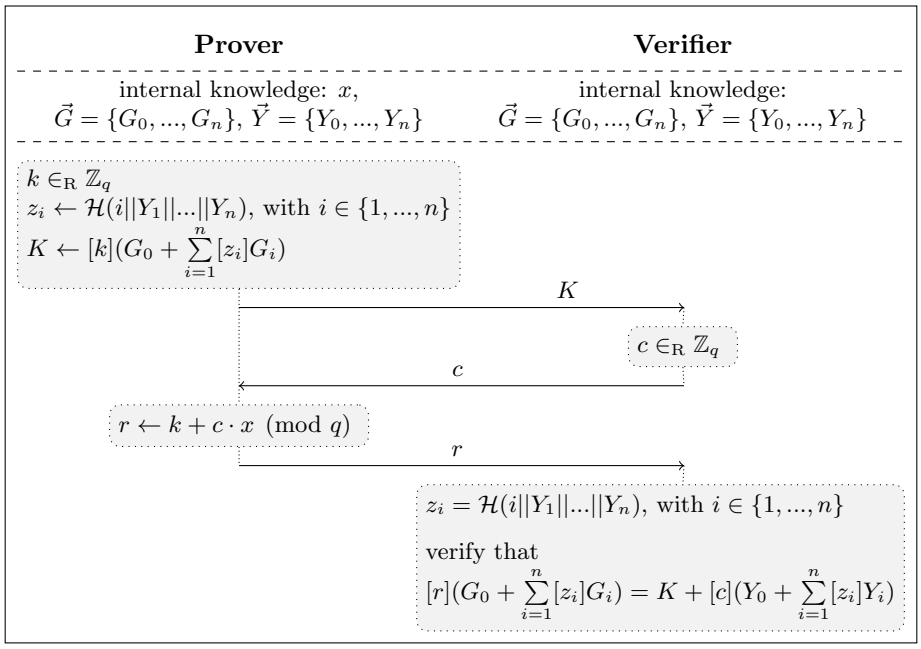


Figure 11: Protocol for proving multiple discrete logarithms

The proof of multiple discrete logarithms can be turned into a non-interactive one by computing the challenge of the proof using a hash function based on the available proof transcript (the generators, the commitment and the targeted points). The proof is generated by the algorithm $PK \leftarrow \mathsf{DLProve}(x, \vec{G})$ (algorithm 14), where $\vec{G} = \{G_0, ...G_n\}$ is the list of generators.

A public verifier accepts the proof if the algorithm $\mathsf{DLVer}(PK, \vec{G}; \vec{Y})$ returns true, where $\vec{Y} = \{Y_0, ..., Y_n\}$. The verification algorithm is described in algorithm 15.

Algorithm 14: $\mathsf{DLProve}(x, \vec{G})$

Data: The private key $x \in \mathbb{Z}_q$
The list of generators $\vec{G} = \{G_0, G_1, ..., G_n\} \in \mathbb{P}^{n+1}$
$k \in_{\mathrm{R}} \mathbb{Z}_q$
for $i \leftarrow 1$ **to** n **by** 1 **do**
  $z_i \leftarrow \mathcal{H}(i||Y_1||...||Y_n)$ // $Y_j = [x]G_j\,,\ j \in \{1, ..., n\}$
end
$K \leftarrow [k](G_0 + \sum_{i=1}^{n}[z_i]G_i)$
$c \leftarrow \mathcal{H}(\vec{G}||K||\vec{Y})$
$r \leftarrow k + c \cdot x \pmod q$
$PK \leftarrow (K, c, r)$
return PK // $PK \in \mathbb{P} \times \mathbb{Z}_q \times \mathbb{Z}_q$

Algorithm 15: $\mathsf{DLVer}(PK, \vec{G}; \vec{Y})$

Data: The proof $PK = (K, c, r) \in \mathbb{P} \times \mathbb{Z}_q \times \mathbb{Z}_q$
The list of generators $\vec{G} = \{G_0, G_1, ..., G_n\} \in \mathbb{P}^{n+1}$
The list of public keys $\vec{Y} = \{Y_0, Y_1, ..., Y_n\} \in \mathbb{P}^{n+1}$
for $i \leftarrow 1$ **to** n **by** 1 **do**
  $z_i \leftarrow \mathcal{H}(i||Y_1||...||Y_n)$
end
if $c = \mathcal{H}(\vec{G}||K||\vec{Y})$
and $[r](G_0 + \sum_{i=1}^{n}[z_i]G_i) = K + [c](Y_0 + \sum_{i=1}^{n}[z_i]Y_i)$ **then**
  $b \leftarrow 1$ // proof is valid
else
  $b \leftarrow 0$ // proof is invalid
end
return b // $b \in \mathbb{B}$

B.4 Elgamal cryptosystem

B.4.1 Encryption scheme

The *Elgamal cryptosystem* is an asymmetric, randomized encryption scheme where anybody can encrypt a message using the encryption key, resulting in a *cryptogram*. In contrast, only the one with the decryption key can extract the message of a cryptogram. The scheme consists of three algorithms KeyGen, Enc, Dec.

An Elgamal key pair is a tuple $(x, Y) \leftarrow \mathsf{KeyGen}()$ (algorithm 16), where x is a randomly chosen scalar representing the private decryption key and Y is an elliptic curve point corresponding to the public encryption key.

The encryption algorithm $e = (R, C) \leftarrow \mathsf{Enc}(Y, M; r)$ (algorithm 17) can be

Algorithm 16: KeyGen()

$x \in_{\mathrm{R}} \mathbb{Z}_q$
$Y \leftarrow [x]G$
return (x, Y) // $(x, Y) \in \mathbb{Z}_q \times \mathbb{P}$

used by anybody in possession of the public encryption key Y to generate a cryptogram on a message M, using the randomizer r. The cryptogram e can be decrypted back to the original message M only by the one in possession of the private decryption key x in the decryption algorithm $M \leftarrow \mathsf{Dec}(x, e)$ (algorithm 18). Note that both Enc and Dec work on messages that are formatted as elliptic curve points $M \in \mathbb{P}$.

For the sake of notation, we define $\mathbb{E} = \mathbb{P} \times \mathbb{P}$ as the set of all possible cryptograms.

Algorithm 17: $\mathsf{Enc}(Y, M; r)$

Data: The encryption key $Y \in \mathbb{P}$
The message $M \in \mathbb{P}$
The randomizer $r \in \mathbb{Z}_q$
$R \leftarrow [r]G$
$S \leftarrow [r]Y$
$C \leftarrow S + M$
$e \leftarrow (R, C)$
return e // $e \in \mathbb{E}$

Algorithm 18: $\mathsf{Dec}(x, e)$

Data: The decryption key $x \in \mathbb{Z}_q$
The cryptogram $e = (R, C) \in \mathbb{E}$
$S \leftarrow [x]R$
$M \leftarrow C - S$
return M // $M \in \mathbb{P}$

B.4.2 Homomorphic Encryption

Elgamal encryption is a *homomorphic* encryption scheme concerning point addition. That means the component-wise addition of two cryptograms would result in a new, valid cryptogram containing the two messages summed up.

$$\mathsf{Enc}(Y, M_1; r_1) + \mathsf{Enc}(Y, M_2; r_2) = \mathsf{Enc}(Y, M_1 + M_2; r_1 + r_2)$$

The resulting encryption of the homomorphic addition of two cryptograms is $e' = (R', C') \leftarrow \mathsf{HomAdd}(e_1; e_2)$ (algorithm 19).

