Electa

Assembly Voting

March 2023

Documentation of the cryptographic protocol

Version 2.0

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

1.1 State of solution

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.

Apart from the core features of the product described in the main sections, appendix C.1 and appendix C.2 present additional functionality required for the Mobile Voting Project [1] in the US.

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 mathematical descriptions of all algorithms used throughout the protocol.

The adversary model in section 5 provides a non-cryptographic description of the attack scenarios that the protocol protects against. This section could interest readers with a more general interest in security in online election systems.

1.3 Protocol scope and objectives

Some of the core features of the election protocol include: voters vote remotely, votes are encrypted, the system uses threshold cryptography, votes are cryptographically shuffled to ensure anonymity, all essential processes are verifiable, and auditing can be performed on all system components throughout the election event. The system cannot always prevent fraud or unauthorized access, but can detect it.

Multiple election types are supported, such as a referendum, candidate, multiple choice, or ranked elections. Multiple result types are also supported. The protocol has support for write-in votes. Additionally, the system provides continuous turnout statistics.

The scope of the protocol covers an entire election event, starting from election configuration, voter authorization, vote casting, tallying, and auditing. Cryptographic algorithms are crucial in terms of the security and auditing features of the system, but many non-cryptographic processes are also necessary to conduct a safe election. This document describes an online election system. Users, i.e., election officials and voters, access the system through a web browser or a native application on an internet-connected device such as a PC, laptop, tablet, smartphone, and so forth.

The overall objective of the document is to describe, claim and argue the achievement of the following requirements of our protocol, which are described in greater detail in section 2.6:

- Mobility
- Vote & go
- Transparency
- Multiple voting rounds
- Multiple election types
- Confirming selected options
- Support correcting mistakes
- Vote overwrites
- Individual verification
- Universal verification
- Full audit
- Eligibility
- Privacy
- Anonymity
- Integrity
- Ent-to-end verifiability
- Receipt-freeness.

1.4 Document outline

Section 2 lists all the key parts of the election system, including the stakeholders, system components, and the communication channels they use in the protocol. Then, it presents the implications of having a public bulletin board that collects all election data. Next, it describes the voter authentication modes that are supported. The section ends with a list of requirements the election system must fulfill.

Section 3 presents the cryptographic algorithms and processes that the election entities must follow in the life cycle of an election. The section is split into pre-election, election, and post-election processes. The section ends by listing the election properties and explaining how they are achieved.

Section 4 presents all the auditing processes. It describes who can perform each particular audit process, when it can happen, and what inputs are needed.

Section 5 presents the adversary model that the system is designed to handle. It lists all trust assumptions the system relies on, the threats arising from them, and how they are mitigated.

Appendix A lists the applied cryptographic algorithms and their mathematical principles.

Appendix B contains a comprehensive list of all items that appear on the public bulletin board. It defines the rules and structure for each item type.

Appendix C presents a list of additional optional features that are not considered as the core product. Each feature is described in terms of how it modifies the election protocol and how it impacts the election properties.

1.5 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 $\in_{\text{R}}$ to denote random assignment from a set $x \in_{\text{R}} \mathbb{Z}$,
- 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 entities seen throughout this documentation in addition to the requirements.

2.1 Stakeholders

This section describes the human actors that have a stake in an election. All stakeholders can be grouped into four categories.

2.1.1 Election official

Election officials use the election system to configure and run an election. Election officials own credentials that are used to access different election components, such as the election administration service. Election officials have an interest in the election running correctly, according to the configuration they set up.

2.1.2 Trustee

Trustees are a particular type of election officials. In addition to the election official role, trustees are responsible for preserving the secrecy of the voting data throughout the election. As described in the protocol, trustees are exposed to keys that they have to protect and keep secret. Trustees are crucial to the election protocol, as they actively participate in the result computation.

2.1.3 Voter

All the voting data is generated by a set of predefined voters. They own credentials that get them authorized to cast a digital ballot. Voters have an interest in verifying that their vote has been processed correctly and is included in the final tally.

2.1.4 Public auditor

Any person can be a public auditor of the election process, assuming they have access to the proper auditing tools that will perform all the cryptographic operations on behalf of the auditor. Auditors don't have an active role in the election process, and most of the time, the auditing will happen without the election system noticing.

2.2 System components

This section describes all parties that are involved in the election protocol. Each party represents a computer or simply an application that follows a particular protocol. Each party is accessed and controlled by one of the stakeholders or by an organization that hosts that specific application. All these parties can be categorized into the following nine types:

2.2.1 Election Administrator

There exists one administrator that is responsible for setting up the election event and making updates to the configurations. The election administrator $\mathcal{E}$ is a single service that all election officials use to set up an election event, configure it and keep it updated. The election administrator service is an online application hosted by an organization. Election officials can access the service based on some pre-established set of credentials.

The election administrator $\mathcal{E}$ owns a key pair $(x_{\mathcal{E}}, Y_{\mathcal{E}})$ used for signing the election configuration, and is responsible for privately storing its private key $x_{\mathcal{E}}$.

2.2.2 Trustee application

There is a set of trustees, each denoted as $\mathcal{T}_i$, with $i \in \{1, ..., n_{\text{t}}\}$, where n_{t} is the total number of trustees. Each trustee uses the trustee application to perform all cryptographic processes involved in the protocol. Trustees are responsible for preserving the privacy and the fairness of the election during the election phase by working together to build the election encryption key while safely storing their shares of the decryption key.

The trustee application will compute and deliver to its trustee $\mathcal{T}_i$, a key pair $(x_{\mathcal{T}_i}, Y_{\mathcal{T}_i})$ and a share of the election decryption key sx_i. The trustee is responsible for privately storing the keys until a result is computed. Trustees are responsible for destroying the keys after the election event has ended.

2.2.3 Voting application

There is a list of pre-defined eligible voters, each denoted $\mathcal{V}_i$, with $i \in \{1, ..., n_{\text{v}}\}$, where n_{v} is the total number of voters. The voting application is the software that voters use to perform all the cryptographic operations involved in the protocol. The voting application runs locally, on the voter's device, such as an application on the phone or a web application in the browser. The voting application requires an internet connection.

During the protocol, the voting application generates a key pair (x_i, Y_i) that represents the cryptographic identity of voter $\mathcal{V}_i$. Apart from the private key, the voting application learns all secrets that its voter inputs, e.g., the voter's credentials and the plain-text vote.

2.2.4 Credentials Authority

The Credentials Authority role is relevant only in the **credential-based** voter authentication mode, described in section 2.5.1.

There is a set of credentials authorities, each noted as $\mathcal{C}_i$, with $i \in \{1, ..., n_{\text{c}}\}$, where n_{c} is the total number of credentials authorities. A credentials authority is an institution consisting of both humans and software processes. Each credentials authority is responsible for generating and privately distributing voter credentials to the voters. It is recommended that each credentials authority use a different communication channel for distributing credentials (e.g., e-mail, post, SMS). Credentials Authorities must delete the voter credentials after they have been distributed.

2.2.5 Identity Provider

The Identity Provider role is relevant only in the **identity-based** voter authentication mode, described in section 2.5.2.

There is a set of Identity Providers $\mathcal{I}_i$, with $i \in \{1, ..., n_\mathrm{i}\}$, where n_i is the total number of identity providers. An Identity Provider is a third-party application responsible for authenticating a voter $\mathcal{V}$ during the election phase. Identity Providers must follow the OIDC protocol.

2.2.6 Voter Authorizer

The Voter Authorizer $\mathcal{A}$ is a service responsible for authorizing a Voter $\mathcal{V}$ after being authenticated. The voter authentication can be performed by providing the correct voter credentials or authenticating with all Identity Providers, depending on the voter authentication mode (described in section 2.5).

The Voter Authorizer $\mathcal{A}$ is responsible for preserving the election eligibility property by preventing non-eligible voters from voting. The Voter Authorizer $\mathcal{A}$ owns a key pair $(x_\mathcal{A}, Y_\mathcal{A})$ used for signing voter authorization and it is responsible for privately storing its private key $x_\mathcal{A}$.

2.2.7 Digital Ballot Box

The Digital Ballot Box $\mathcal{D}$ is the central communication unit, so all other parties push/pull data to/from it. It is a single service, publicly accessible via the internet. The Digital Ballot Box $\mathcal{D}$ has a bulletin board which contains all the public information about an election. The data which is published on the bulletin board is thoroughly described in section 2.4.2, but it can be summarized into the following categories:

- configuration data; This is set up during the pre-election phase (described in section 3.2), mostly generated by the Election Administrator $\mathcal{E}$.
- voting data; This is populated during the election phase (described in section 3.3) and is generated by the voting application in collaboration with the Digital Ballot Box $\mathcal{D}$.
- result data; This is collected during the post-election phase (described in section 3.4) and includes mixing and decryption files generated by Trustees and enable the result to be verifiable.

The Digital Ballot Box $\mathcal{D}$ owns a key pair $(x_\mathcal{D}, Y_\mathcal{D})$ used for signing data on the bulletin board, and it is responsible for privately storing its private key $x_\mathcal{D}$.

2.2.8 External Verifier

The External Verifier $\mathcal{X}$ is an auditing tool that voters use if they choose to perform the process of challenging a vote cryptogram (section 3.3.4). The External Verifier lets voters check that their vote has been correctly encrypted and stored on the bulletin board. During this process, the External Verifier $\mathcal{X}$ will generate a new key pair $(x_{\mathcal{X}}, Y_{\mathcal{X}})$ and it has to protect its private key $x_{\mathcal{X}}$.

2.2.9 Auditing tools

Two auditing tools are used by different actors to perform the auditing process. The election officials use the administrative auditing tool to run all the cryptographic operations involved in the administrative auditing process. This confirms to the election officials that the election system behaved according to their configuration.

Any public auditor uses public auditing tool to perform the public audit process. This verifies the integrity of all public data from the bulletin board.

2.3 Communication channels

The election protocol uses three types of communication channels to transfer data between two parties, i.e., a sender and a receiver. They are categorized as private, authentic, or public channels. Two relevant criteria differentiate the channel types, namely secrecy, and authenticity.

A secret communication channel implies that any outside observer cannot read the data being transferred. The communication channel provides a way to obfuscate the data. An authentic communication channel involves some mechanism that grants the receiver a confirmation that the data has been genuinely constructed by the sender.

The following sub-sections describe what criteria are provided by each of the communication channels. The type of channel being used during the election protocol depends on the cryptographic environment available at that step in the process and on the data being transferred.

2.3.1 Private channels

A private channel provides both secrecy and authenticity to the data being communicated. This type of channel is used when the data in transfer is confidential to the two actors communicating but also sensitive (i.e., any tampering with the data causes the protocol to break). A private channel prevents any outsider from reading any part of the data or modifying it. Usually, private channels are used where a cryptographic infrastructure has not been established yet.

The requirement of private channels is seen as a weakness as it introduces external security dependencies to achieve specific properties. In general, the election protocol was designed with the least need for private communication channels.

2.3.2 Authentic channels

An authentic channel provides only the authenticity property to the data that is being communicated. This channel type is used when the data in transfer is not secret but cannot be tampered with. Therefore, the data must contain proof that it genuinely comes from the sender. An authentic channel protects against a man-in-the-middle attack but allows that man in the middle to read all the traffic.

An example is when exchanging public keys. They are, as the name suggests, public, while they have to represent their owner authentically.

2.3.3 Public channels

A public channel does not provide secrecy or authenticity by itself. Instead, the data in transfer must have built-in mechanisms that ensure secrecy and authenticity. Examples of such mechanisms are encryption and digital signatures.

Obviously, we recommend taking measures to secure all communication channels in an election deployment. Though, theoretically speaking, not all of them need to be secure for the protocol to work.

2.4 Public bulletin board

All events happening during an election are published by the digital ballot box as items on a publicly available bulletin board. Each item from the board is owned (or written) by a relevant actor. Each item posted on the bulletin board describes a specific event, and it is uniquely identifiable by its *hash value* or *address*. The *address* of the last item on the board represents the *board hash value* at that specific point in time. All events are stored as an *append only list*, meaning no event can be removed or replaced, and each new event is appended at the end of the list. The structure of the bulletin board was inspired by [2].

The way we deviate from [2] is that to append a new item on the board, the writer needs to include the address of any existing item from the board as part of the new item, instead of referencing exactly the previous item. We call this reference the *parent* item. Finally, the address of the new item is computed by the digital ballot box $\mathcal{D}$ by hashing the content of the item (including the reference to the parent item) concatenated with the current board hash value (i.e., the address of the previous item) and a registration timestamp. Then, it signs the address of the new item and delivers it to the writer as proof of acceptance of the new item on the board. Note that it is the digital ballot box which ensures the link of the new item to the previous item on the board.

As of this modification, each bulletin board item references two other items:

- an existing item that the writer chooses as the parent item
- and the previous item on the board.

This modification to the bulletin board structure implies that the digital ballot box protects the *history* property described in [2]. Furthermore, we introduce a new property to the bulletin board called *ancestry*, which is defined by items being related to each other meaningfully. As a result, when traversed on the *ancestry* line, the structure of the bulletin board looks like a tree. However, when traversed on the *history* line, the structure looks linear.

In addition, we introduce a new concept to the bulletin board structure called a *hidden verification track*, used to perform the ballot checking process described in section 3.3.4. It is called:

- *hidden* because it is not publicly available as part of the bulletin board. Instead, it is available on request based on the *address* of a specific item.
- *verification* because it is used only for the ballot checking process.
- *track* because it spawns an extra *history* of events that is injected under a specific item from the main *history.*

As a consequence of these modifications, any i^{th} item from the bulletin board consists of the following tuple $b_i = (m_i, c_i, \mathcal{W}, \sigma_i, t_i, p_i, h'_i, h_i)$, where m_i is the item type, c_i is the content of the item that describes the event, $\mathcal{W}$ is a reference to the item writer, σ_i is the writer's signature, p_i is the address of the parent item with $p_i \in \{h_1, ..., h_{i-1}\}$, h'_i is the address of the previous item in the *history* (i.e., $h'_i = h_{i-1}$), t_i is the registration timestamp, and h_i is the item address.

Because of the two properties of the bulletin board, we define two auditing algorithms. Given a list of items $\boldsymbol{b} = \{b_1, ..., b_n\}$, an auditor runs $\mathsf{AncestryVer}(\boldsymbol{b}, h_0)$ (algorithm 1), where h_0 is the parent of the list (i.e., the parent of the first item of the list b_1) to check the *ancestry* of the list. Likewise, an auditor can run $\mathsf{HistoryVer}(\boldsymbol{b}, h_0)$ (algorithm 2), where h_0 is the previous item of the list to check the *history* property of the list.

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

```
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 $i \leftarrow 1$ to $n$ by 1 do
    if $h_i \neq \mathcal{H}(m_i || c_i || p_i || h'_i || t_i)$
    or $p_i \neq h_{i-1}$ then
        return 0                                    // ancestry is invalid
    end
end
return 1                                            // ancestry is valid
```

The different kinds of items (i.e., the values that m_i can have) and the events they support are described in section 2.4.2. The rules about how items can reference a parent item and what actors can write them are described in appendix B.

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

```
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 $i \leftarrow 1$ to $n$ by $1$ do
    if $h_i \neq \mathcal{H}(m_i||c_i||p_i||h'_i||t_i)$
    or $h'_i \neq h_{i-1}$ then
        return 0                                    // history is invalid
    end
end
return 1                                            // history is valid
```

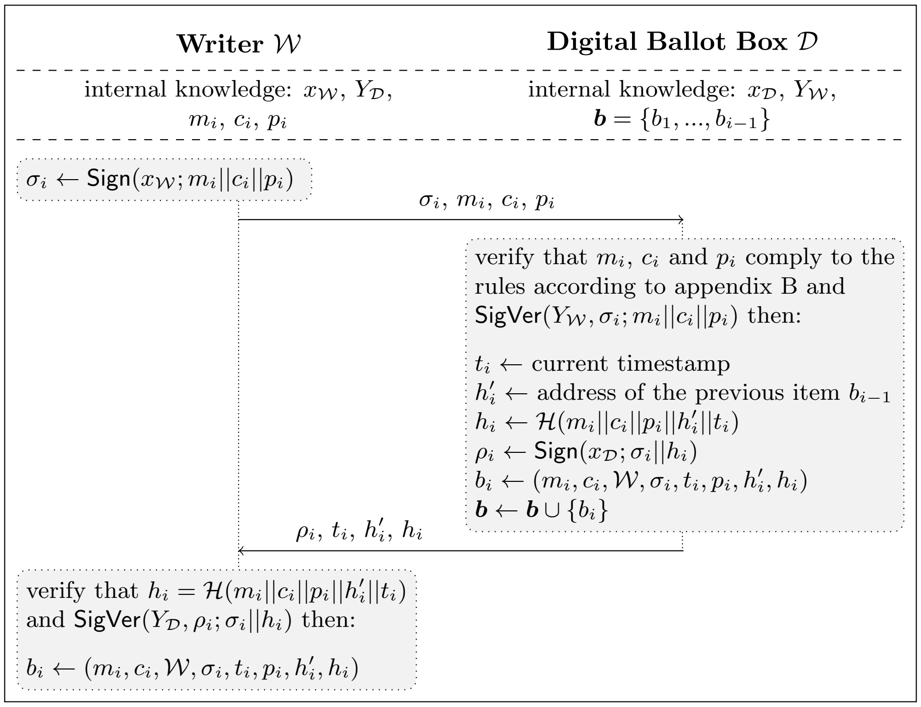
To write a new item on the bulletin board, a writer must follow the protocol described in section 2.4.1. The following actors are allowed to write on the bulletin board:

- Election Administrator $\mathcal{E}$, as the actor which writes all of the configuration events of an election.
- Voter Authorizer $\mathcal{A}$, as the actor which authorizes voters to interact with the digital ballot box based on successful authentication.
- Voters $\mathcal{V}_i$, with $i \in \{1, ..., n_{\text{v}}\}$, as the actors which write all of the vote-related events of an election.
- Digital Ballot Box $\mathcal{D}$, as the actor which ultimately accepts all of the events published on the bulletin board. In addition, $\mathcal{D}$ also writes all of the auxiliary events supporting the voting process on the board.
- External Verifier $\mathcal{X}$, as the actor which writes events related to the ballot checking process. These events are written on the hidden verification track of the bulletin board.