Algorithm 19: $\mathsf{HomAdd}(e_1; e_2)$

Data: The first cryptogram $e_1 = (R_1, C_1) \in \mathbb{E}$
The second cryptogram $e_2 = (R_2, C_2) \in \mathbb{E}$
$R' \leftarrow R_1 + R_2$
$C' \leftarrow C_1 + C_2$
$e' \leftarrow (R', C')$
return e' // $e' \in \mathbb{E}$

Following the procedure above, a given cryptogram $e = (R, C)$ can be *re-encrypted* by homomorphically adding it to an *empty cryptogram* (i.e., an encryption of the neutral point $\mathcal{O}$) with randomizer $r' \in_{\mathrm{R}} \mathbb{Z}_q$. The result is a new, randomly different cryptogram that contains the same message M. Generating the new cryptogram $e' = (R', C') \leftarrow \mathsf{ReEnc}(Y, e; r')$ is described by algorithm 20.

Algorithm 20: $\mathsf{ReEnc}(Y, e; r')$

Data: The encryption key $Y \in \mathbb{P}$
The initial cryptogram $e = (R, C) \in \mathbb{E}$
The new randomizer $r' \in \mathbb{Z}_q$
$e_2 \leftarrow \mathsf{Enc}(Y, \mathcal{O}; r')$ // algorithm 17
$e' \leftarrow \mathsf{HomAdd}(e, e_2)$ // algorithm 19
return e' // $e' \in \mathbb{E}$

Naturally, cryptogram addition can be expanded to multiplication to achieve the fact that $\mathsf{Enc}(Y, M; r) + \mathsf{Enc}(Y, M; r) = 2 \cdot \mathsf{Enc}(Y, M; r) = \mathsf{Enc}(Y, [2]M; 2 \cdot r)$. The resulting encryption of the homomorphic multiplication of a cryptogram is $e' = (R', C') \leftarrow \mathsf{HomMul}(e; n)$ (algorithm 21).

Algorithm 21: $\mathsf{HomMul}(e; n)$

Data: The initial cryptogram $e = (R, C) \in \mathbb{E}$
The multiplication factor $n \in \mathbb{Z}$
$R' \leftarrow [n]R$
$C' \leftarrow [n]C$
$e' \leftarrow (R', C')$
return e' // $e' \in \mathbb{E}$

B.5 Threshold Cryptosystem

A t out of n threshold cryptosystem is a homomorphic encryption scheme where the decryption key is split among n key holders, called *trustees* $\boldsymbol{\mathcal{T}} = \{\mathcal{T}_1, ..., \mathcal{T}_n\}$. Anybody can encrypt a message using the encryption key. The decryption of a message happens during a process in which at least t trustees have to collaborate in a cryptographic protocol. It is recommended that $t \geq 2/3 \cdot n$. The entire

scheme was introduced in [8], which is based on mathematical principles of the threshold cryptosystem [4, 9].

The key generation process concludes with the following $(sx_1, ..., sx_n, Y)$, where Y is the public encryption key, and each sx_i is a private share of the decryption key, one for each of the n trustees. The process is performed by all trustees while being facilitated by a central entity called *the server*. The entire process is described by the protocol called *the threshold ceremony* illustrated in figure 12.

During the *threshold ceremony*, each trustee $\mathcal{T}_i \in \boldsymbol{\mathcal{T}}$ generates a private-public key pair $(x_i, Y_i) \leftarrow \mathsf{KeyGen}()$ (algorithm 16) and publishes the public key to the server. The public encryption key is computed by the sum of the public keys of all trustees $Y = \sum_{i=1}^{n} Y_i$, while nobody knowing the decryption key $x = \sum_{i=1}^{n} x_i$ because all x_i are secret. Instead, all trustees work together to distribute x such that any t trustees can find it when necessary.

Each trustee $\mathcal{T}_i \in \boldsymbol{\mathcal{T}}$ generates a polynomial function of degree $t-1$

$$f_i(z) = x_i + p_{i,1} \cdot z + ... + p_{i,t-1} \cdot z^{t-1}$$

and publishes to the server the points $\{P_{i,1}, ..., P_{i,t-1}\}$, where each private-public coefficient pair is $(p_{i,k}, P_{i,k}) \leftarrow \mathsf{KeyGen}()$ (algorithm 16), with $k \in \{1, ..., t-1\}$.

When all public coefficients have been published, each trustee $\mathcal{T}_i \in \boldsymbol{\mathcal{T}}$ computes a *partial secret share of the decryption key* for each of the other trustees by $s_{i,j} \leftarrow f_i(j)$, where $j \in \{1, ..., n\}$. Then, $\mathcal{T}_i$ encrypts each partial secret share with a key derived from the Diffie-Hellman key exchange mechanism with each of the other trustees' public keys, i.e., $c_{i,j} \leftarrow \mathsf{SymEnc}(k_{i,j}, s_{i,j})$ (algorithm 22), where $k_{i,j} \leftarrow \mathsf{DerSymKey}(x_i, Y_j)$ (algorithm 24). Finally, all trustees publish to the server all encrypted *partial secret shares of the decryption key*.

By encrypting the partial secret shares with each trustee's public keys, we ensure that only that specific trustee can read his *partial secret shares of the decryption key*. This procedure is a slight deviation from [8], which we introduced to simulate a private communication channel between trustees.

Finally, each trustee $\mathcal{T}_i \in \boldsymbol{\mathcal{T}}$ downloads from the server their encrypted *partial secret shares* $c_{j,i}$, with $j \in \{1, ..., n\}$ and decrypts them $s_{j,i} \leftarrow \mathsf{SymDec}(k_{i,j}, c_{j,i})$ (algorithm 23), where $k_{i,j} \leftarrow \mathsf{DerSymKey}(x_i, Y_j)$ (algorithm 24). Recall form appendix B.7.1 that $k_{i,j} = k_{j,i}$ as $\mathsf{DerSymKey}(x_i, Y_j) = \mathsf{DerSymKey}(x_j, Y_i)$, when $Y_i = [x_i]G$ and $Y_j = [x_j]G$.

Then, each trustee $\mathcal{T}_i \in \boldsymbol{\mathcal{T}}$ validates that the partial secret shares generated by all the other trustees are consistent with their respective polynomial coefficients $[s_{j,i}]G = Y_j + \sum_{k=1}^{t-1} [i^k] P_{j,k}$, with $j \in \{1, ..., n\}$. If all *partial secret shares* validate, then trustee $\mathcal{T}_i$ computes his *secret share of the decryption key* by $sx_i \leftarrow \sum_{j=1}^{n} s_{j,i}$ and stores it privately until needed for decryption. At the end of the *threshold ceremony*, for each trustee $\mathcal{T}_i \in \boldsymbol{\mathcal{T}}$, the *public share of the*

decryption key ($sY_i = [sx_i]G$) is publicly computable by the following:

$$sY_i \leftarrow \sum_{j=1}^{n}(Y_j + \sum_{k=1}^{t-1}[i^k]P_{j,k}).$$

The encryption algorithm of the threshold cryptosystem is identical to the algorithm described in appendix B.4.1: $e = (R, C) \leftarrow \mathsf{Enc}(Y, M; r)$.