2.4.1 Writing on the bulletin board

This section describes the protocol that any writer must follow to write an event on the bulletin board. The election protocol allows a predefined set of actors ($\mathcal{E}$, $\mathcal{A}$, $\mathcal{V}_i$, $\mathcal{D}$, $\mathcal{X}$) to write events on the bulletin board. For the sake of generalization, protocol 1 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 this interaction as $(b_i, \rho_i) \leftarrow \texttt{WriteOnBoard}(\mathcal{W}, m_i, c_i, p_i)$ which outputs the new board item b_i and its receipt ρ_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 $x_{\mathcal{D}}$.


Writer $\mathcal{W}$
Digital Ballot Box $\mathcal{D}$
internal knowledge: $x_{\mathcal{W}}$, $Y_{\mathcal{D}}$, m_i, c_i, p_i
internal knowledge: $x_{\mathcal{D}}$, $Y_{\mathcal{W}}$, $\boldsymbol{b} = \{b_1, ..., b_{i-1}\}$
$\sigma_i \leftarrow \mathsf{Sign}(x_{\mathcal{W}}; m_i||c_i||p_i)$
σ_i, m_i, c_i, p_i
verify that m_i, c_i and p_i comply to the rules according to appendix B and $\mathsf{SigVer}(Y_{\mathcal{W}}, \sigma_i; m_i||c_i||p_i)$ then:
$t_i \leftarrow$ current timestamp
$h'_i \leftarrow$ address of the previous item b_{i-1}
$h_i \leftarrow \mathcal{H}(m_i||c_i||p_i||h'_i||t_i)$
$\rho_i \leftarrow \mathsf{Sign}(x_{\mathcal{D}}; \sigma_i||h_i)$
$b_i \leftarrow (m_i, c_i, \mathcal{W}, \sigma_i, t_i, p_i, h'_i, h_i)$
$\boldsymbol{b} \leftarrow \boldsymbol{b} \cup \{b_i\}$
ρ_i, t_i, h'_i, h_i
verify that $h_i = \mathcal{H}(m_i||c_i||p_i||h'_i||t_i)$ and $\mathsf{SigVer}(Y_{\mathcal{D}}, \rho_i; \sigma_i||h_i)$ then:
$b_i \leftarrow (m_i, c_i, \mathcal{W}, \sigma_i, t_i, p_i, h'_i, h_i)$

Protocol 1: `WriteOnBoard`$(\mathcal{W}, m_i, c_i, p_i)$

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. All of the event types are described in the section 2.4.2. The item content is a data structure describing a specific event which must follow the rules described in appendix B depending on the type of item chosen. 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 all board items $\boldsymbol{b}$. The choice of parent item is made according to the rules described in appendix B depending on the type of item chosen.

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 25) 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 B 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, for example, the 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).

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

```
Data: The board item $b = (m, c, \mathcal{W}, \sigma, t, p, h', h)$
      The public key of the writer $Y_\mathcal{W}$
if $h = \mathcal{H}(m||c||p||h'||t)$
and $\mathsf{SigVer}(Y_\mathcal{W}, \sigma; m||c||p)$          // algorithm 26
 then
    return 1                                   // item is valid
else
    return 0                                   // item is invalid
end
```

2.4.2 Bulletin board event types

The bulletin board has been designed as a self-documented event log. To support its function as such, it must contain many kinds of items that document different events throughout the election. These include events related to the pre-election phase for configuring the election, events related to the voting process, or events associated with the post-election phase for publishing a result. Each event is documented as an item on the bulletin board.

All bulletin board items are structured $(m_i, c_i, \mathcal{W}, \sigma_i, t_i, p_i, h'_i, h_i)$ as described in section 2.4.1 but each item type has its own rules when it comes to:

- what data c_i it contains,
- who the author $\mathcal{W}$ is,
- what parent p_i it can have.

The comprehensive list of item types and rules can be studied in appendix B. The list below briefly describes all item types which can be grouped into the following categories:

Configuration items

1. The *genesis* is the initial item of the bulletin board which describes some metadata of the election. This is, basically, the item which spawns a new bulletin board. It defines the elliptic curve domain parameters, i.e., the tuple (p, a, b, G, q, h), the public key of the Digital Ballot Box $Y_{\mathcal{D}}$, the public key of the Election Administrator $Y_{\mathcal{E}}$, and the URL of the bulletin board. This is the only item that doesn't have a parent reference, as it is the very first item on the board.

2. The *election configuration* specifies the configuration on the election level (e.g., election title, enabled languages). Follow-up *election configuration* items act like configuration updates. Generally, all configuration items reference the previous configuration item as a *parent*.

3. The *contest configuration* is an item defining the configuration of a contest. It contains a unique identifier of the contest, its marking rules, question type, result rules, and the list of candidates with their distinct labels $\{m_1, ..., m_{n_c}\}$, where n_c is the total number of candidates. Follow-up *contest configuration* items with the same contest identifier act like updates to that contest configuration.

4. The *threshold configuration* is the item defining the ballot encryption key Y_{enc} and the threshold setup t out-of n_t, where t is the amount of trustees needed for decryption and n_t is the total number of trustees. It also specifies all of the trustee data, that includes: the set of trustees $\boldsymbol{\mathcal{T}} = \{\mathcal{T}_1, ..., \mathcal{T}_{n_t}\}$, their public keys $Y_{\mathcal{T}_i}$ and their public polynomial coefficients $P_{\mathcal{T}_i,j}$, with $i \in \{1, ..., n_t\}$ and $j \in \{1, ..., t-1\}$. The threshold configuration cannot be updated during the election phase.

5. The *actor config* is an item that introduces a new actor on the bulletin board. This new actor is defined by a role and a public key. The role that actors can have is the *Voter Authorizer* $\mathcal{A}$, with its public key $Y_{\mathcal{A}}$. New roles might be included in the following versions of the protocol.

6. The *voter authorization configuration* is the item describing the way voters must authenticate themselves to be authorized to vote. The item defines the *voter authorization mode* and, if applicable (i.e. when *voter authorization mode* is **identity-based**), the configuration of all Idenitity Providers $\{\mathcal{I}_1, ..., \mathcal{I}_{n_i}\}$, where n_i is the number of providers.

7. The *voting round* item describes what contests can be voted on at the time. The item also defines how long the election phase lasts (i.e., the start and end dates). Multiple voting rounds can be enabled simultaneously or follow each other sequentially.

Voting items

8. The *voter session* is the item which documents that a new voter $\mathcal{V}_i$ has been authorized to cast a vote. The item contains the voter identifier vID_i, the voter's public key Y_i, the voter's weight, and an authentication fingerprint used for auditing. When a voter tries to vote again (therefore overwriting the previous vote), a new *voter session* item is generated containing the same voter identifier vID_i. The protocol for appending this item is described in section 3.3.1.

9. The *voter encryption commitment* is the item which settles the encryption parameters chosen by the voter during the vote cryptogram generation process (see section 3.3.3). The item consists only of a commitment c_{v} to the voter randomizer values. Note that this item is written by the voter, while the public key Y_i is defined in the *voter session* item.

10. The *server encryption commitment* is the item which settles the encryption parameters chosen by the Digital Ballot Box during the vote cryptogram generation process (see section 3.3.3). The item consists of the commitment c_{d} to the randomizer values of the Digital Ballot Box. This item is generated in response to the *voter encryption commitment* item being published.

11. The *ballot cryptograms* is the item which contains the encrypted digital vote, i.e., the cryptogram e_i.

12. The *cast request* is the item which documents the action of casting a previously submitted vote.

13. The *spoil request* is the item which documents the decision to challenge a previously submitted vote cryptogram, process described in section 3.3.4.

Hidden items

14. The *verification track start* is the initial item of the hidden verification track, essentially spawning a verification track for each *ballot cryptogram* item. The item is automatically written by the digital ballot box $\mathcal{D}$ after a *ballot cryptograms* item has been posted.

15. The *verifier* item defines the external verifier $\mathcal{X}$ and its public key $Y_{\mathcal{X}}$.

16. The *voter commitment opening* is the item containing the voter's encryption parameters which are necessary for unpacking the spoiled encrypted ballot. This data is encrypted, so only the external verifier can read it.

17. The *server commitment opening* is the item containing the encryption parameters of the digital ballot box, which are necessary for unpacking the spoiled encrypted ballot. This data is encrypted, so only the external verifier can read it. This item is generated as a response to the *voter commitment opening* item being published.

Result items

18. The *extraction intent* is the item which documents the request for a result to be computed. The request is made by the election administrator $\mathcal{E}$.
19. The *extraction data* is the item which lists all of the ballot cryptograms making up the *initial mixed board* (see details in section 3.4.1). These cryptograms are the only ones that will count as the election result.
20. The *extraction confirmation* is the item which documents that the result has been computed. It contains fingerprints of the files containing mixing and decryption data leading to the final result. All of this data is signed by the trustees, proving that the rightful actors have computed the result.

2.5 Voter authentication modes

During the pre-election phase, the voter authorizer service $\mathcal{A}$ is loaded with a list of eligible voters $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, ..., \mathcal{V}_{n_\mathrm{v}}\}$, where n_v is the total number of voters. To be authorized to cast a vote on the bulletin board, a user has to authenticate to the voter authorizer as voter $\mathcal{V}_i$, with $i \in \{1, ..., n_\mathrm{v}\}$. Once authenticated and authorized (section 3.3.1), voter $\mathcal{V}_i$ can interact directly with the digital ballot box in the voting protocol as described in section 3.3.3.

The voting system supports two mutually exclusive voter authentication modes: **credential-based** and **identity-based**. Both names refer to the means of the authentication taking place. One involves proving possession of some credentials that have been pre-established before the election starts, while the other consists in proving ownership of some identity provided by a third party. Some actors mentioned in the authentication modes presented below are exclusive to that mode only (i.e., credentials authority $\mathcal{C}$ for **credential-based** mode and identity provider $\mathcal{I}$ for **identity-based** mode).

2.5.1 Credential-based mode

For this mode, the voter authorizer configures a set of credentials authorities $\boldsymbol{\mathcal{C}} = \{\mathcal{C}_1, ..., \mathcal{C}_{n_\mathrm{c}}\}$, where n_c is the number of authorities, which are responsible for generating and distributing the voter credentials during the pre-election phase. Each credentials authority is supposed to use a distinct communication channel to distribute credentials to the voters (i.e., email, SMS, or postal). Therefore, all voters must be defined with contact information for each communication channel supported by all credentials authorities $\boldsymbol{\mathcal{C}}$.

All credentials are converted into private-public key pairs (as described in section 3.2.4) by all credentials authorities, which then return all voters' public keys to the voter authorizer. The voter authorizer aggregates all public keys for each voter to form their *authentication public key*.

When trying to authenticate to the voter authorizer, a voter must generate a *proof of credentials*, which can be achieved by aggregating all credentials received from all credentials authorities. If the proof validates, the voter authorizer $\mathcal{A}$ authorizes voter $\mathcal{V}_i$ to interact with the digital ballot box for casting a ballot. The entire authorization process is described in section 3.3.1.

2.5.2 Identity-based mode

In this mode, the voter authorizer service lists a set of third-party identity providers $\boldsymbol{\mathcal{I}} = \{\mathcal{I}_1, ..., \mathcal{I}_{n_\mathrm{i}}\}$, where n_i is the number of them.

Voters have to authenticate with all identity providers $\boldsymbol{\mathcal{I}}$ and receive identity tokens from each of them. Then, to get authorized to cast a vote, a voter must submit all identity tokens to the voter authorizer, which checks whether they relate to an eligible voter identity from $\boldsymbol{\mathcal{V}}$. The process is further described in section 3.3.1.

Because the voting system has to integrate into third-party identity providers, all voters must be defined with distinct identities supported by all identity providers $\boldsymbol{\mathcal{I}}$.

For auditing purposes, the voter authorizer stores all identity tokens received for each successful authorization performed. This must be audited and validated during the administration auditing process, as described in section 4.2.

2.6 Requirements

The requirements that the election protocol must fulfill are split into the following three categories: functional, non-functional, and security requirements.

2.6.1 Functional Requirements

Functional requirements relate to properties of the election system that voters, or users in general (including election officials, candidates, or auditors), can actively choose to perform. These properties are, in some way, measurable. The protocol has the following functional requirements:

- election types supported:
  - referendum: direct vote on a proposal or issue,
  - candidate election: one vote for a candidate from a predefined list,
  - multiple choice: a selection of multiple vote options,
  - ranked election: an ordered selection of multiple vote options,
  - write-in vote: a free-form text of a maximum size,
- verification mechanisms for voters to check that the encrypted ballot contains what they expect,
- possibility to confirm selected options after voting by an overview of the complete ballot,

- possibility to correct mistakes before submitting an encrypted ballot,
- ability for overwrite your vote, i.e., a voter can vote multiple times while only the latest submitted vote will be counted in the final tally,
- ability to check the status of your ballot after submission,
- public auditability of the election process throughout the election period.

2.6.2 Non-functional Requirements

Non-functional requirements describe properties of the election system that impact the user experience while interacting with the system.

Mobility is the property that enables voters to use any internet-connected device (PC, laptop, tablet, smartphone) to connect to the election system. They do not need to vote from a particular location (e.g., a polling station). Instead, they can participate in the voting process from any place they consider private and with an internet connection.

Vote & go entails that voters are only required to be present during the voting phase. Results can be computed without the presence of voters.

Transparency implies that election data is available for auditing through a public bulletin board.

Multiple voting rounds enable election officials to reuse most of the election configuration for multiple sub-elections where the same set of voters must vote on different ballots. A separate election result is computed for each voting round.

2.6.3 Security Requirements

Security requirements describe properties of the election system that contribute to the quality and reliability of an election result. This section briefly describes the properties, while the explanation of how these properties are achieved is presented in section 3.5.

Eligibility property is defined as the fact that only a limited number of predefined voters can cast a valid vote.

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

Anonymity property implies that no single entity can determine how a particular voter voted.

Integrity of voting data is the property that implies detection mechanisms of whether any votes recorded on the bulletin board during the election phase have been modified or deleted.

Verifiability property describes that all steps of the election protocol are verifiable by following some auditing process.

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 Election protocol

3.1 Overview

This section briefly describes the entire election protocol. A full election process is split into three main phases:

- the pre-election phase, where the election context is created and all components are configured, as presented in section 3.2,
- the election phase, where the actual votes are being generated and stored, as presented in section 3.3,
- and the post-election phase, where all collected votes are being processed into an election result, as described in section 3.4.

All of these phases internally consist of different processes that are triggered by specific stakeholders. A map of all processes is presented in figure 2, 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.

Specifically, an entire election process is started by an election official initializing a digital ballot box, as presented in section 3.2.1, and setting up the election configuration as in sections 3.2.2, 3.2.3 and 3.2.6. Then, the trustees perform the threshold ceremony, as described in section 3.2.5. Optionally, at the end of the pre-election phase, the voter credentials distribution process occurs, as presented in section 3.2.4.

During the election phase, voters can get authorized to cast a vote as presented in section 3.3.1 and then perform the voting process as described in sections 3.3.2, 3.3.3 and 3.3.5. Optionally, voters can perform some verification mechanisms on their encrypted ballot, as described in sections 3.3.4 and 3.3.5. More about voter-specific auditing is presented in section 4.1.

In the final, post-election phase, election officials run an administration auditing process to check that the election system behaved correctly, as described in section 4.2. Then, trustees compute the election result, as presented in sections 3.4.1 to 3.4.3. The result is published as in section 3.4.4, so any auditor can run a public auditing process on the entire election process, as presented in section 4.3.

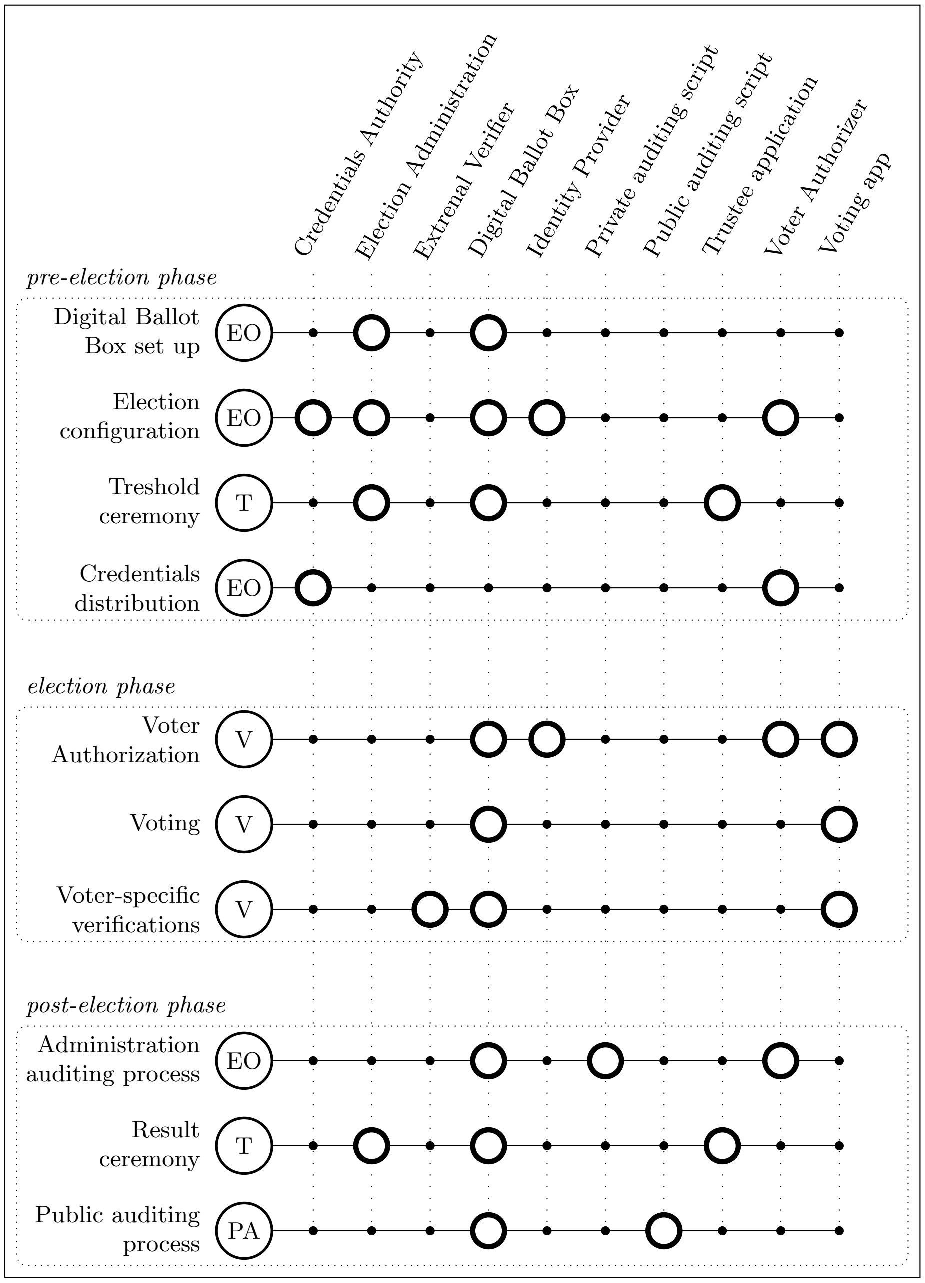


Figure 2: Processes map

3.2 Pre-election phase

During the *pre-election phase*, human election officials use the Election Administrator service to configure and set up a new election. This consists of the following steps:

- initiate a new bulletin board as described in section 3.2.1,
- define the election level configuration, including election title, contest titles, candidates, and other services required in the election, as described in section 3.2.2,
- define eligible voters and configure the voter authorization mode as described in section 3.2.3,
- facilitate the threshold ceremony as described in section 3.2.5,
- configure the election phase by setting up voting rounds as in section 3.2.6.

3.2.1 Digital Ballot Box initialization

An election official selects the elliptic curve domain parameters (p, a, b, G, q, h) for a predefined set, listed in appendix A.2.3. Based on these parameters, the election administrator service generates a new key pair $(x_{\mathcal{E}}, Y_{\mathcal{E}}) \leftarrow \mathsf{KeyGen}()$ (algorithm 17), 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 17), 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 $(b_1, \rho_1) \leftarrow \mathtt{WriteOnBoard}(\mathcal{D}, m_1, c_1, p_1)$ (protocol 1), where $m_1 =$ "genesis", $p_1 = \varnothing$, and the content c_1 is constructed according to the rules specified in appendix B. 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.

3.2.2 Election configuration

Once a bulletin board exists, the election administrator $\mathcal{E}$ can write all of the configuration items on it by following $\mathtt{WriteOnBoard}(\mathcal{E}, m_i, c_i, p_i)$ (protocol 1). All items are computed and published one by one, based on the rules defined in appendix B. 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,
- contest configuration items for each contest and
- actor configuration items for all other service which need to interact with the digital ballot box, e.g., the voter authorizer.

For each contest, the election official has to configure a contest identifier, marking rules, result rules, and a list of candidates represented by $\{m_1, ..., m_{n_\mathrm{c}}\}$, where n_c is the number of candidates.

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 17). 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 assigns the role of *voter authorizer* to the public key $Y_\mathcal{A}$. Once the actor configuration item containing the public key $Y_\mathcal{A}$ is published on the bulletin board, the voter authorizer becomes identity $\mathcal{A}$ and can interact with the digital ballot box.

3.2.3 Voter authorization configuration

An election official interacts with the Voter Authorizer service to configure the voter authentication mode, which can be either **credential-based** or **identity-based**. To define the list of eligible voters, the election official performs some extra configuration, which depends on the chosen voter authentication mode.

When **credential-based** voter authentication mode is enabled, the election official establishes a set of credentials authorities $\boldsymbol{\mathcal{C}} = \{\mathcal{C}_1, ..., \mathcal{C}_{n_c}\}$ used for distributing voter credentials, with each authority $\mathcal{C}_i \in \boldsymbol{\mathcal{C}}$ needing to use a specific communication channel to distribute voter credentials, such as via e-mail, post or SMS. Then the election official provides the list of eligible voters $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, ..., \mathcal{V}_{n_\mathrm{v}}\}$, each defined by a unique identifier and a list of contact information for all communication channel used by the credentials authorities.

Next, the credentials authorities and the voter authorizer perform the voter credential distribution process (describe in section 3.2.4) and set up the public authentication key of each voter.

Finally, the voter authorizer writes the voter authorization configuration item on the bulletin board by performing $\texttt{WriteOnBoard}(\mathcal{A}, m_i, c_i, p_i)$ (protocol 1) based on the rules defined in appendix B. The item is signed by the voter authorizer signing key $x_\mathcal{A}$ and it contains the voter authentication mode.

When **identity-based** voter authentication mode is enabled, the election official selects a list of third-party identity providers $\boldsymbol{\mathcal{I}} = \{\mathcal{I}_1, ..., \mathcal{I}_{n_\mathrm{i}}\}$ used for authenticating voters during the election phase. The election official then provides the list of eligible voters $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, ..., \mathcal{V}_{n_\mathrm{v}}\}$, each defined by a unique identifier and a list of identities supported by all of the identity providers.

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 1) based on the rules defined in appendix B. The item is signed by the voter authorizer signing key $x_{\mathcal{A}}$ and it contains the voter authentication mode and the list of identity providers $\mathcal{I}_j$, with $j \in \{1, ..., n_{\mathrm{i}}\}$, each defined by their public key $Y_{\mathcal{I}_j}$.

3.2.4 Voter credential distribution process

This process is only applicable if the vote authentication mode is **credential-based**, as described in section 2.5.1.

Each credentials authority $\mathcal{C}_j \in \boldsymbol{\mathcal{C}}$, receives a list of voters consisting of contact details for each voter $\{a_1, ..., a_{n_{\mathrm{v}}}\}$ in the form of e-mail addresses, postal addresses or phone numbers, depending on the credentials authority's communication channel. The credentials authority generates random credentials $c_{i,j} \in_{\mathrm{R}} \mathbb{B}^{\ell}$ for each voter, with $i \in \{1, ..., n_{\mathrm{v}}\}$. The credentials 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 $Y_{\mathrm{auth};i,j}$ in the list of voters next to $\mathcal{V}_i$, where $(x_{\mathrm{auth};i,j}, Y_{\mathrm{auth};i,j}) \leftarrow \mathsf{Pass2Key}(c_{i,j})$ (algorithm 35).

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, i.e., $\ell \geq 80$. That corresponds to a 14-character alphanumeric code that has to be sent to the voter and inputted in the voting application.

All credentials authorities $\mathcal{C}_j \in \boldsymbol{\mathcal{C}}$ return the lists with voters' contact details and public authentication keys $(a_i, Y_{\mathrm{auth};i,j})$ 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 $Y_{\mathrm{auth};i} = \sum_{j=1}^{n_{\mathrm{c}}} Y_{\mathrm{auth};i,j}$.

For authenticating to the voter authorizer, the voter $\mathcal{V}_i \in \boldsymbol{\mathcal{V}}$ must input all credentials $\{c_{i,1}, ..., c_{i,n_{\mathrm{c}}}\}$ received from all credentials authorities in the voting application. The application will thereafter derive keys from each credential $(x_{\mathrm{auth};i,j}, Y_{\mathrm{auth};i,j}) \leftarrow \mathsf{Pass2Key}(c_{i,j})$ (algorithm 35) and aggregate all of them to form the voter's private authentication key $x_{\mathrm{auth};i} = \sum_{j=1}^{n_{\mathrm{c}}} x_{\mathrm{auth};i,j} \pmod q$. The private authentication key is used to compute a proof of credentials PK_{auth}, as described in section 3.3.1, which is used to authenticate the voter.

3.2.5 Threshold ceremony

An election official interacts with the election administrator service to define the lists of trustees $\boldsymbol{\mathcal{T}} = \{\mathcal{T}_1, ..., \mathcal{T}_{n_{\mathrm{t}}}\}$. Then, the election administrator 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 A.5.3 to 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 administrator writes the threshold configuration item on the bulletin board by following `WriteOnBoard`$(\mathcal{E}, m_i, c_i, p_i)$ (protocol 1) based on the rules defined in appendix B. This item contains:

- election encryption key Y_{enc},
- 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 $j \in \{1, ..., t-1\}$.

3.2.6 Voting rounds

An election official interacts with the election administrator service to define when the election phase is taking 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 `WriteOnBoard`$(\mathcal{E}, m_i, c_i, p_i)$ (protocol 1) based on the rules from appendix B, specifying the start and end date, and the enabled contests.

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

3.3 Election phase

The election phase lasts from the start date until the end date of a voting round. During this time, any voter $\mathcal{V}_i \in \boldsymbol{\mathcal{V}}$ can cast a valid digital ballot by performing the following steps:

- obtain a list with all configuration items of the bulletin board α_{cnf} from the digital ballot box,
- authenticate and become authorized to cast a digital ballot on the bulletin board as described in section 3.3.1,
- select and encode vote choices as described in section 3.3.2,
- encrypt the ballot following the process from section 3.3.3,
- optionally, perform an audit/verification on the encrypted ballot as described in section 3.3.4 and
- finally, cast the encrypted ballot and obtain a vote confirmation receipt as in section 3.3.5.

3.3.1 Voter authorization procedure

A voter $\mathcal{V}_i$ is considered authorized to cast a digital ballot when he/she owns a secret signing key x_i which corresponds to an eligible signature verification key Y_i from the bulletin board. This is achieved differently depending on the voter authentication mode.

When credential-based voter authentication mode
Each voter $\mathcal{V}_i$ has to follow the protocol from figure 3 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 35 $\mathsf{Pass2Key}(c_j)$), 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 15) 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 $\boldsymbol{\mathcal{V}}$.

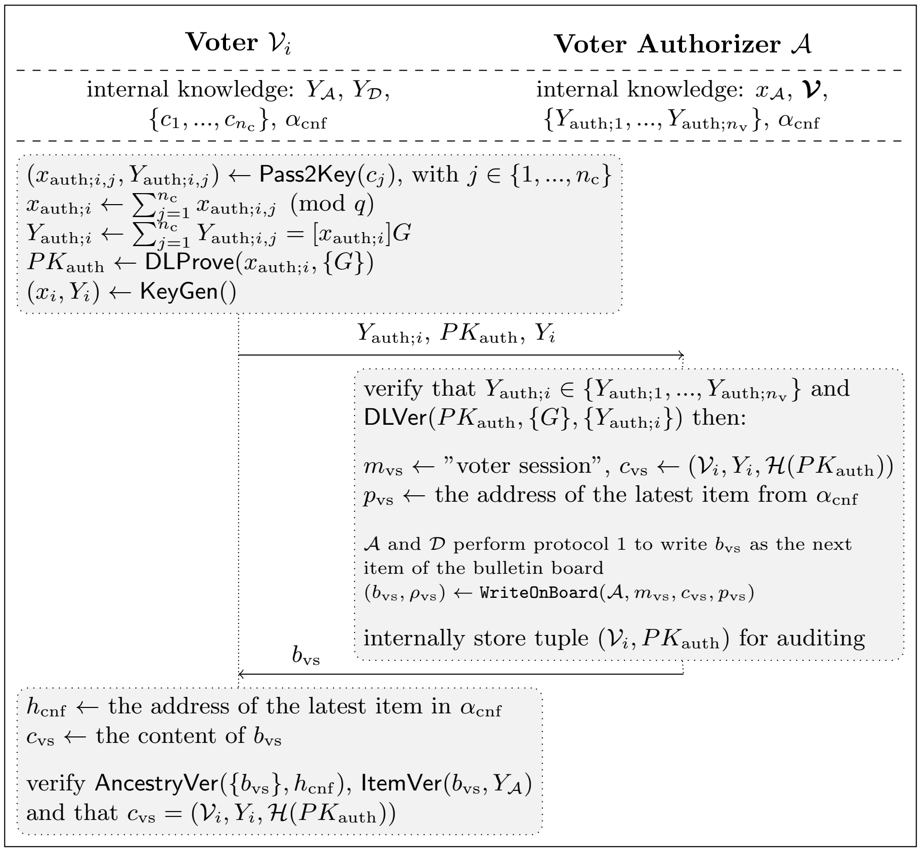
If the authentication succeeds, the voter authorizer service will authorize the use of public key Y_i for the voter $\mathcal{V}_i$ by interacting with the digital ballot box $\mathcal{D}$ in $\texttt{WriteOnBoard}(\mathcal{A}, m_{\text{vs}}, c_{\text{vs}}, p_{\text{vs}})$ (protocol 1) to write a voter session item b_{vs} as the next item on the bulletin board, according to the rules specified in appendix B, where $m_{\text{vs}} =$ "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 1. 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 4.2.

When identity-based voter athentication mode
Each voter $\mathcal{V}_i$ must follow the protocol from figure 4 to obtain authorization to cast a digital ballot on the bulletin board. Specifically, the voter must authenticate and receive identity tokens $\sigma_{\text{id},j}$ from all of the 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 17) and forwards all identity tokens $\{\sigma_{\text{id},1}, ..., \sigma_{\text{id},n_i}\}$ and the public key Y_i to the voter authorizer service $\mathcal{A}$ proving the identity of the voter $\mathcal{V}_i$.



Voter $\mathcal{V}_i$

internal knowledge: $Y_{\mathcal{A}}$, $Y_{\mathcal{D}}$, $\{c_1, ..., c_{n_c}\}$, α_{cnf}

Voter Authorizer $\mathcal{A}$

internal knowledge: $x_{\mathcal{A}}$, $\boldsymbol{\mathcal{V}}$, $\{Y_{\text{auth};1}, ..., Y_{\text{auth};n_v}\}$, α_{cnf}

$(x_{\text{auth};i,j}, Y_{\text{auth};i,j}) \leftarrow \mathsf{Pass2Key}(c_j)$, with $j \in \{1, ..., n_c\}$
$x_{\text{auth};i} \leftarrow \sum_{j=1}^{n_c} x_{\text{auth};i,j} \pmod q$
$Y_{\text{auth};i} \leftarrow \sum_{j=1}^{n_c} Y_{\text{auth};i,j} = [x_{\text{auth};i}]G$
$PK_{\text{auth}} \leftarrow \mathsf{DLProve}(x_{\text{auth};i}, \{G\})$
$(x_i, Y_i) \leftarrow \mathsf{KeyGen}()$

$\xrightarrow{Y_{\text{auth};i},\ PK_{\text{auth}},\ Y_i}$

verify that $Y_{\text{auth};i} \in \{Y_{\text{auth};1}, ..., Y_{\text{auth};n_v}\}$ and $\mathsf{DLVer}(PK_{\text{auth}}, \{G\}, \{Y_{\text{auth};i}\})$ then:

$m_{\text{vs}} \leftarrow$ "voter session", $c_{\text{vs}} \leftarrow (\mathcal{V}_i, Y_i, \mathcal{H}(PK_{\text{auth}}))$
$p_{\text{vs}} \leftarrow$ the address of the latest item from α_{cnf}

$\mathcal{A}$ and $\mathcal{D}$ perform protocol 1 to write b_{vs} as the next item of the bulletin board
$(b_{\text{vs}}, \rho_{\text{vs}}) \leftarrow \texttt{WriteOnBoard}(\mathcal{A}, m_{\text{vs}}, c_{\text{vs}}, p_{\text{vs}})$

internally store tuple $(\mathcal{V}_i, PK_{\text{auth}})$ for auditing

$\xleftarrow{b_{\text{vs}}}$

$h_{\text{cnf}} \leftarrow$ the address of the latest item in α_{cnf}
$c_{\text{vs}} \leftarrow$ the content of b_{vs}

verify $\mathsf{AncestryVer}(\{b_{\text{vs}}\}, h_{\text{cnf}})$, $\mathsf{ItemVer}(b_{\text{vs}}, Y_{\mathcal{A}})$ and that $c_{\text{vs}} = (\mathcal{V}_i, Y_i, \mathcal{H}(PK_{\text{auth}}))$

Figure 3: **Credential-based** voter authentication protocol

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 1 $\texttt{WriteOnBoard}(\mathcal{A}, m_{\text{vs}}, c_{\text{vs}}, p_{\text{vs}})$ 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 B, 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 1. 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 $\mathcal{V}_i$.