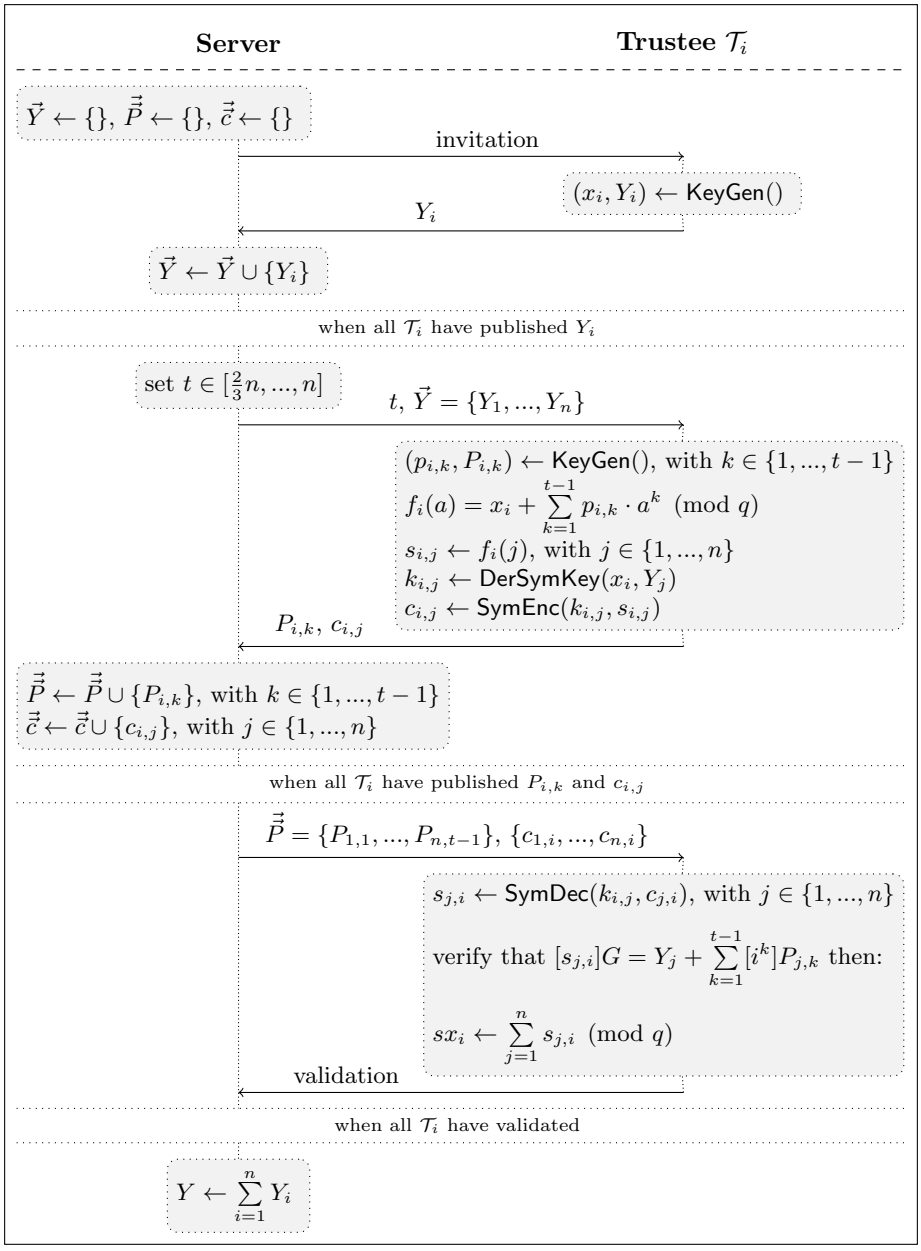


Figure 12: Threshold ceremony

The decryption protocol of the threshold cryptosystem is inspired by paper [4]. At least t trustees are needed to collaborate in the protocol described in figure 13 to extract the message M of a cryptogram $e = (R, C)$. We define $\tau \subset \{1, ..., n\}$ as the subset of trustees participating in the decryption protocol, with $|\tau| \geq t$.

Each trustee $\mathcal{T}_i$, with $i \in \tau$, computes a partial decryption $S_i \leftarrow [sx_i]R$ and sends it to the server, where sx_i is trustee's share of the decryption key. The trustee also publishes a proof of correct decryption in the form of a non-interactive discrete logarithm zero-knowledge proof $PK \leftarrow \mathsf{DLProve}(sx_i, \{G, R\})$ (algorithm 14).

When receiving a partial decryption from a trustee $\mathcal{T}_i$, the server accepts it if the proof of correct decryption validates by $\mathsf{DLVer}(PK, \{G, R\}, \{sY_i, S_i\})$ (algorithm 15), where sY_i is trustee's *public share of the decryption key*. After it received valid, partial decryptions from all trustees $\mathcal{T}_i$, with $i \in \tau$, the server aggregates all partial decryptions together to finalize the decryption and to output the message M. The aggregation process from [4] is described as follows:

Basically, $M = C - [x]R$, where x is the main decryption key that nobody has. A possible way of computing $[x]R$ is by calculating the *Lagrange Interpolation Polynomial* where each term is a partial decryption S_i received from a trustee $\mathcal{T}_i$ that needs to be multiplied by the *Lagrange Interpolation Polynomial coefficient* which is $\lambda(i) = \prod_{j \in \tau, j \neq i} \frac{-j}{i-j} \pmod q$. Formally, $M \leftarrow C - \sum_{i \in \tau} [\lambda(i)] S_i$, with $|\tau| \geq t$. Note that the *Lagrange Interpolation Polynomial* can be computed only when the number of terms is at least the degree of the polynomial, i.e., $|\tau| \geq t$.

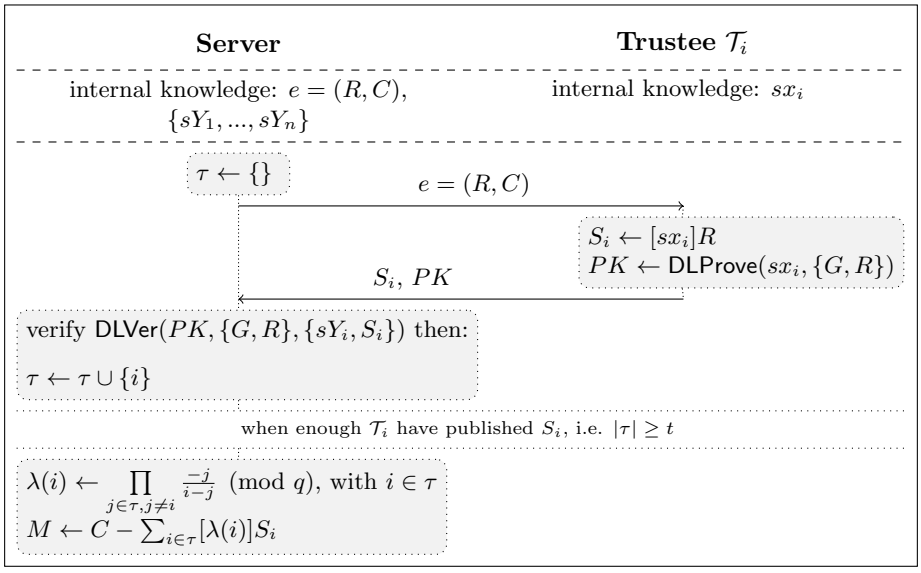


Figure 13: Threshold decryption

B.6 Symmetric encryption

A symmetric encryption scheme (SymEnc, SymDec) exists in case the message to be encrypted is not an elliptic curve point but, instead, an arbitrary length message $m \in \mathbb{B}^*$, e.g., a text message. A difference from Elgamal cryptography is that both encryption and decryption are done based on the same key that needs to be known by both parties (i.e., the sender and the receiver).