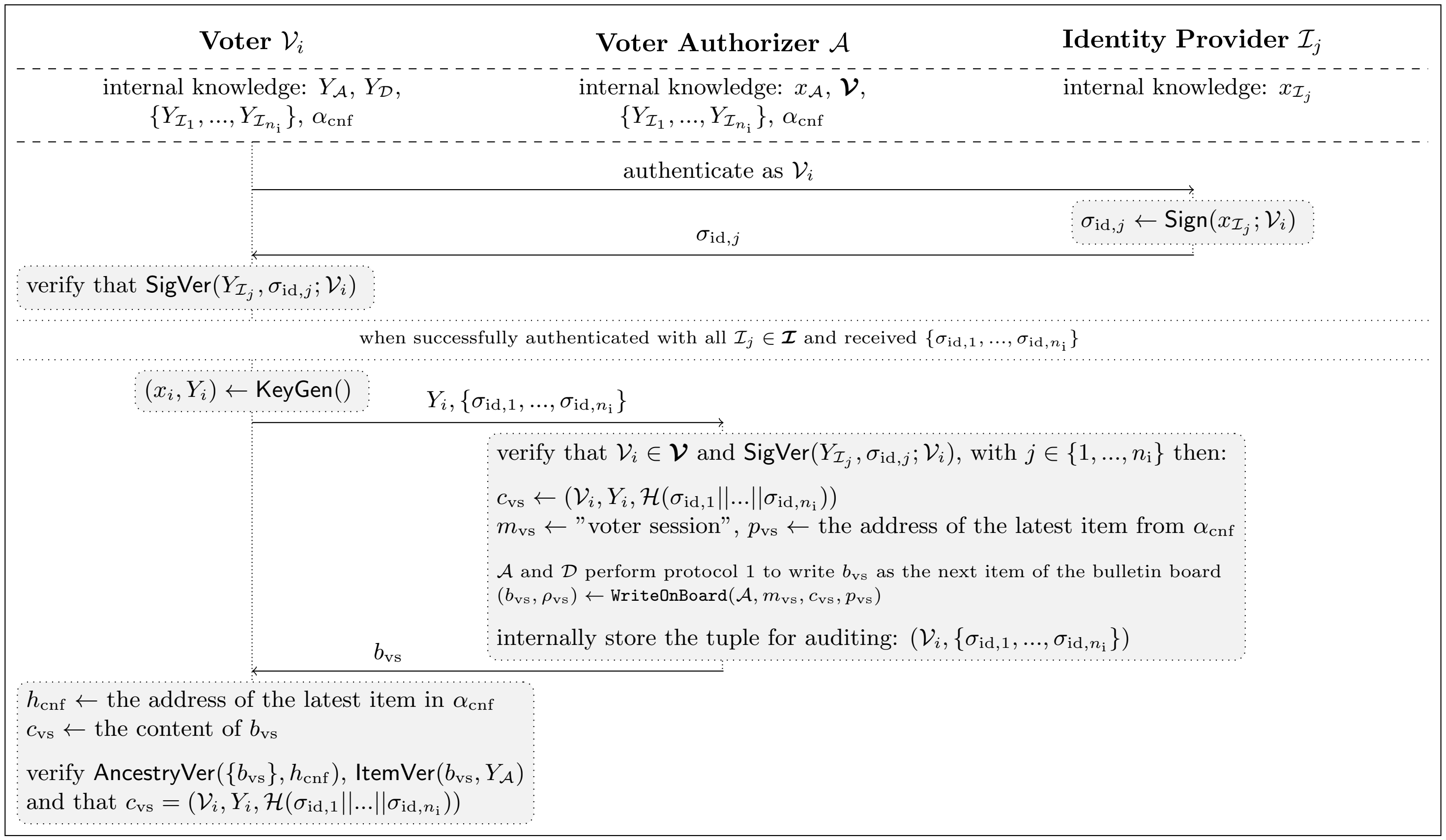

Voter $\mathcal{V}_i$
Voter Authorizer $\mathcal{A}$
Identity Provider $\mathcal{I}_j$
internal knowledge: $Y_{\mathcal{A}}$, $Y_{\mathcal{D}}$, $\{Y_{\mathcal{I}_1}, ..., Y_{\mathcal{I}_{n_\mathrm{i}}}\}$, α_{cnf}
internal knowledge: $x_{\mathcal{A}}$, $\boldsymbol{\mathcal{V}}$, $\{Y_{\mathcal{I}_1}, ..., Y_{\mathcal{I}_{n_\mathrm{i}}}\}$, α_{cnf}
internal knowledge: $x_{\mathcal{I}_j}$
authenticate as $\mathcal{V}_i$
$\sigma_{\mathrm{id},j} \leftarrow \mathsf{Sign}(x_{\mathcal{I}_j}; \mathcal{V}_i)$
$\sigma_{\mathrm{id},j}$
verify that $\mathsf{SigVer}(Y_{\mathcal{I}_j}, \sigma_{\mathrm{id},j}; \mathcal{V}_i)$
when successfully authenticated with all $\mathcal{I}_j \in \boldsymbol{\mathcal{I}}$ and received $\{\sigma_{\mathrm{id},1}, ..., \sigma_{\mathrm{id},n_\mathrm{i}}\}$
$(x_i, Y_i) \leftarrow \mathsf{KeyGen}()$
$Y_i, \{\sigma_{\mathrm{id},1}, ..., \sigma_{\mathrm{id},n_\mathrm{i}}\}$
verify that $\mathcal{V}_i \in \boldsymbol{\mathcal{V}}$ and $\mathsf{SigVer}(Y_{\mathcal{I}_j}, \sigma_{\mathrm{id},j}; \mathcal{V}_i)$, with $j \in \{1, ..., n_\mathrm{i}\}$ then:
$c_{\mathrm{vs}} \leftarrow (\mathcal{V}_i, Y_i, \mathcal{H}(\sigma_{\mathrm{id},1}||...||\sigma_{\mathrm{id},n_\mathrm{i}}))$
$m_{\mathrm{vs}} \leftarrow$ "voter session", $p_{\mathrm{vs}} \leftarrow$ the address of the latest item from α_{cnf}
$\mathcal{A}$ and $\mathcal{D}$ perform protocol 1 to write b_{vs} as the next item of the bulletin board
$(b_{\mathrm{vs}}, \rho_{\mathrm{vs}}) \leftarrow \mathtt{WriteOnBoard}(\mathcal{A}, m_{\mathrm{vs}}, c_{\mathrm{vs}}, p_{\mathrm{vs}})$
internally store the tuple for auditing: $(\mathcal{V}_i, \{\sigma_{\mathrm{id},1}, ..., \sigma_{\mathrm{id},n_\mathrm{i}}\})$
b_{vs}
$h_{\mathrm{cnf}} \leftarrow$ the address of the latest item in α_{cnf}
$c_{\mathrm{vs}} \leftarrow$ the content of b_{vs}
verify $\mathsf{AncestryVer}(\{b_{\mathrm{vs}}\}, h_{\mathrm{cnf}})$, $\mathsf{ItemVer}(b_{\mathrm{vs}}, Y_{\mathcal{A}})$
and that $c_{\mathrm{vs}} = (\mathcal{V}_i, Y_i, \mathcal{H}(\sigma_{\mathrm{id},1}||...||\sigma_{\mathrm{id},n_\mathrm{i}}))$

Figure 4: **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 4.2. 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.

3.3.2 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 vote encoding rules, the plain-text vote has a byte representation $\vec{b} \in \mathbb{B}^*$.

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 3.3.3. Thus, the set of points $\vec{V}$ represents the voter's choices in cryptographic form.

Recovering the byte array $\vec{b}$ from $\vec{V}$ can be done by $\vec{b} \leftarrow \mathsf{DecodeVote}(\vec{V})$ (algorithm 5). Depending on the vote encoding rules, the byte array $\vec{b}$ can further be interpreted as a plain-text vote.

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

```
Data: The plain-text vote $\vec{b} = \{b_1, ..., b_n\} \in \mathbb{B}^n$
$m \leftarrow \mathsf{ByteLengthOf}(p)$                          // algorithm 14
$\ell \leftarrow \lceil n/m \rceil$
for $i \leftarrow 0$ to $\ell - 1$ by $1$ do
    $V_i \leftarrow \mathsf{Bytes2Point}(\{b_{i+1}, ..., b_{i+m}\})$     // algorithm 10
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 $\ell$ by $1$ do
    $\vec{b} \leftarrow \vec{b} \cup \mathsf{Point2Bytes}(V_i)$             // algorithm 11
end
return $\vec{b}$                                            // $\vec{b} \in \mathbb{B}^*$
```

3.3.3 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 (recall from appendix A.5.1 that $e = \mathsf{Enc}(Y_{\text{enc}}, V; r)$). 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 for the voters not to know this value not to be able to produce cryptographic evidence of the way they voted (as in appendix A.5.4), thus achieving *receipt freeness*. The entire process consists of committing to the encryption randomizers (figure 5) and submitting the encrypted ballot (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 28), where $s_{\text{v}} \in_{\text{R}} \mathbb{Z}_q$. The voting application subsequently interacts with the digital ballot box in the protocol 1 $\texttt{WriteOnBoard}(\mathcal{V}_i, m_{\text{vec}}, c_{\text{vec}}, p_{\text{vec}})$ 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 3.3.1. Note that before appending the new item, the board consists of $\{b_1, ..., b_{k-1}\}$, thus b_{vec} becoming the k^{th} item.

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 28), 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 1 $\texttt{WriteOnBoard}(\mathcal{D}, m_{\text{sec}}, c_{\text{sec}}, p_{\text{sec}})$, 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 1 described in section 2.4.1 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 1 and continues, if successful.

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 3.3.2) by computing $e_{\text{v};i} \leftarrow \mathsf{Enc}(Y_{\text{enc}}, V_i, r_{\text{v};i})$ (algorithm 18), 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 20). The voting application also computes as proof of correct encryption $PK_i \leftarrow \mathsf{DLProve}(r_{\text{v};i}, \{G\})$ (algorithm 15) to confirm that the empty cryptograms $\vec{e}_{\text{d}}$ have been used in the creation of the final ballot cryptograms $\vec{e}$.

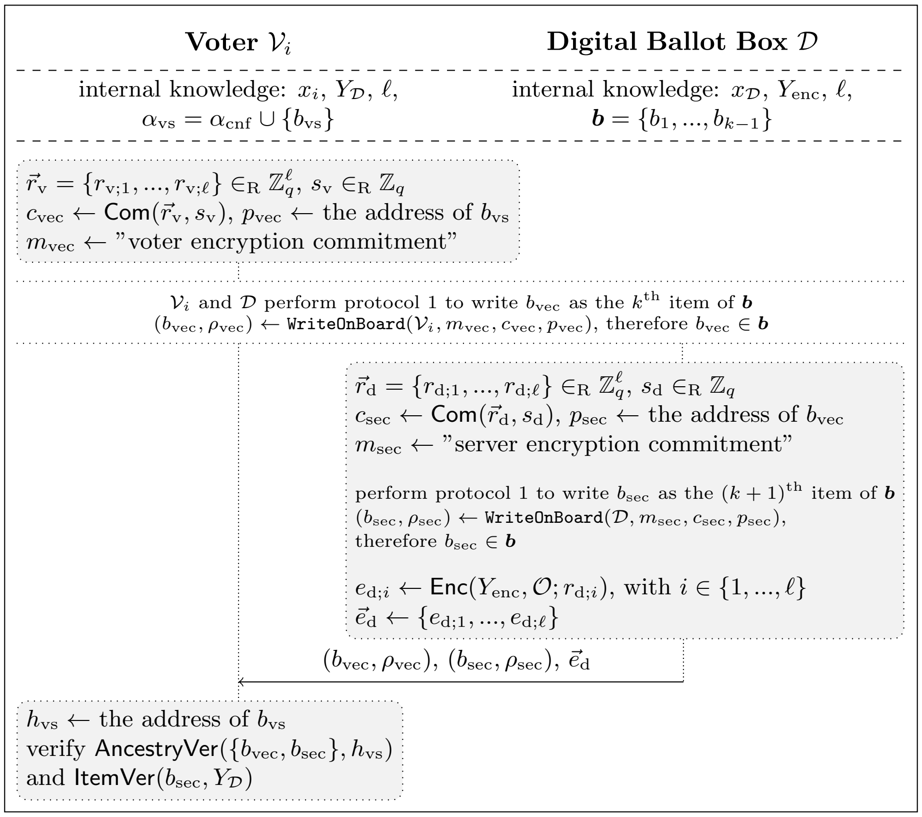

Voter $\mathcal{V}_i$
Digital Ballot Box $\mathcal{D}$
internal knowledge: x_i, $Y_\mathcal{D}$, ℓ, $\alpha_{\text{vs}} = \alpha_{\text{cnf}} \cup \{b_{\text{vs}}\}$
internal knowledge: $x_\mathcal{D}$, Y_{enc}, ℓ, $\boldsymbol{b} = \{b_1, ..., b_{k-1}\}$
$\vec{r}_{\text{v}} = \{r_{\text{v};1}, ..., r_{\text{v};\ell}\} \in_{\text{R}} \mathbb{Z}_q^\ell$, $s_{\text{v}} \in_{\text{R}} \mathbb{Z}_q$
$c_{\text{vec}} \leftarrow \mathsf{Com}(\vec{r}_{\text{v}}, s_{\text{v}})$, $p_{\text{vec}} \leftarrow$ the address of b_{vs}
$m_{\text{vec}} \leftarrow$ "voter encryption commitment"
$\mathcal{V}_i$ and $\mathcal{D}$ perform protocol 1 to write b_{vec} as the k^{th} item of $\boldsymbol{b}$
$(b_{\text{vec}}, \rho_{\text{vec}}) \leftarrow \texttt{WriteOnBoard}(\mathcal{V}_i, m_{\text{vec}}, c_{\text{vec}}, p_{\text{vec}})$, therefore $b_{\text{vec}} \in \boldsymbol{b}$
$\vec{r}_{\text{d}} = \{r_{\text{d};1}, ..., r_{\text{d};\ell}\} \in_{\text{R}} \mathbb{Z}_q^\ell$, $s_{\text{d}} \in_{\text{R}} \mathbb{Z}_q$
$c_{\text{sec}} \leftarrow \mathsf{Com}(\vec{r}_{\text{d}}, s_{\text{d}})$, $p_{\text{sec}} \leftarrow$ the address of b_{vec}
$m_{\text{sec}} \leftarrow$ "server encryption commitment"
perform protocol 1 to write b_{sec} as the $(k+1)^{\text{th}}$ item of $\boldsymbol{b}$
$(b_{\text{sec}}, \rho_{\text{sec}}) \leftarrow \texttt{WriteOnBoard}(\mathcal{D}, m_{\text{sec}}, c_{\text{sec}}, p_{\text{sec}})$,
therefore $b_{\text{sec}} \in \boldsymbol{b}$
$e_{\text{d};i} \leftarrow \mathsf{Enc}(Y_{\text{enc}}, \mathcal{O}; r_{\text{d};i})$, with $i \in \{1, ..., \ell\}$
$\vec{e}_{\text{d}} \leftarrow \{e_{\text{d};1}, ..., e_{\text{d};\ell}\}$
$(b_{\text{vec}}, \rho_{\text{vec}})$, $(b_{\text{sec}}, \rho_{\text{sec}})$, $\vec{e}_{\text{d}}$
$h_{\text{vs}} \leftarrow$ the address of b_{vs}
verify $\mathsf{AncestryVer}(\{b_{\text{vec}}, b_{\text{sec}}\}, h_{\text{vs}})$
and $\mathsf{ItemVer}(b_{\text{sec}}, Y_\mathcal{D})$

Figure 5: Encryption commitments submission protocol

Finally, 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 1) to append the ballot cryptogram item b_{bc} on the board, where $m_{\text{bc}} =$ "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 1 if $\mathsf{DLVer}(PK_i, \{G\}, \{R_i - [r_{\text{d};i}]G\})$ (algorithm 16) 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 $e_i = (R_i, C_i)$.