We use standard AES encryption algorithms with 256-bit keys in *Galois Counter Mode* as presented in [10]. Therefore, we define algorithm $\mathsf{SymEnc}(k, m)$ (algorithm 22) that encrypts message m with the key k and returns the tuple $e = (iv, t, c)$, where iv is the initialization vector, t is the authentication tag and c is the ciphertext. We also define algorithm $\mathsf{SymDec}(k, e)$ (algorithm 23) that returns plain text m as the decryption of ciphertext c with key k. Note that m is returned only if the authentication tag t validates. Both algorithms use standard implementation, therefore the inner workings are not described.

Algorithm 22: $\mathsf{SymEnc}(k, m) \rightarrow e = (iv, t, c)$

Data: The symmetric key $k \in \mathbb{B}^{256}$
The message $m \in \mathbb{B}^*$
Result: The initialization vector $iv \in \mathbb{B}^{96}$
The authentication tag $t \in \mathbb{B}^{128}$
The ciphertext $c \in \mathbb{B}^*$

Algorithm 23: $\mathsf{SymDec}(k, e) \rightarrow m$

Data: The symmetric key $k \in \mathbb{B}^{256}$
The encryption $e = (iv, t, c)$
The initialization vector $iv \in \mathbb{B}^{96}$
The authentication tag $t \in \mathbb{B}^{128}$
The ciphertext $c \in \mathbb{B}^*$
Result: The plain text $m \in \mathbb{B}^*$ or failure

B.7 Key derivation

Key derivation functions are algorithms that convert one source of randomness and secrecy (such as private keys or passwords) into different formats that can be used in different applications.

B.7.1 Diffie Hellman key derivation function

The strategy to convert from a private-public key infrastructure into a symmetric key is to use a *key encapsulation method* based on *Diffie Hellman Key Exchange*.

For two entities that have a private-public key infrastructure in place (i.e., entity 1 has key pair (x_1, Y_1) and entity 2 has key pair (x_2, Y_2), where $Y_1 = [x_1]G$ and $Y_2 = [x_2]G$) and that know each other (i.e., entity 1 knows Y_2 and entity 2 knows Y_1), they can both derive symmetric key k by running $\mathsf{DerSymKey}(x_1, Y_2)$ as entity 1 and $\mathsf{DerSymKey}(x_2, Y_1)$ as entity 2.

The algorithm performs a Diffie Hellman key exchange to reach a shared secret $S \leftarrow [x_1]Y_2 = [x_2]Y_1 = [x_1 + x_2]G$. The resulting value is used as the keying material of a hash-based key derivation function HKDF (algorithm 25) to convert it into a uniform key k. Particularly, no salt and info arguments are used (i.e., salt and info are set to $\varnothing$), therefore only one symmetric key can be derived from two particular key pairs (x_1, Y_1) and (x_2, Y_2).

The implementation of the key derivation function algorithm 25 is described in [11], therefore omitted here.

Algorithm 24: $\mathsf{DerSymKey}(x, Y)$

Data: A private key $x \in \mathbb{Z}_q$
A public key $Y \in \mathbb{P}$
$salt \leftarrow \varnothing$
$info \leftarrow \varnothing$
$length \leftarrow 256$
$S \leftarrow [x]Y$
$k \leftarrow \mathsf{HKDF}(S, salt, info, length)$
return k // $k \in \mathbb{B}^{256}$

Algorithm 25: $\mathsf{HKDF}(ik, s, i, \ell) \rightarrow k$

Data: The input keying material $ik \in \mathbb{B}^*$
The salt $s \in \mathbb{B}^*$
The info $i \in \mathbb{B}^*$
The output key length $\ell \in \mathbb{N}$
Result: The derived key $k \in \mathbb{B}^\ell$

B.7.2 Password-based key derivation function

PBKDF2 (algorithm 27) is a standard algorithm, described in [12], that converts passwords (arbitrary text) into keys that can be used in a cryptographic context. The algorithm takes as arguments a pseudorandom function, a password, a salt, an iteration count, and the desired length of the output key in bytes. The internal operations of the function are omitted in this paper.

We use PBKDF2 as a building block for $\mathsf{Pass2Key}(m)$ (algorithm 26) that converts password m into a key pair (x, Y). This can be seen as an alternative algorithm to $\mathsf{KeyGen}()$ (algorithm 16) that can be used to get a deterministic key pair based on some random seed.

Particularly, no salt is used (i.e., salt is set to $\varnothing$), therefore, only one key pair can be derived from password m. The amount of iterations is set to 600.000, according to recommendations from [13]. The password is concatenated with a counter that gets incremented until the output of the key derivation function can be interpreted as a correct private key (i.e., the output bytes are decoded as integer x, then checked whether $x \in \mathbb{Z}_q$).

Algorithm 26: $\mathsf{Pass2Key}(m)$

```
Data: A text m ∈ 𝔹*
salt ← ∅
iterations ← 600.000
ℓ ← ByteLengthOf(q)                                   // algorithm 11
i ← 0
repeat
    x ← PBKDF2(ℋ, m||i, salt, iterations, ℓ)
    if x ≥ q then
        i ← i + 1
        continue
    end
until x < q
Y ← [x]G
return (x, Y)                                         // (x, Y) ∈ ℤ_q × ℙ
```

Algorithm 27: $\mathsf{PBKDF2}(\mathcal{H}, p, s, i, \ell) \to k$

Data: The hashing function $\mathcal{H}$
The password $p \in \mathbb{B}^*$
The salt $s \in \mathbb{B}^*$
The iteration count $i \in \mathbb{N}$
The output key length $\ell \in \mathbb{N}$
Result: The derived key $k \in \mathbb{B}^\ell$

B.8 Schnorr digital signature

The *Schnorr digital signature scheme*, introduced in [14], consists of a triple of algorithms (KeyGen, Sign, SigVer), which are based on elliptic curve cryptographic primitive. A Schnorr key pair is a tuple $(x, Y) \leftarrow \mathsf{KeyGen}()$ (algorithm 16), where x is the random, private signing key and Y is the corresponding public signature verification key.

Only the owner of the private signing key is able to generate a valid signature $\sigma = (c, s) \leftarrow \mathsf{Sign}(x, m)$, on an arbitrary message $m \in \mathbb{B}^*$. To generate a signature, the signer follows algorithm 28. Given a signature σ on a message m, anybody in possession of the public verification key Y can verify the validity of

the signature $b \leftarrow \mathsf{SigVer}(Y, \sigma; m)$, with $b \in \mathbb{B}$ which represents *true* or $false$. The signature verification algorithm is described in algorithm 29.