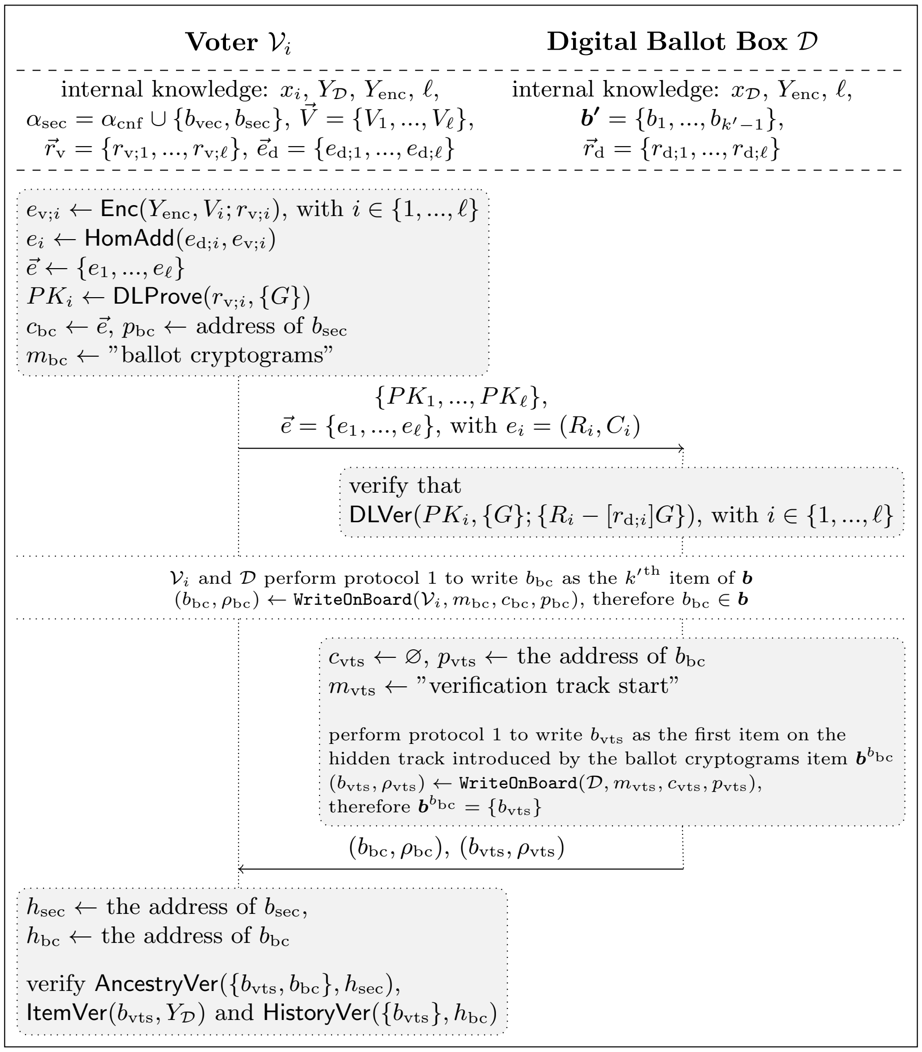

Voter $\mathcal{V}_i$
Digital Ballot Box $\mathcal{D}$
internal knowledge: x_i, $Y_{\mathcal{D}}$, Y_{enc}, ℓ, $\alpha_{\text{sec}} = \alpha_{\text{cnf}} \cup \{b_{\text{vec}}, b_{\text{sec}}\}$, $\vec{V} = \{V_1, ..., V_\ell\}$, $\vec{r}_{\text{v}} = \{r_{\text{v};1}, ..., r_{\text{v};\ell}\}$, $\vec{e}_{\text{d}} = \{e_{\text{d};1}, ..., e_{\text{d};\ell}\}$
internal knowledge: $x_{\mathcal{D}}$, Y_{enc}, ℓ, $\boldsymbol{b'} = \{b_1, ..., b_{k'-1}\}$, $\vec{r}_{\text{d}} = \{r_{\text{d};1}, ..., r_{\text{d};\ell}\}$
$e_{\text{v};i} \leftarrow \mathsf{Enc}(Y_{\text{enc}}, V_i; r_{\text{v};i})$, with $i \in \{1, ..., \ell\}$
$e_i \leftarrow \mathsf{HomAdd}(e_{\text{d};i}, e_{\text{v};i})$
$\vec{e} \leftarrow \{e_1, ..., e_\ell\}$
$PK_i \leftarrow \mathsf{DLProve}(r_{\text{v};i}, \{G\})$
$c_{\text{bc}} \leftarrow \vec{e}$, $p_{\text{bc}} \leftarrow$ address of b_{sec}
$m_{\text{bc}} \leftarrow$ "ballot cryptograms"
$\{PK_1, ..., PK_\ell\}$, $\vec{e} = \{e_1, ..., e_\ell\}$, with $e_i = (R_i, C_i)$
verify that $\mathsf{DLVer}(PK_i, \{G\}; \{R_i - [r_{\text{d};i}]G\})$, with $i \in \{1, ..., \ell\}$
$\mathcal{V}_i$ and $\mathcal{D}$ perform protocol 1 to write b_{bc} as the k'^{th} item of $\boldsymbol{b}$
$(b_{\text{bc}}, \rho_{\text{bc}}) \leftarrow \mathtt{WriteOnBoard}(\mathcal{V}_i, m_{\text{bc}}, c_{\text{bc}}, p_{\text{bc}})$, therefore $b_{\text{bc}} \in \boldsymbol{b}$
$c_{\text{vts}} \leftarrow \varnothing$, $p_{\text{vts}} \leftarrow$ the address of b_{bc}
$m_{\text{vts}} \leftarrow$ "verification track start"
perform protocol 1 to write b_{vts} as the first item on the hidden track introduced by the ballot cryptograms item $\boldsymbol{b}^{b_{\text{bc}}}$
$(b_{\text{vts}}, \rho_{\text{vts}}) \leftarrow \mathtt{WriteOnBoard}(\mathcal{D}, m_{\text{vts}}, c_{\text{vts}}, p_{\text{vts}})$,
therefore $\boldsymbol{b}^{b_{\text{bc}}} = \{b_{\text{vts}}\}$
$(b_{\text{bc}}, \rho_{\text{bc}})$, $(b_{\text{vts}}, \rho_{\text{vts}})$
$h_{\text{sec}} \leftarrow$ the address of b_{sec},
$h_{\text{bc}} \leftarrow$ the address of b_{bc}
verify $\mathsf{AncestryVer}(\{b_{\text{vts}}, b_{\text{bc}}\}, h_{\text{sec}})$, $\mathsf{ItemVer}(b_{\text{vts}}, Y_{\mathcal{D}})$ and $\mathsf{HistoryVer}(\{b_{\text{vts}}\}, h_{\text{bc}})$

Figure 6: Encrypted ballot submission protocol

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 $\mathtt{WriteOnBoard}(\mathcal{D}, m_{\text{vts}}, c_{\text{vts}}, p_{\text{vts}})$ (protocol 1), where $m_{\text{vts}} =$ "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 1. The voting application validates the two board items according to the protocol 1. 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.

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 $i \in \{1, ..., \ell\}$.

3.3.4 Challenging a vote cryptogram

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 3.3.3 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 7) 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 B. Note that the set of writers, as presented in section 2.4, 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 $\texttt{WriteOnBoard}(\mathcal{V}_i, m_{\text{sr}}, c_{\text{sr}}, p_{\text{sr}})$ (protocol 1) 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 17) and interacts with the digital ballot box in $\texttt{WriteOnBoard}(\mathcal{X}, m_{\text{v}}, c_{\text{v}}, p_{\text{v}})$ (protocol 1) 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 8 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 3.3.3. 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 23), 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 36). 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 1), 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 1), 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 9), 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 24) 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 3.3.3, i.e., verification of the voter commitment $\mathsf{ComVer}(c_{\text{v}}, \vec{r}_{\text{v}}, s_{\text{v}})$ (algorithm 29) 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 19), 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 3.3.2. 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.

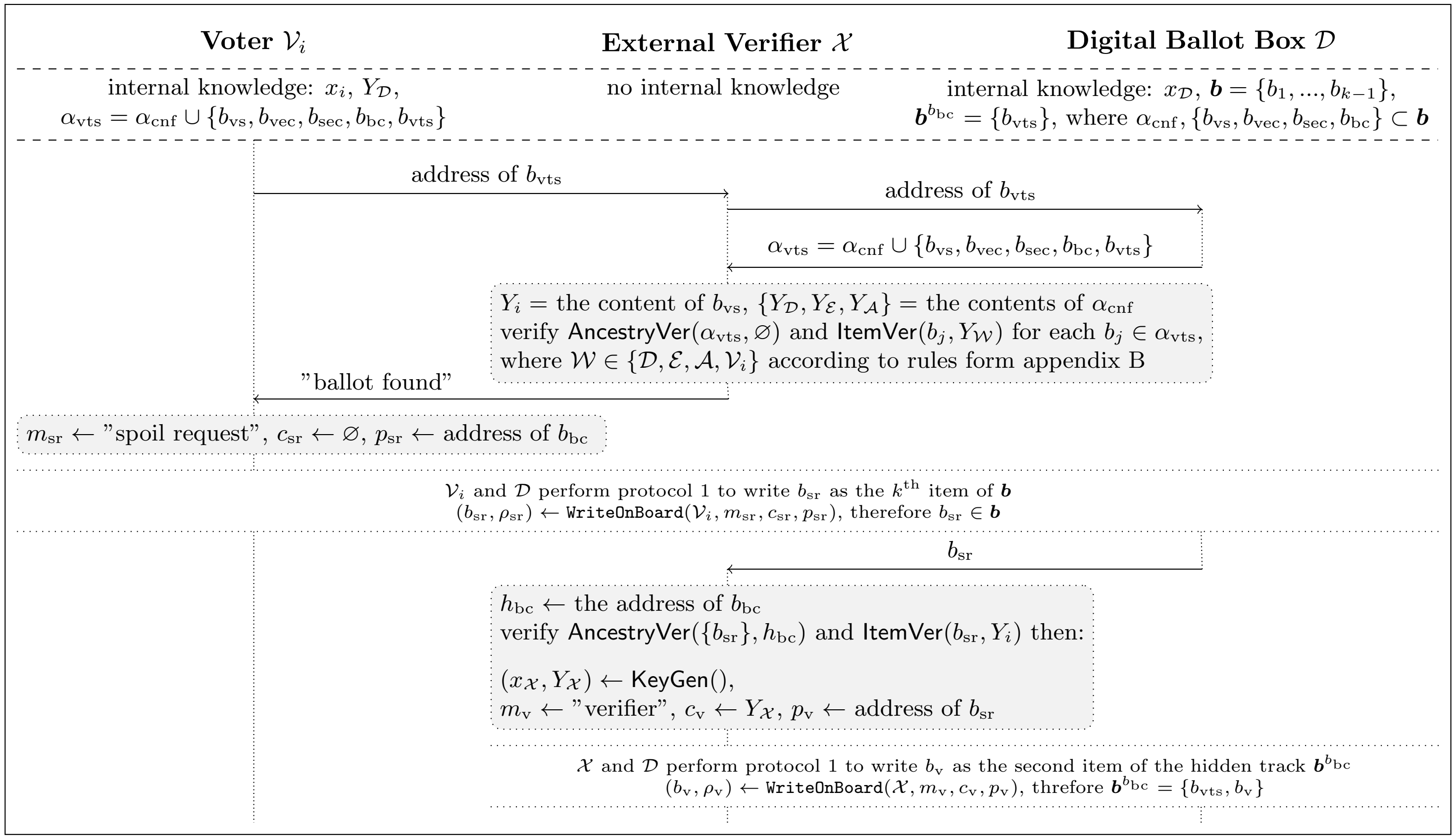

Voter $\mathcal{V}_i$
External Verifier $\mathcal{X}$
Digital Ballot Box $\mathcal{D}$
internal knowledge: x_i, $Y_\mathcal{D}$, $\alpha_{\text{vts}} = \alpha_{\text{cnf}} \cup \{b_{\text{vs}}, b_{\text{vec}}, b_{\text{sec}}, b_{\text{bc}}, b_{\text{vts}}\}$
no internal knowledge
internal knowledge: $x_\mathcal{D}$, $\boldsymbol{b} = \{b_1, ..., b_{k-1}\}$, $\boldsymbol{b}^{b_{\text{bc}}} = \{b_{\text{vts}}\}$, where $\alpha_{\text{cnf}}, \{b_{\text{vs}}, b_{\text{vec}}, b_{\text{sec}}, b_{\text{bc}}\} \subset \boldsymbol{b}$
address of b_{vts}
address of b_{vts}
$\alpha_{\text{vts}} = \alpha_{\text{cnf}} \cup \{b_{\text{vs}}, b_{\text{vec}}, b_{\text{sec}}, b_{\text{bc}}, b_{\text{vts}}\}$
$Y_i =$ the content of b_{vs}, $\{Y_\mathcal{D}, Y_\mathcal{E}, Y_\mathcal{A}\} =$ the contents of α_{cnf}
verify $\mathsf{AncestryVer}(\alpha_{\text{vts}}, \varnothing)$ and $\mathsf{ItemVer}(b_j, Y_\mathcal{W})$ for each $b_j \in \alpha_{\text{vts}}$,
where $\mathcal{W} \in \{\mathcal{D}, \mathcal{E}, \mathcal{A}, \mathcal{V}_i\}$ according to rules form appendix B
"ballot found"
$m_{\text{sr}} \leftarrow$ "spoil request", $c_{\text{sr}} \leftarrow \varnothing$, $p_{\text{sr}} \leftarrow$ address of b_{bc}
$\mathcal{V}_i$ and $\mathcal{D}$ perform protocol 1 to write b_{sr} as the k^{th} item of $\boldsymbol{b}$
$(b_{\text{sr}}, \rho_{\text{sr}}) \leftarrow \texttt{WriteOnBoard}(\mathcal{V}_i, m_{\text{sr}}, c_{\text{sr}}, p_{\text{sr}})$, therefore $b_{\text{sr}} \in \boldsymbol{b}$
b_{sr}
$h_{\text{bc}} \leftarrow$ the address of b_{bc}
verify $\mathsf{AncestryVer}(\{b_{\text{sr}}\}, h_{\text{bc}})$ and $\mathsf{ItemVer}(b_{\text{sr}}, Y_i)$ then:
$(x_\mathcal{X}, Y_\mathcal{X}) \leftarrow \mathsf{KeyGen}()$,
$m_{\text{v}} \leftarrow$ "verifier", $c_{\text{v}} \leftarrow Y_\mathcal{X}$, $p_{\text{v}} \leftarrow$ address of b_{sr}
$\mathcal{X}$ and $\mathcal{D}$ perform protocol 1 to write b_{v} as the second item of the hidden track $\boldsymbol{b}^{b_{\text{bc}}}$
$(b_{\text{v}}, \rho_{\text{v}}) \leftarrow \texttt{WriteOnBoard}(\mathcal{X}, m_{\text{v}}, c_{\text{v}}, p_{\text{v}})$, threfore $\boldsymbol{b}^{b_{\text{bc}}} = \{b_{\text{vts}}, b_{\text{v}}\}$