Algorithm 28: $\mathsf{Sign}(x, m)$

Data: The signing key $x \in \mathbb{Z}_q$
The message to be signed $m \in \mathbb{B}^*$
$r \in_{\mathrm{R}} \mathbb{Z}_q$
$K \leftarrow [r]G$
$c \leftarrow \mathcal{H}(K||m)$
$s \leftarrow r - c \cdot x \pmod q$
$\sigma \leftarrow (c, s)$
return σ // $\sigma \in \mathbb{Z}_q \times \mathbb{Z}_q$

Algorithm 29: $\mathsf{SigVer}(Y, \sigma; m)$

Data: The signature verification key $Y \in \mathbb{P}$
The signature $\sigma = (c, s) \in \mathbb{Z}_q \times \mathbb{Z}_q$
The signed message $m \in \mathbb{B}^*$
$K \leftarrow [s]G + [c]Y$
if $c = \mathcal{H}(K||m)$ **then**
  $b \leftarrow 1$ // signature is valid
else
  $b \leftarrow 0$ // signature is invalid
end
return b // $b \in \mathbb{B}$

B.9 Pedersen commitment

A *commitment scheme* consists of a tuple of algorithms (Com, ComVer) that enables a writer to commit to a specific message m while keeping it secret. At a later point, if appropriate, the writer can open the commitment and reveal the committed message m. The *Pedersen Commitment Scheme* is a randomized commitment scheme introduced in [15]. Later, the scheme has been updated in [16] to enable commitment computation on a list of messages $\vec{m} = \{m_1, ..., m_n\}$, where each $m_i \in \mathbb{Z}_q$, with $i \in \{1, ..., n\}$.

A prerequisite part of the commitment scheme is the existence of multiple generators (one for each message in the list $\vec{m}$) in the subgroup such that the discrete logarithm amongst any two of them is unknown. To support that, we define the algorithm $\mathsf{BaseGen}$ that outputs a new generator H such that the value x is unknown where $H = [x]G$.

In order to commit to messages $\vec{m} = \{m_1, ..., m_n\}$ a writer computes the commitment $C \leftarrow \mathsf{Com}(\vec{m}; r)$ (algorithm 31), where $r \in_{\mathrm{R}} \mathbb{Z}_q$ is a randomizer. The algorithm internally computes a list of generators $\vec{G} = \{G_1, ..., G_n\}$ where each $G_i \leftarrow \mathsf{BaseGen}(i)$ (algorithm 30), with $i \in \{1, ..., n\}$.

To reveal messages $\vec{m}$, the writer needs to publish values $\vec{m}$ and r. A verifier is convinced that the commitment C opens to messages $\vec{m}$ by running $b \leftarrow \mathsf{ComVer}(C, \vec{m}; r)$ (algorithm 32).

Algorithm 30: $\mathsf{BaseGen}(i)$

Data: An index $i \in \mathbb{N}$
$j \leftarrow 0$
repeat
  $x \leftarrow \mathcal{H}(G||i||j)$
  $\vec{b} \leftarrow 02||$ byte representation of $\mathcal{H}(G||i||j)$
  $H \leftarrow \mathsf{DecodePoint}(\vec{b})$ `// algorithm 9`
  if *H is invalid* **then**
    $j \leftarrow j + 1$
    continue
  end
until *H is valid*
return H `//` $H \in \mathbb{P}$

Algorithm 31: $\mathsf{Com}(\vec{m}; r)$

Data: The list of messages $\vec{m} = \{m_1, ..., m_n\} \in \mathbb{Z}_q^n$
  The randomizer $r \in \mathbb{Z}_q$
for $i \leftarrow 1$ **to** n **by** 1 **do**
  $G_i \leftarrow \mathsf{BaseGen}(i)$ `// algorithm 30`
end
$C \leftarrow [r]G + \sum_{i=1}^{n} [m_i]G_i$
return C `//` $C \in \mathbb{P}$

B.10 Groth argument of shuffle

A *cryptographic shuffle* (or mixing) is a process that, given as input a list of cryptograms, outputs another list of cryptograms such that each cryptogram from the input list is re-encrypted and permuted in a random new order, forming the output list. This can be further extended to *mixing* a matrix of cryptograms, where all cryptograms are re-encrypted, and rows are permuted in a new order. Formally, given a matrix of cryptograms $\vec{\vec{e}} = \{e_{1,1}, ..., e_{n,\ell}\} \in \mathbb{E}^{n,\ell}$, with each $e_{i,j} = (R_{i,j}, C_{i,j})$, $i \in \{1, ..., n\}$ and $j \in \{1, ..., \ell\}$, a matrix of randomizers $\vec{\vec{r}} = \{r_{1,1}, ..., r_{n,\ell}\} \in \mathbb{Z}_q^{n \times \ell}$ and a permutation $\psi : \{1, ..., n\} \leftarrow \{1, ..., n\}$ from the set Ψ_n of all permutations of n elements, the shuffle algorithm outputs the matrix $\vec{\vec{e}}' = \{e'_{1,1}, ..., e'_{n,\ell}\} \leftarrow \mathsf{Shuffle}(Y, \vec{\vec{e}}; \vec{\vec{r}}, \psi)$ (algorithm 33) where each $e'_{i,j} = (R'_{i,j}, C'_{i,j}) \leftarrow \mathsf{ReEnc}(Y, e_{k,j}; r_{i,j})$ (algorithm 20) for $k = \psi(i)$.

The interesting aspect of mixing is how to prove in zero-knowledge that the shuffling calculations were done correctly and that no content of the cryptograms

Algorithm 32: $\mathsf{ComVer}(C, \vec{m}; r)$

Data: The commitment $C \in \mathbb{P}$
The list of messages $\vec{m} = \{m_1, ..., m_n\} \in \mathbb{Z}_q^n$
The randomizer $r \in \mathbb{Z}_q$
for $i \leftarrow 1$ **to** n **by** 1 **do**
  $G_i \leftarrow \mathsf{BaseGen}(i)$ // algorithm 30
end
if $C = [r]G + \sum_{i=1}^{n} [m_i]G_i$ **then**
  $b \leftarrow 1$ // commitment is valid
else
  $b \leftarrow 0$ // commitment is invalid
end
return b // $b \in \mathbb{B}$

Algorithm 33: $\mathsf{Shuffle}(Y, \vec{\vec{e}}, \vec{\vec{r}}, \psi)$

Data: The encryption key $Y \in \mathbb{P}$
The matrix of initial cryptograms $\vec{\vec{e}} = \{e_{1,1}, ..., e_{n,\ell}\} \in \mathbb{E}^{n \times \ell}$
The matrix of randomizers $\vec{\vec{r}} = \{r_{1,1}, ..., r_{n,\ell}\} \in \mathbb{Z}_q^{n \times \ell}$
The permutation $\psi \in \Psi_n$
for $i \leftarrow 1$ **to** n **by** 1 **do**
  for $j \leftarrow 1$ **to** ℓ **by** 1 **do**
    $e'_{i,j} \leftarrow \mathsf{ReEnc}(Y, e_{\psi(i),j}; r_{i,j})$ // algorithm 20
  end
end
$\vec{\vec{e}}' \leftarrow \{e'_{1,1}, ..., e'_{n,\ell}\}$
return $\vec{\vec{e}}'$ // $\vec{\vec{e}}' \in \mathbb{E}^{n \times \ell}$

has been changed. Our mixing proof is based on an algorithm presented by Jens Groth in [17]. The proof uses as a building block an *Argument for Shuffle of Known Contents*, which is based on proving the knowledge of opening a commitment to a permutation of a set of known messages. The strategy of Groth's algorithm is to reduce the problem of proving that $\vec{\vec{e}}'$ is the shuffled list of re-encrypted cryptograms $\vec{\vec{e}}$ to the problem of proving the shuffling of some known messages where the same permutation ψ is applied.

The protocol for the Argument of Shuffle of Known Contents is presented in figure 14. During this protocol, the *Prover* convinces the *Verifier* that C is a commitment to a set of known messages $\vec{m} = \{m_1, ..., m_n\}$ that are shuffled by a secret permutation ψ. Note that, in this protocol, the *Prover* does not reveal the permutation ψ.