Figure 7: External verifier setup protocol

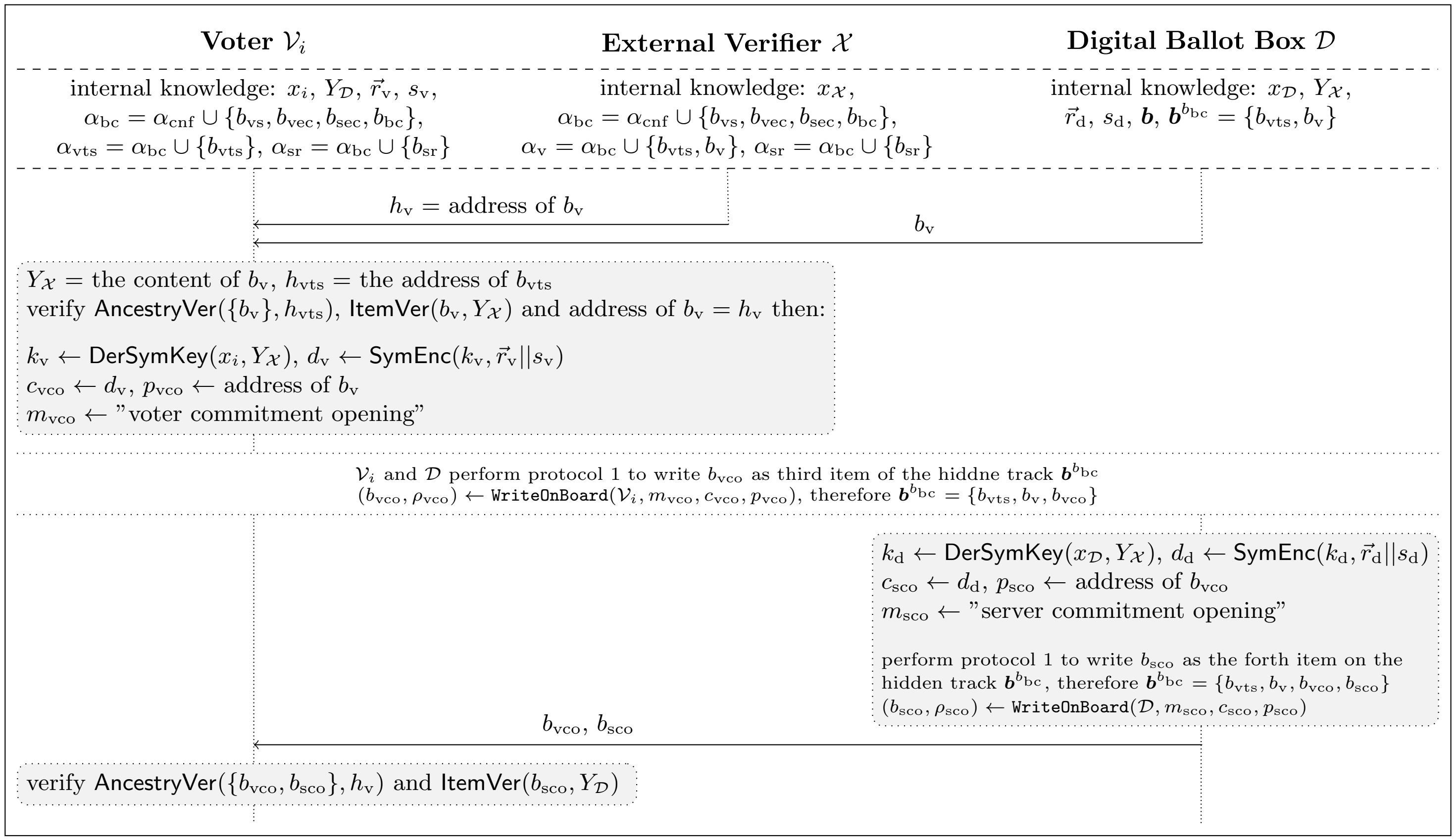
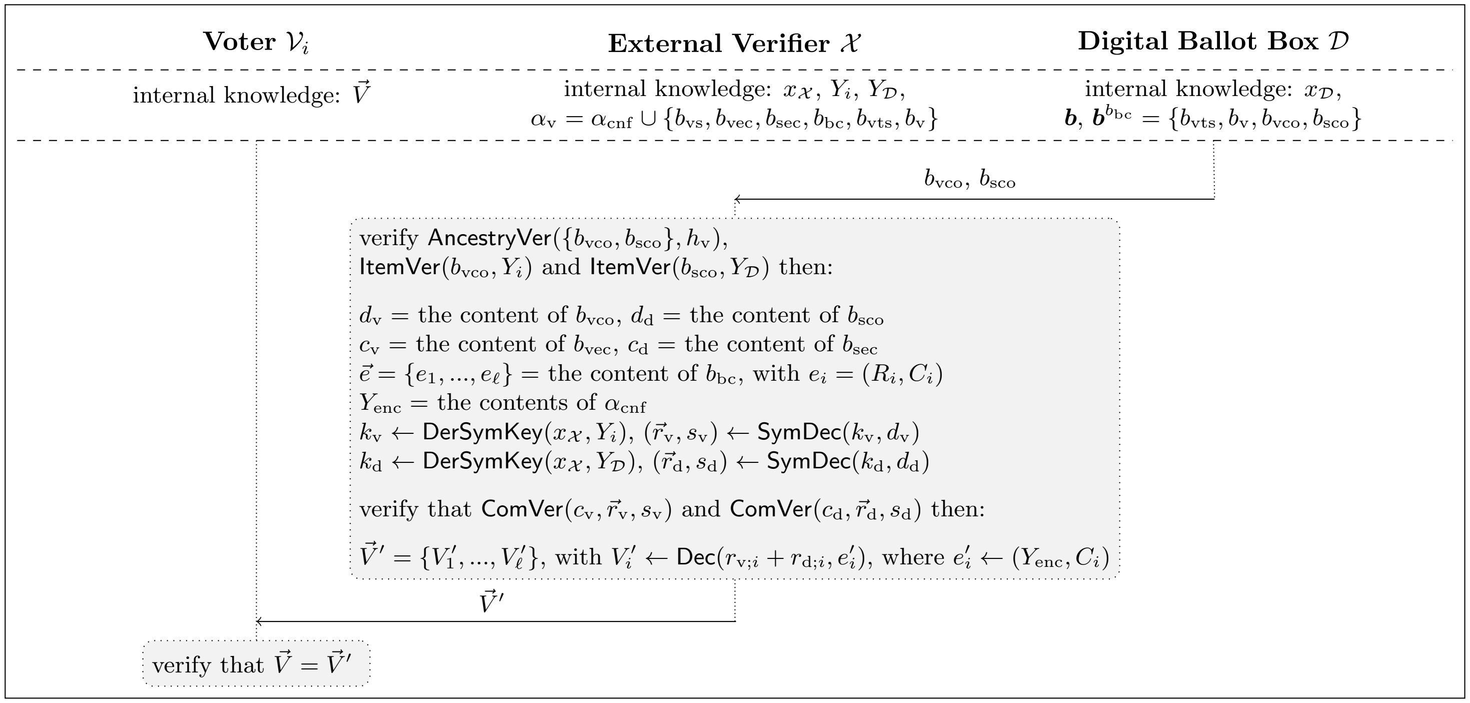

Voter $\mathcal{V}_i$
External Verifier $\mathcal{X}$
Digital Ballot Box $\mathcal{D}$
internal knowledge: x_i, $Y_{\mathcal{D}}$, $\vec{r}_{\mathrm{v}}$, s_{v},
$\alpha_{\mathrm{bc}} = \alpha_{\mathrm{cnf}} \cup \{b_{\mathrm{vs}}, b_{\mathrm{vec}}, b_{\mathrm{sec}}, b_{\mathrm{bc}}\}$,
$\alpha_{\mathrm{vts}} = \alpha_{\mathrm{bc}} \cup \{b_{\mathrm{vts}}\}$, $\alpha_{\mathrm{sr}} = \alpha_{\mathrm{bc}} \cup \{b_{\mathrm{sr}}\}$
internal knowledge: $x_{\mathcal{X}}$,
$\alpha_{\mathrm{bc}} = \alpha_{\mathrm{cnf}} \cup \{b_{\mathrm{vs}}, b_{\mathrm{vec}}, b_{\mathrm{sec}}, b_{\mathrm{bc}}\}$,
$\alpha_{\mathrm{v}} = \alpha_{\mathrm{bc}} \cup \{b_{\mathrm{vts}}, b_{\mathrm{v}}\}$, $\alpha_{\mathrm{sr}} = \alpha_{\mathrm{bc}} \cup \{b_{\mathrm{sr}}\}$
internal knowledge: $x_{\mathcal{D}}$, $Y_{\mathcal{X}}$,
$\vec{r}_{\mathrm{d}}$, s_{d}, $\boldsymbol{b}$, $\boldsymbol{b}^{b_{\mathrm{bc}}} = \{b_{\mathrm{vts}}, b_{\mathrm{v}}\}$
h_{v} = address of b_{v}
b_{v}
$Y_{\mathcal{X}}$ = the content of b_{v}, h_{vts} = the address of b_{vts}
verify $\mathsf{AncestryVer}(\{b_{\mathrm{v}}\}, h_{\mathrm{vts}})$, $\mathsf{ItemVer}(b_{\mathrm{v}}, Y_{\mathcal{X}})$ and address of $b_{\mathrm{v}} = h_{\mathrm{v}}$ then:
$k_{\mathrm{v}} \leftarrow \mathsf{DerSymKey}(x_i, Y_{\mathcal{X}})$, $d_{\mathrm{v}} \leftarrow \mathsf{SymEnc}(k_{\mathrm{v}}, \vec{r}_{\mathrm{v}}||s_{\mathrm{v}})$
$c_{\mathrm{vco}} \leftarrow d_{\mathrm{v}}$, $p_{\mathrm{vco}} \leftarrow$ address of b_{v}
$m_{\mathrm{vco}} \leftarrow$ ”voter commitment opening”
$\mathcal{V}_i$ and $\mathcal{D}$ perform protocol 1 to write b_{vco} as third item of the hiddne track $\boldsymbol{b}^{b_{\mathrm{bc}}}$
$(b_{\mathrm{vco}}, \rho_{\mathrm{vco}}) \leftarrow \mathtt{WriteOnBoard}(\mathcal{V}_i, m_{\mathrm{vco}}, c_{\mathrm{vco}}, p_{\mathrm{vco}})$, therefore $\boldsymbol{b}^{b_{\mathrm{bc}}} = \{b_{\mathrm{vts}}, b_{\mathrm{v}}, b_{\mathrm{vco}}\}$
$k_{\mathrm{d}} \leftarrow \mathsf{DerSymKey}(x_{\mathcal{D}}, Y_{\mathcal{X}})$, $d_{\mathrm{d}} \leftarrow \mathsf{SymEnc}(k_{\mathrm{d}}, \vec{r}_{\mathrm{d}}||s_{\mathrm{d}})$
$c_{\mathrm{sco}} \leftarrow d_{\mathrm{d}}$, $p_{\mathrm{sco}} \leftarrow$ address of b_{vco}
$m_{\mathrm{sco}} \leftarrow$ ”server commitment opening”
perform protocol 1 to write b_{sco} as the forth item on the hidden track $\boldsymbol{b}^{b_{\mathrm{bc}}}$, therefore $\boldsymbol{b}^{b_{\mathrm{bc}}} = \{b_{\mathrm{vts}}, b_{\mathrm{v}}, b_{\mathrm{vco}}, b_{\mathrm{sco}}\}$
$(b_{\mathrm{sco}}, \rho_{\mathrm{sco}}) \leftarrow \mathtt{WriteOnBoard}(\mathcal{D}, m_{\mathrm{sco}}, c_{\mathrm{sco}}, p_{\mathrm{sco}})$
b_{vco}, b_{sco}
verify $\mathsf{AncestryVer}(\{b_{\mathrm{vco}}, b_{\mathrm{sco}}\}, h_{\mathrm{v}})$ and $\mathsf{ItemVer}(b_{\mathrm{sco}}, Y_{\mathcal{D}})$

Figure 8: Commitment opening submission protocol


Voter $\mathcal{V}_i$
External Verifier $\mathcal{X}$
Digital Ballot Box $\mathcal{D}$
internal knowledge: $\vec{V}$
internal knowledge: $x_{\mathcal{X}}$, Y_i, $Y_{\mathcal{D}}$,
$\alpha_{\text{v}} = \alpha_{\text{cnf}} \cup \{b_{\text{vs}}, b_{\text{vec}}, b_{\text{sec}}, b_{\text{bc}}, b_{\text{vts}}, b_{\text{v}}\}$
internal knowledge: $x_{\mathcal{D}}$,
$\boldsymbol{b}$, $\boldsymbol{b}^{b_{\text{bc}}} = \{b_{\text{vts}}, b_{\text{v}}, b_{\text{vco}}, b_{\text{sco}}\}$
b_{vco}, b_{sco}
$\vec{V}'$
verify that $\vec{V} = \vec{V}'$

verify $\mathsf{AncestryVer}(\{b_{\text{vco}}, b_{\text{sco}}\}, h_{\text{v}})$,
$\mathsf{ItemVer}(b_{\text{vco}}, Y_i)$ and $\mathsf{ItemVer}(b_{\text{sco}}, Y_{\mathcal{D}})$ then:

d_{v} = the content of b_{vco}, d_{d} = the content of b_{sco}
c_{v} = the content of b_{vec}, c_{d} = the content of b_{sec}
$\vec{e} = \{e_1, ..., e_\ell\}$ = the content of b_{bc}, with $e_i = (R_i, C_i)$
Y_{enc} = the contents of α_{cnf}
$k_{\text{v}} \leftarrow \mathsf{DerSymKey}(x_{\mathcal{X}}, Y_i)$, $(\vec{r}_{\text{v}}, s_{\text{v}}) \leftarrow \mathsf{SymDec}(k_{\text{v}}, d_{\text{v}})$
$k_{\text{d}} \leftarrow \mathsf{DerSymKey}(x_{\mathcal{X}}, Y_{\mathcal{D}})$, $(\vec{r}_{\text{d}}, s_{\text{d}}) \leftarrow \mathsf{SymDec}(k_{\text{d}}, d_{\text{d}})$

verify that $\mathsf{ComVer}(c_{\text{v}}, \vec{r}_{\text{v}}, s_{\text{v}})$ and $\mathsf{ComVer}(c_{\text{d}}, \vec{r}_{\text{d}}, s_{\text{d}})$ then:

$\vec{V}' = \{V_1', ..., V_\ell'\}$, with $V_i' \leftarrow \mathsf{Dec}(r_{\text{v};i} + r_{\text{d};i}, e_i')$, where $e_i' \leftarrow (Y_{\text{enc}}, C_i)$

Figure 9: Unpacking the encrypted ballot protocol

3.3.5 Vote confirmation receipt

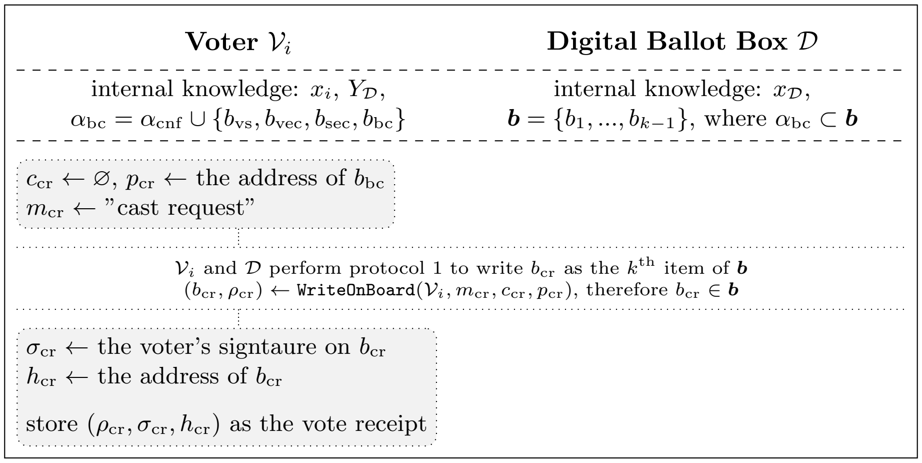
After encrypting a ballot, the voter $\mathcal{V}_i$ can choose whether to test or cast it. After deciding to cast the ballot, the voter receives a receipt from the digital ballot box that confirms that the ballot has been registered as cast on the public bulletin board.

The voter has to follow the protocol from figure 10 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 1) 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 1, 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 4.1.2.

Note that if a voter $\mathcal{V}_i$ has a vote confirmation receipt $(\rho_{\text{cr}}, \sigma_{\text{cr}}, h_{\text{cr}})$ that is valid (i.e., $\mathsf{SigVer}(Y_{\mathcal{D}}, \rho_{\text{cr}}; \sigma_{\text{cr}}||h_{\text{cr}})$ (algorithm 26), where $Y_{\mathcal{D}}$ exists on the bulletin board $\boldsymbol{b}$) but does not correspond with the current state of the bulletin board (i.e., h_{cr} is not the address of any item of 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.


Voter $\mathcal{V}_i$ | **Digital Ballot Box $\mathcal{D}$**

internal knowledge: x_i, $Y_{\mathcal{D}}$, $\alpha_{\text{bc}} = \alpha_{\text{cnf}} \cup \{b_{\text{vs}}, b_{\text{vec}}, b_{\text{sec}}, b_{\text{bc}}\}$

internal knowledge: $x_{\mathcal{D}}$, $\boldsymbol{b} = \{b_1, ..., b_{k-1}\}$, where $\alpha_{\text{bc}} \subset \boldsymbol{b}$

$c_{\text{cr}} \leftarrow \varnothing$, $p_{\text{cr}} \leftarrow$ the address of b_{bc}
$m_{\text{cr}} \leftarrow$ "cast request"

$\mathcal{V}_i$ and $\mathcal{D}$ perform protocol 1 to write b_{cr} as the k^{th} item of $\boldsymbol{b}$
$(b_{\text{cr}}, \rho_{\text{cr}}) \leftarrow \texttt{WriteOnBoard}(\mathcal{V}_i, m_{\text{cr}}, c_{\text{cr}}, p_{\text{cr}})$, therefore $b_{\text{cr}} \in \boldsymbol{b}$

$\sigma_{\text{cr}} \leftarrow$ the voter's signtaure on b_{cr}
$h_{\text{cr}} \leftarrow$ the address of b_{cr}

store $(\rho_{\text{cr}}, \sigma_{\text{cr}}, h_{\text{cr}})$ as the vote receipt

Figure 10: Ballot casting protocol

3.4 Post-election phase

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 voters to check that their encrypted ballot is included (using their confirmation receipt) and for auditors to check that the list item addresses are consistent (the integrity of the board is persistent).

The process of computing a result consists of the following:

- the election administrator requests a result to be computed by interacting with the digital ballot box in `WriteOnBoard`$(\mathcal{E}, m_{\mathrm{ei}}, c_{\mathrm{ei}}, p_{\mathrm{ei}})$ (protocol 1) to publish an extraction intent item b_{ei} on the bulletin board, according to the rules from appendix B,
- the digital ballot box identifies all the valid ballots to be included in the tally, according to section 3.4.1,
- a subset of all trustees $\mathcal{T}_i$, with $i \in \tau$ and $\tau \subset \{1, ..., n_{\mathrm{t}}\}$, collaborate in the mixing process to anonymize the encrypted ballots, as described in section 3.4.2, where n_{t} is the total number of trustees and $t \leq |\tau| \leq n_{\mathrm{t}}$ (recall from section 3.2.5 that t is threshold decryption value),
- the same subset of trustees collaborate in the decryption process (section 3.4.3) of the anonymized votes,
- finally, the election administrator publishes the result in a verifiable manner as described in section 3.4.4.

This process of computing a result is executed separately for each voting round configured in the pre-election phase (described in section 3.2.6).

3.4.1 Cleansing procedure

Triggered by the extraction intent item b_{ei} being published, the digital ballot box $\mathcal{D}$ bundles a matrix of cryptograms that represent only the valid votes from the ballot cryptograms items from the bulletin board. Each matrix line consists of a set of cryptograms that make up one ballot. To be considered valid, the cryptograms must be extracted from the latest ballot cryptograms item for each voter, followed by a cast request item. All other ballots are considered overwritten and, therefore, discarded. This is, essentially, the votes that will be decrypted and tallied as a result.

The matrix of cryptograms is called *the initial mixed board*, and it is defined as $\vec{e}_0 = \{e_{1,1}, ..., e_{n_{\mathrm{e}},\ell}\}$, with each $e_{i,j} \in \mathbb{E}$, where $\mathbb{E}$ is the set of all possible cryptograms, n_{e} is the total number of extracted ballots, and ℓ is the number of cryptograms a ballot is made out of. Note that the set $\{e_{i,1}, ..., e_{i,\ell}\}$ represent the cryptograms that make up the i^{th} ballot.

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 1), where $m_{\text{ed}} =$ "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 *initial mixed board* $\vec{\vec{e}}_0$ is used as the input to the mixing phase.

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

3.4.2 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 30), 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 A.8, 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 33). 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 34). 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.

3.4.3 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 figure 13. 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 3.2.5).

Then, trustee $\mathcal{T}_i$ submits the partial decryption $\vec{\vec{S}}_i$ and the proof PK_i to the election administrator service, which in figure 13 is referred to as *the server*. The election administrator 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), where sY_i is the public share of the trustee $\mathcal{T}_i$, which is computable as in appendix A.5.3.

Upon receiving partial decryptions $\vec{\vec{S}}_i$ from all trustees $\mathcal{T}_i$ with $i \in \tau$, the election administrator follows the protocol from figure 13 and 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). $\vec{\vec{V}}$ represents the *raw result* of the election, i.e., the full list of votes as elliptic curve points.

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

```
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 $\ell$ by $1$ do
        $S_{i,j} \leftarrow [sx]R_{i,j}$
    end
end
$\vec{\vec{S}} \leftarrow \{S_{1,1}, ..., S_{n,\ell}\}$
$\vec{R} \leftarrow \{G, R_{1,1}, ..., R_{n,\ell}\}$
$PK \leftarrow \mathsf{DLProve}(sx, \vec{R})$                          // algorithm 15
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($\vec{e}, \vec{\vec{S}}, PK, sY$)

```
Data: The matrix of cryptograms $\vec{e} = \{e_{1,1}, ..., e_{n,\ell}\} \in \mathbb{E}^{n \times \ell}$, 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 $sY \in \mathbb{P}$
$\vec{R} \leftarrow \{G, R_{1,1}, ..., R_{n,\ell}\}$
$\vec{S} \leftarrow \{sY, S_{1,1}, ..., S_{n,\ell}\}$
$b \leftarrow \mathsf{DLVer}(PK, \vec{R}, \vec{S})$                // algorithm 16
return $b$                                        // $b \in \mathbb{B}$
```

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