The protocol for proving the correctness of a shuffle is illustrated in figure 15. The *Prover* convinces the *Verifier* that the matrix of mixed cryptograms $\vec{\vec{e}}' = \{e'_{1,1}, ..., e'_{n,\ell}\}$ is equivalent to the initial cryptograms matrix $\vec{\vec{e}} = \{e_{1,1}, ..., e_{n,\ell}\}$,

where each cryptogram is re-encrypted, and rows of the initial matrix are shuffled amongst each other. Note that, during mixing, the integrity of each row is preserved, i.e., all columns of the matrix are shuffled by the same permutation. The protocol uses, as a building block, the protocol for the Argument of Shuffle of Known Contents, presented in figure 14.

Note that, in the description of the protocols, we abuse notation and define $\sum_{i=1}^{n} e_i = \mathsf{HomAdd}(e_1; \mathsf{HomAdd}(e_2; ...\mathsf{HomAdd}(e_{n-1}; e_n)...))$ (algorithm 19) as the homomorphic addition of multiple cryptograms, with each $e_i \in \mathbb{E}$.

Jens Groth suggests in [17] that the protocols can be turned into non-interactive algorithms by using the *Fiat-Shamir heuristic* strategy [18] to compute the random value x, e, $\vec{t}$ and λ by applying a hash function to the transcript of the protocol. Therefore, we transform each protocol into a set of two algorithms (one for generating a universally verifiable non-interactive proof and another for verifying it).

Explicitly, to prove the correct mixing of cryptograms $\vec{\vec{e}} = \{e_{1,1}, ..., e_{n,\ell}\}$ by randomizers $\vec{\vec{r}} = \{r_{1,1}, ..., r_{n,\ell}\}$ and permutation ψ into the mixed cryptograms $\vec{\vec{e}}' = \{e'_{1,1}, ..., e'_{n,\ell}\}$, the *Prover* generates the tuple (proof of mixing and argument of shuffle) $(PM, AS) \leftarrow \mathsf{MixProve}(\psi, Y, \vec{\vec{r}}, \vec{\vec{e}}, \vec{\vec{e}}')$ (algorithm 36), where Y is the encryption key. Anybody can universally verify a proof of mixing by $\mathsf{MixVer}(PM, AS, Y, \vec{\vec{e}}, \vec{\vec{e}}')$ (algorithm 37).

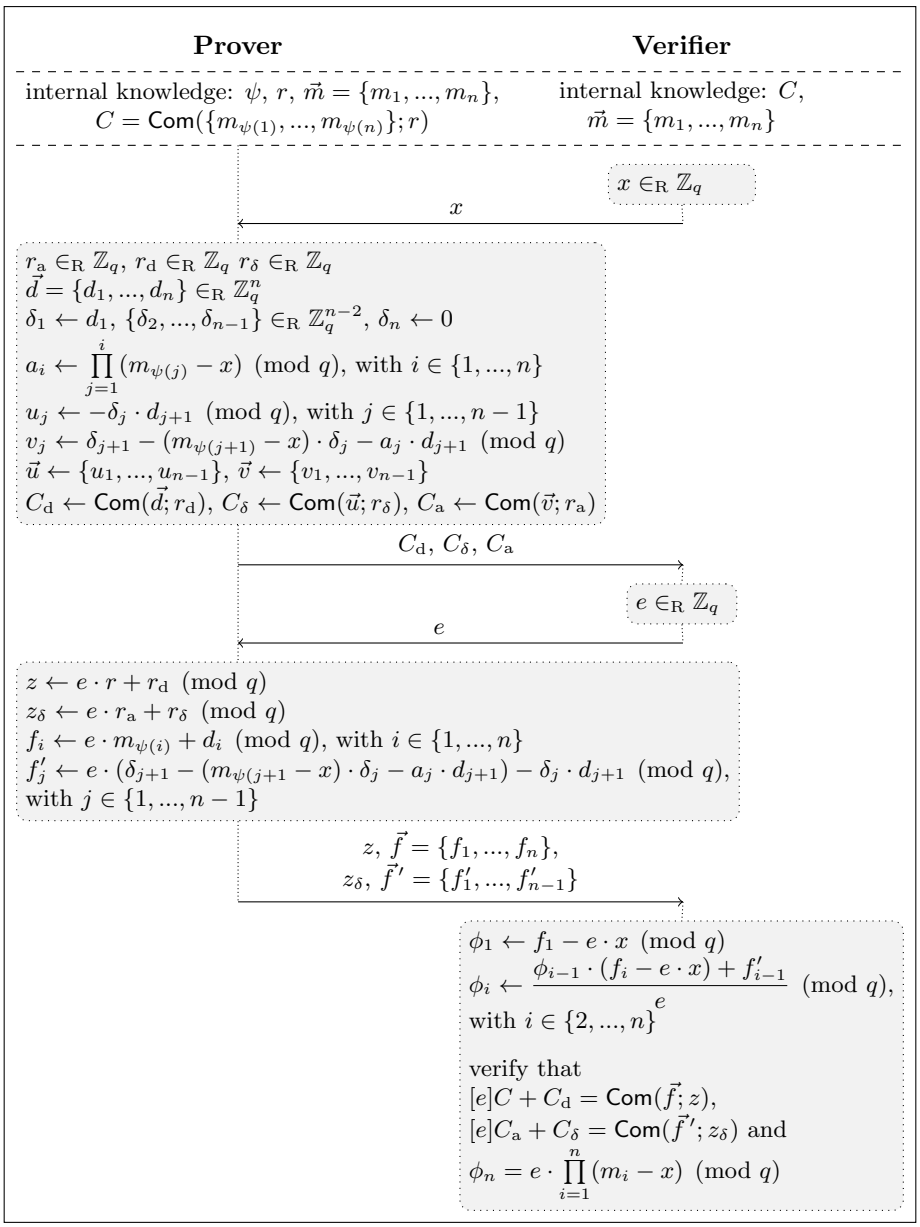


Figure 14: Argument of Shuffle of Known Contents

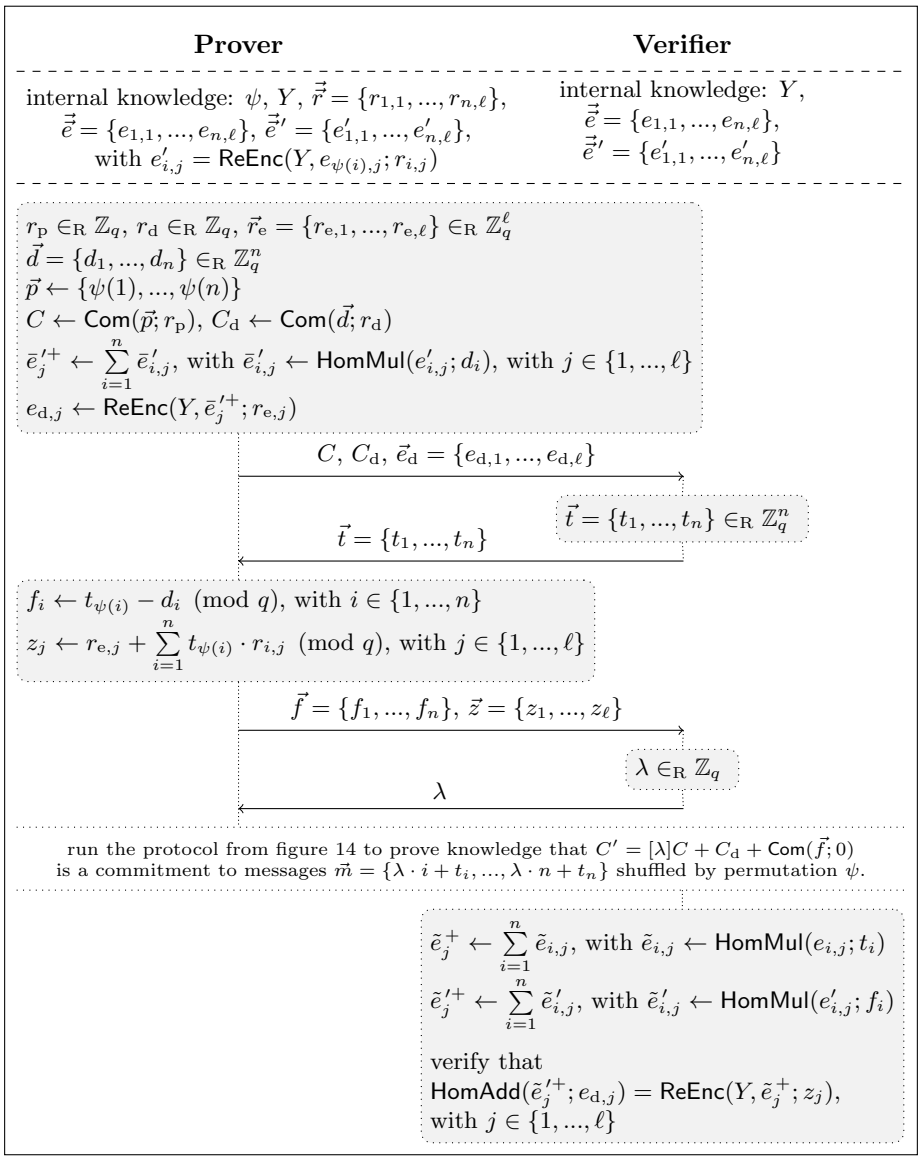


Figure 15: Argument of Shuffle of Cryptograms

Algorithm 34: $\mathsf{ASKCProve}(\psi; r; \vec{m}; C)$

Data: The permutation $\psi \in \Psi_n$
The randomizer $r \in \mathbb{Z}_q$
The list of known messages $\vec{m} = \{m_1, ..., m_n\} \in \mathbb{Z}_q^n$
The public commitment $C \in \mathbb{P}$

$x \leftarrow \mathcal{H}(\vec{m}||C)$
$r_\mathrm{a} \in_\mathrm{R} \mathbb{Z}_q$, $r_\mathrm{d} \in_\mathrm{R} \mathbb{Z}_q$ $r_\delta \in_\mathrm{R} \mathbb{Z}_q$
for $i \leftarrow 1$ **to** n **by** 1 **do**
  $d_i \in_\mathrm{R} \mathbb{Z}_q$
  $a_i \leftarrow \prod_{j=1}^{i} (m_{\psi(j)} - x) \pmod q$
end
$\delta_1 \leftarrow d_1$, $\delta_n \leftarrow 0$
for $i \leftarrow 2$ **to** $n-1$ **by** 1 **do**
  $\delta_i \in_\mathrm{R} \mathbb{Z}_q$
end
for $i \leftarrow 1$ **to** $n-1$ **by** 1 **do**
  $u_i \leftarrow -\delta_i \cdot d_{i+1} \pmod q$
  $v_i \leftarrow \delta_{i+1} - (m_{\psi(i+1)} - x) \cdot \delta_i - a_i \cdot d_{i+1} \pmod q$
end
$\vec{d} \leftarrow \{d_1, ..., d_n\}$, $\vec{u} \leftarrow \{u_1, ..., u_{n-1}\}$, $\vec{v} \leftarrow \{v_1, ..., v_{n-1}\}$
$C_\mathrm{d} \leftarrow \mathsf{Com}(\vec{d}; r_\mathrm{d})$, $C_\delta \leftarrow \mathsf{Com}(\vec{u}; r_\delta)$, $C_\mathrm{a} \leftarrow \mathsf{Com}(\vec{v}; r_\mathrm{a})$ `// algorithm 31`
$e \leftarrow \mathcal{H}(\vec{m}||C||C_\mathrm{d}||C_\delta||C_\mathrm{a})$
$z \leftarrow e \cdot r + r_\mathrm{d} \pmod q$, $z_\delta \leftarrow e \cdot r_\mathrm{a} + r_\delta \pmod q$
for $i \leftarrow 1$ **to** n **by** 1 **do**
  $f_i \leftarrow e \cdot m_{\psi(i)} + d_i \pmod q$
end
for $i \leftarrow 1$ **to** $n-1$ **by** 1 **do**
  $f'_i \leftarrow e \cdot (\delta_{i+1} - (m_{\psi(i+1} - x) \cdot \delta_i - a_i \cdot d_{i+1}) - \delta_i \cdot d_{i+1} \pmod q$
end
$\vec{f} \leftarrow \{f_1, ..., f_n\}$, $\vec{f}' \leftarrow \{f'_1, ..., f'_{n-1}\}$
$AS \leftarrow (C_\mathrm{d}, C_\delta, C_\mathrm{a}, x, e, z, z_\delta, \vec{f}, \vec{f}')$
return AS `//` $AS \in \mathbb{P}^3 \times \mathbb{Z}_q^4 \times \mathbb{Z}_q^n \times \mathbb{Z}_q^{n-1}$

Algorithm 35: $\mathsf{ASKCVer}(AS; \vec{m}; C)$

Data: The argument $AS = (C_{\mathrm{d}}, C_\delta, C_{\mathrm{a}}, x, e, z, z_\delta, \vec{f}, \vec{f}') \in \mathbb{P}^3 \times \mathbb{Z}_q^4 \times \mathbb{Z}_q^n \times \mathbb{Z}_q^{n-1}$
The list of known messages $\vec{m} = \{m_1, ..., m_n\} \in \mathbb{Z}_q^n$
The public commitment $C \in \mathbb{P}$

$\phi_1 \leftarrow f_1 - e \cdot x \pmod q$
for $i \leftarrow 2$ **to** n **by** 1 **do**
  $\phi_i \leftarrow \dfrac{\phi_{i-1} \cdot (f_i - e \cdot x) + f'_{i-1}}{e} \pmod q$
end
if $x = \mathcal{H}(\vec{m}||C)$ **and** $e = \mathcal{H}(\vec{m}||C||C_{\mathrm{d}}||C_\delta||C_{\mathrm{a}})$
and $\phi_n = e \cdot \prod_{i=1}^{n} (m_i - x) \pmod q$
and $[e]C + C_{\mathrm{d}} = \mathsf{Com}(\vec{f}; z)$ **and** $[e]C_{\mathrm{a}} + C_\delta = \mathsf{Com}(\vec{f}'; z_\delta)$ `// algorithm 31`
then
  $b \leftarrow 1$ `// argument is valid`
else
  $b \leftarrow 0$ `// argument is invalid`
end
return b `// b ∈ 𝔹`

Algorithm 36: $\mathsf{MixProve}(\psi, Y, \vec{\vec{r}}, \vec{\vec{e}}, \vec{\vec{e}}')$

Data: The permutation $\psi \in \Psi_n$
The encryption key $Y \in \mathbb{P}$
The matrix of randomizers $\vec{\vec{r}} = \{r_{1,1}, ..., r_{n,\ell}\} \in \mathbb{Z}_q^{n \times \ell}$
The matrix of initial cryptograms $\vec{\vec{e}} = \{e_{1,1}, ..., e_{n,\ell}\} \in \mathbb{E}^{n \times \ell}$
The matrix of mixed cryptograms $\vec{\vec{e}}' = \{e'_{1,1}, ..., e'_{n,\ell}\} \in \mathbb{E}^{n \times \ell}$, with $e'_{i,j} = \mathsf{ReEnc}(Y, e_{\psi(i),j}; r_{i,j})$

$r_\mathrm{p} \in_\mathrm{R} \mathbb{Z}_q$, $r_\mathrm{d} \in_\mathrm{R} \mathbb{Z}_q$
for $i \leftarrow 1$ **to** n **by** 1 **do**
  $d_i \in_\mathrm{R} \mathbb{Z}_q$, $p_i \leftarrow \psi(i)$
  for $j \leftarrow 1$ **to** ℓ **by** 1 **do**
    $\bar{e}'_{i,j} \leftarrow \mathsf{HomMul}(e'_{i,j}; d_i)$ `// algorithm 21`
  end
end
$\vec{d} \leftarrow \{d_1, ..., d_n\}$, $\vec{p} \leftarrow \{p_1, ..., p_n\}$
$C \leftarrow \mathsf{Com}(\vec{p}; r_\mathrm{p})$, $C_\mathrm{d} \leftarrow \mathsf{Com}(\vec{d}; r_\mathrm{d})$ `// algorithm 31`
for $j \leftarrow 1$ **to** ℓ **by** 1 **do**
  $r_{\mathrm{e},j} \in_\mathrm{R} \mathbb{Z}_q$
  $e_{\mathrm{d},j} \leftarrow \mathsf{ReEnc}(Y, \bar{e}'^{+}_j; r_{\mathrm{e},j})$, with $\bar{e}'^{+}_j \leftarrow \sum_{i=1}^{n} \bar{e}'_{i,j}$ `// algorithm 20`
end
$\vec{e}_\mathrm{d} \leftarrow \{e_{\mathrm{d},1}, ..., e_{\mathrm{d},\ell}\}$
for $i \leftarrow 1$ **to** n **by** 1 **do**
  $t_i \leftarrow \mathcal{H}(\vec{\vec{e}}||\vec{\vec{e}}'||C||C_\mathrm{d}||\vec{e}_\mathrm{d}||i)$
end
for $i \leftarrow 1$ **to** n **by** 1 **do**
  $f_i \leftarrow t_{\psi(i)} - d_i \pmod q$
end
$\vec{f} \leftarrow \{f_1, ..., f_n\}$
for $j \leftarrow 1$ **to** ℓ **by** 1 **do**
  $z_j \leftarrow r_{\mathrm{e},j} + \sum_{i=1}^{n} t_{\psi(i)} \cdot r_{i,j} \pmod q$
end
$\vec{z} \leftarrow \{z_1, ..., z_\ell\}$
$\lambda \leftarrow \mathcal{H}(\vec{\vec{e}}||\vec{\vec{e}}'||C||C_\mathrm{d}||\vec{e}_\mathrm{d}||\vec{f}||\vec{z})$
for $i \leftarrow 1$ **to** n **by** 1 **do**
  $m'_i \leftarrow \lambda \cdot \psi(i) + t_{\psi(i)}$
end
$\vec{m}' \leftarrow \{m'_1, ..., m'_n\}$, $r' \leftarrow \lambda + r_\mathrm{d} \pmod q$
$C' \leftarrow \mathsf{Com}(\vec{m}'; r')$ `// algorithm 31`
$AS \leftarrow \mathsf{ASKCProve}(\psi; r'; \vec{m}'; C')$ `// algorithm 34`
$PM \leftarrow (C, C_\mathrm{d}, \vec{e}_\mathrm{d}, \vec{t}, \vec{f}, \vec{z}, \lambda)$
return (PM, AS) `//` $PM \in \mathbb{P}^2 \times \mathbb{E}^\ell \times \mathbb{Z}_q^{2n} \times \mathbb{Z}_q^\ell \times \mathbb{Z}_q$
`//` $AS \in \mathbb{P}^3 \times \mathbb{Z}_q^4 \times \mathbb{Z}_q^n \times \mathbb{Z}_q^{n-1}$

Algorithm 37: $\mathsf{MixVer}(PM, AS, Y, \vec{\vec{e}}, \vec{\vec{e}}')$

Data: The proof $PM = (C, C_{\mathrm{d}}, \vec{e}_{\mathrm{d}}, \vec{t}, \vec{f}, \vec{z}, \lambda) \in \mathbb{P}^2 \times \mathbb{E}^{\ell} \times \mathbb{Z}_q^{2n} \times \mathbb{Z}_q^{\ell} \times \mathbb{Z}_q$
The argument of shuffle $AS \in \times\mathbb{P}^3 \times \mathbb{Z}_q^4 \times \mathbb{Z}_q^n \times \mathbb{Z}_q^{n-1}$
The encryption key $Y \in \mathbb{P}$
The matrix of initial cryptograms $\vec{\vec{e}} = \{e_{1,1}, ..., e_{n,\ell}\} \in \mathbb{E}^{n \times \ell}$
The matrix of mixed cryptograms $\vec{\vec{e}}' = \{e'_{1,1}, ..., e'_{n,\ell}\} \in \mathbb{E}^{n \times \ell}$

for $i \leftarrow 1$ **to** n **by** 1 **do**
  for $j \leftarrow 1$ **to** ℓ **by** 1 **do**
    $\tilde{e}_{i,j} \leftarrow R\mathsf{HomMul}(e_{i,j}; t_i)$ // algorithm 21
    $\tilde{e}'_{i,j} \leftarrow \mathsf{HomMul}(e'_{i,j}; f_i)$ // algorithm 21
  end
  $m_i \leftarrow \lambda \cdot i + t_i \pmod q$
end
for $j \leftarrow 1$ **to** ℓ **by** 1 **do**
  $\tilde{e}_j^{+} \leftarrow \sum_{i=1}^{n} \tilde{e}_{i,j}$
  $\tilde{e}_j'^{+} \leftarrow \sum_{i=1}^{n} \tilde{e}'_{i,j}$
end
$C' \leftarrow [\lambda]C + C_{\mathrm{d}} + \mathsf{Com}(\vec{f}; 0)$ // algorithm 31
$\vec{m} \leftarrow \{m_1, ..., m_n\}$
if $t_i = \mathcal{H}(\vec{\vec{e}}||\vec{\vec{e}}'||C||C_{\mathrm{d}}||\vec{e}_{\mathrm{d}}||i)$, *where* $i \in \{1, ..., n\}$
and $\lambda = \mathcal{H}(\vec{\vec{e}}||\vec{\vec{e}}'||C||C_{\mathrm{d}}||\vec{e}_{\mathrm{d}}||\vec{f}||\vec{z})$
and $\mathsf{HomAdd}(\tilde{e}_j'^{+}; e_{\mathrm{d},j}) = \mathsf{ReEnc}(Y, \tilde{e}_j^{+}; z_j)$, *where* $j \in \{1, ..., \ell\}$
and $\mathsf{ASKCVer}(AS; \vec{m}; C')$ // algorithms 19, 20 and 35
then
  $b \leftarrow 1$ // proof is valid
else
  $b \leftarrow 0$ // proof is invalid
end
return b // $b \in \mathbb{B}$