```
Data: The matrix of cryptograms $\vec{e} = \{e_{1,1}, ..., e_{n,\ell}\} \in \mathbb{E}^{n \times \ell}$, 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 $\ell$ by 1 do
        $V_{i,j} \leftarrow C_{i,j} - \sum_{k \in \tau} [\lambda(k)] S_{k,i,j}$    // $\lambda(k)$ computed as in figure 13
    end
end
$\vec{\vec{V}} \leftarrow \{V_{1,1}, ..., V_{n,\ell}\}$
return $\vec{\vec{V}}$                                   // $\vec{\vec{V}} \in \mathbb{P}^{n \times \ell}$
```

3.4.4 Result Publication

The *results module* is responsible for interpreting the *raw result* and presenting the result of the election 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 has been computed in the result ceremony (both mixing and decryption phases) is collected by the election administrator $\mathcal{E}$ and published on the bulletin board as the extraction confirmation item b_{ec} by performing $\texttt{WriteOnBoard}(\mathcal{E}, m_{\text{ec}}, c_{\text{ec}}, p_{\text{ec}})$ (protocol 1), where m_{ec} = "extraction confirmation", the parent p_{ec} is the address of the extraction data item b_{ed} and the content c_{ec} consists of:

- 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

3.5 Election properties

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 $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, ..., \mathcal{V}_{n_{\text{v}}}\}$ 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. This is achieved differently, depending on the voter authentication mode that has been configured during the pre-election phase. The two voter authentication modes are described in section 2.5.

When voter authentication mode is **identity-based**, voters successfully authenticate to the voter authorizer the moment they manage to authenticate to all identity providers $\boldsymbol{\mathcal{I}} = \{\mathcal{I}_1, ..., \mathcal{I}_{n_\mathrm{i}}\}$, where n_i is the number of identity providers. 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, when voter authentication mode is **identity-based**, 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 authentication 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 3.2.4), where n_c is the total number of authorities. The proof is verified against the voter's public authentication key, computed by the voter authorizer by aggregating all public authentication keys received from all credentials authorities.

Therefore, when **credential-based** voter authentication mode is used, our protocol has the eligibility property on the assumption that there are multiple, distinct credentials authorities participating in the voter credential distribution process (section 3.2.4).

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.

Privacy: Following the election protocol, no partial results are computed during the election process. A result is calculated only once after the election phase has finished. This prevents influencing the subsequent voters throughout the election period.

All votes that are posted on the bulletin board are encrypted using the ElGamal cryptosystem based on elliptic curve cryptography (more details in appendix A.1 and appendix A.5). Moreover, using a *t out of n* threshold encryption scheme (section 3.2.5), it is enforced that there is no single entity that can perform the decryption of any data from the public bulletin board. Instead, minimum t trustees are required to collaborate to compute a result.

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 until the elliptic curve cryptosystem is broken, and so will be able to decrypt all the data contrary to our protocol. This demonstrates that our system does not comply with the *everlasting privacy* property. We take note of this fact and accept it.

Anonymity: This property is reached by implementing a mix-net of nodes (trustees) that sequentially shuffle the list of vote cryptograms in an indistinguishable way before they get decrypted (section 3.4.2).

Obviously, each trustee knows how it shuffled the list of cryptograms but does not know how the other trustees shuffled it. Thus, it is essential that trustees do not communicate with each other.

We claim that our protocol has the *anonymous* property on the assumption that there is a set of multiple trustees, out of which at least one is honest.

Integrity: The integrity of the election is preserved in our system by publishing all events (i.e., election configuration data or voting-related data) on the bulletin board. Moreover, the bulletin board has a *hash-chain* structure that guarantees that the history of the bulletin board never changes. Also, the voters certify the authenticity of the bulletin board whenever they submit a new vote cryptogram by signing on the history of the bulletin board.

Every time a new item is appended on the bulletin board, the writer of that item receives a confirmation receipt ρ_i that contains a pointer to the item on the bulletin board, called the address (or the board hash value) h_i. This value is computed based on the previous board hash value h_{i-1} (the address of the previous item), which is calculated based on the one before, and so on, until it reaches the *genesis* item, which uses value 00 as a previous address. This means that every time a voter checks his vote confirmation receipt (as in section 3.3.5), the entire bulletin board history is validated.

We claim that our protocol achieves the *integrity* property through the bulletin board construction.

Verifiability: There are two 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 challenging the vote cryptogram as in section 3.3.4
- verify that the vote is registered as cast by checking the vote confirmation receipt as described in section 3.3.5

Some aspects of the election protocol are publicly verifiable:

- the distribution of the decryption key during the threshold ceremony
- the integrity of the bulletin board
- the correctness of the cleansing procedure
- the correctness of mixing and decryption phases, therefore verifying that all votes are counted as registered

One particular aspect is privately verifiable and available only to election officials. That is the eligibility verifiability of each submitted vote. The argument for it not being publicly available is that the verification is done based on some voter identification data, such as names or email addresses. Therefore, this is not publicly disclosed.

Being able to verify that a vote is cast as intended, registered as cast, and counted as registered, we claim that our election protocol is *end-to-end verifiable*.

Receipt-freeness: During the vote cryptogram generation process (described in section 3.3.3), the voter receives from the digital ballot box $\mathcal{D}$ a set of empty cryptograms $\vec{e}_{\text{d}}$ which are used to generate the voter's final vote cryptograms $\vec{e}$ that is an encryption of the vote $\vec{V}$. At the end of the process, each vote cryptogram $e_i \in \vec{e}$ would be equal to $\mathsf{Enc}(Y_{\text{enc}}, V_i, r_{\text{v};i} + r_{\text{d};i})$, where $V_i \in \vec{V}$. Note that $\vec{r}_{\text{d}} = \{r_{\text{d};1}, ..., r_{\text{d};\ell}\}$ is known by the digital ballot box $\mathcal{D}$ and $\vec{r}_{\text{v}} = \{r_{\text{v};1}, ..., r_{\text{v};\ell}\}$ is known by the voter.

After the encrypted ballot $\vec{e}$ has been published and cast on the bulletin board, the voter is not able to produce valid cryptographic evidence that $\vec{e}$ is an encryption of $\vec{V}$ referencing only the publicly available data, as the voting application does not know value $\vec{r}_{\text{d}}$. More details are described in appendix A.5.4.

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

4 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 4.1),
- internal auditing processes performed by election officials that target the behavior of the election system (section 4.2),
- publicly available audit processes that verify all data on the public bulletin board (section 4.3).

4.1 Voter-specific 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.

4.1.1 Vote is cast as intended

At the end of the vote cryptogram generation process (section 3.3.3), 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 a challenge of the vote cryptogram, as presented in section 3.3.4 to verify the activity of the voting application and digital ballot box.

If the voter chooses to challenge the cryptogram, 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 3.3.4.

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 audited the vote, it 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 a vote cryptogram (as presented in section 3.3.3) and choose again whether to challenge or submit. Voters should challenge again until they have enough confidence in the system to cast their vote as intended.

4.1.2 Vote is registered as cast

When the voter submits and casts an encrypted ballot (by submitting a cast request item as described in section 3.3.5), 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 26), 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 26) 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.

4.2 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.

4.2.1 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 can be done continuously throughout the election phase or as a final auditing step at the end of the election phase but before a result is computed.

Formally, the auditing starts by providing all eligible voter identities $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, ..., \mathcal{V}_{n_\mathrm{v}}\}$, the public key of the voter authorizer $Y_\mathcal{A}$ and the list of voter session items from the bulletin board $\{b_{\mathrm{vs};1}, ..., b_{\mathrm{vs};n_\mathrm{vs}}\}$, where n_vs is the total number of voter session items. Recall from section 2.4 that each item has the following structure $b_{vs;i} = (m_i, c_i, \mathcal{A}, \sigma_i, t_i, p_i, h'_i, h_i)$, with $i \in \{1, ..., n_\mathrm{vs}\}$.

The auditor checks that the voter authorizer has signed each item by running $\mathsf{SigVer}(Y_\mathcal{A}; \sigma_i, m_i||c_i||p_i)$ (algorithm 26) and its content relates to an eligible voter, i.e., $c_i = (\mathcal{V}_i, Y_i, H_i)$, with $\mathcal{V}_i \in \boldsymbol{\mathcal{V}}$. Then, the auditor verifies that all voter session items have been authorized after successful authentication, which is achieved differently, depending on the voter authentication mode (section 2.5).

When **credential-based** voter authentication mode is enabled, the auditor has also been provided with all voter authentication public keys $\{Y_{\mathrm{auth};1}, ..., Y_{\mathrm{auth};n_\mathrm{v}}\}$ for each of the voters in $\boldsymbol{\mathcal{V}}$. The voter authorizer provides all proofs of credentials $\{PK_1, ..., PK_{n_\mathrm{vs}}\}$ associated with each voter session item from the bulletin board. The auditor checks that $H_i = \mathcal{H}(PK_i)$ and $\mathsf{DLVer}(PK_i, \{G\}, \{Y_{\mathrm{auth};i}\})$ (algorithm 16), where H_i is the authentication fingerprint from the content of the voter session item $b_{\mathrm{vs};i}$ and $Y_{\mathrm{auth};i}$ 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 3.2.4 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.

When **identity-based** voter authentication mode is used, the auditor is provided instead, with the certificates of all identity providers $\boldsymbol{\mathcal{I}} = \{\mathcal{I}_1, ..., \mathcal{I}_{n_\mathrm{i}}\}$ including their public keys $\{Y_{\mathcal{I}_1}, ..., Y_{\mathcal{I}_{n_\mathrm{i}}}\}$ and identities for all voters in $\boldsymbol{\mathcal{V}}$ mapped to each identity provider in $\boldsymbol{\mathcal{I}}$. When being audited, the voter authorizer has to provide all identity tokens $\sigma_{\mathrm{id};i,j}$ generated by each identity provider $\mathcal{I}_j \in \boldsymbol{\mathcal{I}}$ used to create the voter session item $b_{\mathrm{vs};i}$. The auditor checks that the identity tokens are associated with the voter session item $H_i = \mathcal{H}(\sigma_{\mathrm{id};i,1}||...||\sigma_{\mathrm{id};i,n_\mathrm{i}})$, where H_i is the authentication fingerprint from the item content. Also, it checks the validity of the identity tokens by $\mathsf{SigVer}(Y_{\mathcal{I}_j}, \sigma_{\mathrm{id};i,j}, \mathcal{V}_i)$ (algorithm 26) and whether they are associated with an eligible voter, i.e., $\mathcal{V}_i \in \boldsymbol{\mathcal{V}}$. 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, this auditing step is available only to election officials.

4.3 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 4.3.1.
- After a result has been initiated (i.e., an extraction data item has been published as in section 3.4.1), anybody can verify that the cleansing procedure has been performed correctly, as explained in section 4.3.2.
- After a result has been computed and published, any public auditor can check the correctness of the result by verifying both the mixing process (see section 4.3.3) and the decryption process (see section 4.3.4).

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.

4.3.1 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 2.4) 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 B).

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 B, 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.

4.3.2 Verification of the cleansing 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 $\vec{e}_0$ present in the extraction data item. Recall from section 3.4.1 that $\vec{e}_0$ lists all votes that will make up the election result.

The auditor reruns the cleansing procedure on the bulletin board $\boldsymbol{b}$, applying all the filtering rules specified in section 3.4.1 to recompute the *initial mixed board* $\vec{e}_0'$. If $\vec{e}_0'$ is identical with $\vec{e}_0$, then the cleansing procedure has been performed correctly.

4.3.3 Verification of mixing procedure

After a result has been published via an extraction confirmation item (as described in section 3.4.4), any public auditor can verify the mixing procedure of each trustee that participated in the mixing phase. This checks that no votes have been tampered with during mixing. The data that an auditor needs to collect comes from the following sources:

- the initial mixed board $\vec{e}_0$ is listed in the extraction data item,
- the encryption keys Y_{enc} and the list of trustees $\boldsymbol{\mathcal{T}} = \{\mathcal{T}_1, ..., \mathcal{T}_{n_\text{t}}\}$ 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 $\tau \subset \{1, ..., n_\text{t}\}$, together with all the intermediate mixed boards that they produced $\vec{e}_i$, with $i \in \tau$, and their respective proofs of correct mixing (PM_i, AS_i) are included in the extraction confirmation item.

The auditor validates each intermediate mixed board by running algorithm 34 $\mathsf{MixVer}(PM_i, AS_i, Y_{\text{enc}}, \vec{e}_{i-1}, \vec{e}_i)$, for each $i \in \tau$. If any validations fail, the published result is not trustworthy.

4.3.4 Verification of the decryption

After a result has been published via an extraction confirmation item (as described in section 3.4.4), any public auditor can verify the decryption procedure by checking all the partial decryptions of each trustee that participated in the decryption phase. The data that an auditor needs to collect comes from the following sources:

- the list of all trustees $\boldsymbol{\mathcal{T}} = \{\mathcal{T}_1, ..., \mathcal{T}_{n_\mathrm{t}}\}$, together with their public keys $\{Y_{\mathcal{T}_1}, ..., Y_{\mathcal{T}_{n_\mathrm{t}}}\}$ and public polynomial coefficients $\{P_{\mathcal{T}_1,1}, ..., P_{\mathcal{T}_{n_\mathrm{t}},t-1}\}$, are listed in the threshold configuration item, where t is the threshold value set during the threshold ceremony (see section 3.2.5), and n_t is the total number of trustees,
- the final mixed board $\vec{\vec{e}}$ is listed in the extraction confirmation item,
- the subset of trustees that participated in the mixing and decryption phases $\tau \subset \{1, ..., n_\mathrm{t}\}$, together with all their partial decryption $\vec{\vec{S}}_i$, with $i \in \tau$, and their respective proofs of correct decryption PK_i are also included in the extraction confirmation item.

The auditor validates the proof of each partial decryption by running algorithm 7 $\mathsf{PartialDecryptionVer}(\vec{\vec{e}}, \vec{\vec{S}}_i, PK_i, sY_i)$, for each $i \in \tau$, where sY_i is the public share of the trustee $\mathcal{T}_i$ computed as in appendix A.5.3.

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 published result. If any validation step fails, the result is considered untrustworthy.

Counting the votes and sorting the candidates based on their vote count is considered trivial and out of scope.

5 Adversary model

This section describes the capabilities and limitations of an attacker that the system is designed to protect against. Then, it exemplifies some attack vectors and describes how the system defends against them.

Some scenarios present the way the system detects malicious activity rather than preventing it from happening. In that situation, it is up to election officials to assess the situation and make a recovery decision.

5.1 Attacker types

The election system is designed to protect against the following types of attackers. We categorize attacker types based on what they can control and what private information they can access.

5.1.1 Malicious users

The first category describes users that behave maliciously or have been compromised and impersonated by malicious actors. Therefore, we consider each stakeholder listed in section 4.1 potentially malicious.

A malicious election official can try to alter the election configuration in any imaginable way, either for destructive purposes or for modifying a candidate's appearance. A malicious election official can also try to gain prohibited information, such as a partial result.

A malicious voter can try to cast multiple ballots that get counted in the tally. A voter can also try to disrupt an election by submitting an invalid vote. Additionally, a voter can try to get cryptographic evidence that, at a later point, will convince a third party of the content of the previously submitted vote.

A malicious trustee may try to gain sensitive information, such as compute a partial result, read a particular voter's vote, or even compute the main decryption key. As a destructive action, a malicious trustee can try to prevent a result from being calculated by refusing to participate in the post-election phase protocol.

A malicious auditor may falsely claim the status of an election. For example, a malicious auditor might try to convince the public that the integrity property of an election is broken when it is not, or the other way around.

5.1.2 Compromised componenets

The second category of attackers relates to system components that get compromised due to successful hacking attempts or malicious system administrators.

A compromised trustee application can leak all the private data that belongs to that particular trustee. This includes the trustee's share of the decryption and mixing coefficients. It can also tamper with the output of the protocol it is supposed to perform.

A compromised voting application can tamper with the output of the protocol it is supposed to perform.

A compromised credentials authority can leak all voter credentials generated in section 3.2.4. As a disruptive action, it can distribute wrong credentials to the voters, i.e., credentials that do not correspond to the public authentication keys sent to the voter authorizer, as in section 3.2.4.

A compromised identity provider can generate fraudulent identity tokens for voters, i.e., without successful authentication.

Compromised voter authorizer can leak its internal secrets including its private key $x_{\mathcal{A}}$. Can generate fraudulent voter session items on the bulletin board granting unauthorized voting access.

Compromised external verifier can display fake values to its user during the protocol described in section 3.3.4.

5.1.3 Compromised communication channels

The last category of attackers describes breaches that happen to communication channels. Recall from section 2.3 that an authentic channel can leak the data it transports, while a public channel can leak and even allow tampering with the data being transported.

The following list contains all the authentic communication channels that are used throughout the protocol:

- the channel used between the election administration service and all the other services that need to get their public key authorized for their role, as described in section 3.2.2.

The following list contains all the public communication channels that are used throughout the protocol:

- the channels used by the election administration service and the voter authorizer to communicate with the digital ballot box
- the channel used by the voting application to the digital ballot box during the vote cryptogram generation process section 3.3.3
- the channel used by the voting application to the voter authorizer during the voter authorization procedure section 3.3.1

- the channel used between the trustee application and the election administration service in the threshold ceremony (section 3.2.5) and post-election phase (section 3.4)

5.2 Assumptions

Our election protocol assumes the following limitations about a potential attacker. The first category describes assumptions necessary for the election system to work correctly. The second category describes the assumptions required to fulfill the security properties described in section 3.5.

Assumptions related to the well-functioning of the election protocol:

- An attacker's computation power is assumed to be polynomially bound.
- The elliptic curve discrete logarithm problem is assumed to be infeasible to break, as described in appendix A.2.2.
- All the private communication channels are assumed to be secret and tamper-resistant. The full list of private communication channels used in the protocol consists of the following:
  - generally, users are assumed to interact genuinely with their devices,
  - the voting application assumes to receive correct inputs from voters,
  - voters are not observed while interacting with their devices,
  - the trustee application is assumed to correctly and secretly receive trustee inputs, including the share of the decryption key in the post-election phase section 3.4,
  - election officials can interact genuinely and privately with a browser to access the election administration service and voter authorizer,
  - election officials can interact genuinely and privately with the auditing scripts during the administration auditing process (section 4.2),
  - credential authorities are assumed to secretly and correctly receive voter contact information from an election official during the voter credential distribution process (section 3.2.4),
  - credential authorities are assumed to secretly and correctly distribute voter credentials to the voters during the voter credential distribution process (section 3.2.4),
  - the voting application is assumed to genuinely and privately interact with all third-party identity providers.
- All the authentic communication channels are assumed to be tamper-resistant. The full list of authentic communication channels used in the protocol consists of the following:

  - the channel used between the election administration service and all the other services that need to get their public key authorized for their role, as described in section 3.2.2.

Assumptions related to security properties:

- For achieving the eligibility property, we assume:
  - when voter authorization mode is **credential-based**, there is at least one honest credential authority that generates and distributes correct voter credentials,
  - when voter authorization mode is **identity-based**, there is at least one honest third-party identity provider that generates genuine identity tokens on successful voter authentication,
  - the administration auditing process (section 4.2) is trustworthy, i.e., an honest election official runs genuine auditing tools against real election data.
- For achieving the privacy and anonymity properties, we assume:
  - there are no more than t malicious trustees or compromised trustee applications, where t is the decryption threshold configured during the threshold ceremony (section 3.2.5),
  - the voting application does not leak voter secret information.
- For achieving verifiability, we assume:
  - A voter uses at least one honest 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.
- For preserving the integrity, we assume that the integrity audit (section 4.3.1) is trustworthy, i.e., an honest election official runs genuine auditing tools against real election data.

5.3 Recoverable attack scenarios

Here, we present an extensive list of attacks that we considered and might happen to an election. We also describe how the attack is detected and how the system responds. On the other hand, in section 5.4, we describe some attacks that the protocol only provides a way to detect. If they successfully happen, it is up to election officials to assess the damage and decide on a resolution, e.g., repeat the election event.

A corrupt election official maliciously changes the configuration
We consider the case when an election official decides to manipulate the election by changing the configuration in an unauthorized way. The attack can happen for various reasons, such as the stakeholder being malicious or the user's account being compromised and controlled by an external attacker.

Recall from section 3.2.2 that all configuration updates are documented on the bulletin board as configuration items (i.e., election configuration items, contest configuration items, etc.). Therefore, a vigilant honest election official can observe and detect suspicious changes to the election configuration that have not been agreed upon.

In response, the honest election official can change the configuration back to the correct one. If voters have voted according to the incorrect configuration, they should be contacted and asked to vote again. Further investigation can reveal what user account triggered the malicious configuration changes, and relevant actions should be taken.

A corrupt election official attempts to read an unauthorized result
A corrupt election official has the ability to request a result to be computed at any time. That requires altering the configuration to end the election phase, then posting an extraction intent item on the bulletin board.

For this result to be computed, at least a threshold of trustees must collaborate in the mixing and decryption ceremonies (section 3.4). Recall from section 5.2 that there are less than a threshold of corrupt trustees who are considered to be willing to participate in an unauthorized result computation. Therefore, a corrupt election official cannot read an unauthorized result, even if colluding with corrupt trustees.

A malicious voter submits multiple ballots; ballot stuffing attempt
Voters are allowed to cast a ballot multiple times. Each action of casting a ballot is done through a separate voter session which the voter authorizer introduces based on distinct vote authentications. When an election official requests a result calculation (section 3.4), the digital ballot box filters the entire bulletin board and includes in the list of extracted ballots only the last cast ballot per voter identifier, as described in section 3.4.1. Therefore, the system decrypts and tallies no more than one ballot per voter.

A corrupt trustee attempts to decrypt the vote of a particular voter
Trustees are the stakeholders that hold and control the shares of the ballot decryption key. Hence, we consider the case that a corrupt trustee decides to decrypt a particular vote from the bulletin board as an attempt to break anonymity. Even though the mixing phase can be skipped for such a scenario, the decryption ceremony (section 3.4.3) is essential. Holding a decryption ceremony, which is not even documented on the bulletin board by an extraction intent item, resembles an attempt to compute an unauthorized result, covered

in an attack scenario described above. On the same assumption that there are less than a threshold of corrupt trustees, it is infeasible for a targeted ballot to be decrypted.

A corrupt trustee attempts to prevent a result from being computed
A result computation depends on the decryption phase, which depends on at least a threshold of honest trustees to collaborate. This dependency is met because of the assumption that there are at least a threshold of honest trustees. Therefore, an authentic election result is computable, given that enough honest trustees are functional.

A dishonest auditor spreads deceiving information about the integrity status of the election
Any individual can claim any property or characteristic about an election running on our protocol. The essential part is to prove any of the claims correct or false. To prove the integrity of an election, one must present a bulletin board trace (list of board items) that is valid according to validations from section 4.3.1, and that is initiated by an expected genesis item (i.e., the genesis item has a specific address). Note that a proof of election integrity is bound to the extent of the bulletin board trace, i.e., to prove the integrity of an entire election process, one must provide a bulletin board trace that contains all election events, from the genesis item to a result being computed and published.

Another aspect of verifying election integrity is the legitimacy of the auditing software used to validate a bulletin board trace. Theoretically, one can run all mathematical calculations by hand, but practically, that is outrageously infeasible. An auditor must rely on some software or scripts to run the auditing process. Therefore, assuming possession of genuine auditing scripts and authentic input data, an auditor is convinced about the integrity of an election, that being the case.

A trustee gets compromised, the share of the decryption key and the mixing coefficients get leaked
Regardless of whether a trustee turns malicious or gets compromised, the share of the decryption key and the mixing coefficients (as generated in section 3.4.2) of a trustee might get leaked. One share of the decryption key alone is useless. One set of mixing coefficients can prove the connection of cryptograms between two consecutive mixed boards of the mixing process but does not reveal the full connection of ballots between the initial mixed board to the final one that gets decrypted.

The key objective is to not leak too much information such that the anonymity of an election is broken. The main decryption key cannot reasonably be computed on the assumption that no more than a threshold of trustees can get compromised. Moreover, all mixing coefficients used during mixing cannot be collected. Therefore anonymity is preserved.

A forged trustee application is used that defies the protocol
This scenario can happen due to a malicious trustee running unauthorized software or a compromised trustee machine that replaced the genuine software. Regardless of the reason, the unauthorized software is involved in the threshold ceremony (section 3.2.5) during the pre-election phase and in the mixing and decryption processes (sections 3.4.2 and 3.4.3) during the post-election phase.

Assuming a trustee application misbehaves and delivers incorrect data during the threshold ceremony, the malicious activity is exposed at the end of the ceremony when all trustees validate that their partial share of the decryption key has been computed correctly. The process is better explained in appendix A.5.3. When exposed, the election official organizing the threshold ceremony must decide whether to redo the process or exclude that trustee and continue without.

Considering the case of a misbehaving trustee application during the mixing phase (i.e., delivering faulty mixed board of cryptograms or proof of mixing), the election administration service rejects the request of submitting a mixed board because of the invalid proof of mixing. In this case, it is the decision of the election official that organizes the ceremony whether to retry the mixing process of that trustee or consider that trustee corrupt and excluded from the ceremony.

When a trustee application misbehaves during the decryption phase (i.e., delivering a faulty partial decryption or proof of correct decryption), the election administration service rejects the partial decryption because of the invalid proof of correct decryption. Again, the election official decides whether to let that trustee retry the decryption process or consider that trustee corrupt and excluded from the ceremony.

A forged voting application is used that defies the protocol This scenario can occur because of a malicious voter trying to disrupt the process or a voter using a compromised device that has the genuine voting application replaced with malicious software. The voting application is involved in the voter authorization procedure (section 3.3.1), vote cryptogram generation process (section 3.3.3), the process of challenging a vote cryptogram (section 3.3.4) and the process of confirming a vote receipt (section 3.3.5).

First, we consider a misbehaving voting application during the voter authorization procedure. Regardless of the authentication mode (**credential-based** or **identity-based**), the first thing a malicious application could do is block the voter authorization by sending incorrect authentication data to the voter authorizer (i.e., tampering with the identity tokens in the **identity-based** mode or submit a broken proof of credentials in the **credential-based** mode). The voter is informed that the authorization failed. In response, the voter should close the application and retry the authorization procedure from a different device.

A second attack the voting application could do is to steal the genuine authentication data, use it to get a valid voter session, and vote on that voter's behalf, all while informing the voter that the authorization failed. Again, the voter should close the application and retry the authorization process from a different device. If the second device behaves correctly, the voter session and all voting data it generates replace the forged data published by the corrupt application.

Note that a vote cannot differentiate between using a compromised application that prevents the authorization and actually using the wrong credentials. Therefore, it is a human decision to evaluate how many times to retry the authorization process before seeking a different kind of support.

Next, we consider the voting application misbehaving during the vote cryptogram generation process (i.e., the protocols from figures 5 and 6 between the voting application and the digital ballot box). Verification mechanisms are built into the digital ballot box for each interaction triggered by the voting application. Eventually, the voting application presents the voter with a value that should uniquely identify the encrypted ballot on the bulletin board (i.e., the address of the verification track start item). The voter verifies the activity of the voting application by using an external verifier to query the bulletin board for the encrypted ballot.

Finally, the voting application might misbehave while handling the vote receipt. If the voting application presents a fabricated vote receipt to the voter, the external verifier will label the receipt as invalid. Suppose the voting application claims the ballot has not been cast. In that case, the external verifier can query the bulletin board for the actual ballot status by the address of the verification track start item.

A forged voting application is used that changes the voter's vote
This classical attack scenario can happen if a voter uses a compromised device with a malicious voting application running. Basically, during the vote cryptogram generation process, the application replaces the plain-text vote with a different one. The way a voter can verify the behavior of the voting application is to perform the process of challenging a vote cryptogram, as described in section 3.3.4. If the voting application is malicious, then the external verifier is assumed to be trustworthy, therefore, through the challenging process, it presents the voter with the actual vote encoded in the encrypted ballot.

Note that if there are discrepancies between the voter's intention and the output of the external verifier, there is no way of differentiating between a malicious voting application and a mistype of the voter. Therefore, it is a human decision to evaluate how many times to retry the vote challenging process before declaring the voting application malicious.

A forged external verifier is used that defies the protocol
This attack scenario can happen because one of the multiple external verifier deployments is malicious or because the voter uses a compromised device that runs unauthorized software. The external verifier is actively involved in the

process of challenging a vote cryptogram. Suppose it presents incorrect data to mislead the voter into believing the vote has been incorrectly encoded. In that case, the voter is encouraged to retry the vote challenging process even by interacting with a different external verifier. Eventually, an honest external verifier will confirm the legitimacy of the voter's ballot.

A compromised credentials authority leaks all its voter credentials
This scenario is relevant when voter authentication mode is **credentials-based**.

Leaking all voter credentials generated by a credentials authority means that the public knows the information that only that credentials authority had. The other credentials authorities also gain this information. That means voters can still authenticate as long as there is at least one honest credential authority. In this case, that authority is trusted not to authenticate on behalf of voters, even if it can do so.

Otherwise, as long as there are at least two sets of secret credentials, only the voters are in possession of all their credentials. Therefore only the actual voters can successfully authenticate, get authorized, and cast a ballot.

A malicious credentials authority distributes wrong voter credentials
This scenario is relevant when voter authentication mode is **credentials-based**.

This attack can happen because of a malicious credentials authority willing to disrupt the election process by attempting to block the voter authorization process. Another reason for distributing wrong voter credentials could be a human accident, a bug in the source code, or a distribution error that is not necessarily intentional.

Regardless of the reason, the problem can be detected rapidly by all voters not being able to authenticate. Once detected, the problem can be fixed by reconfiguring the voter authorizer to discard the voter authentication public keys received from the problematic credential authority. The voter authentication process can recover by making the voters authenticate with one less credential. If a single active credentials authority is left in the process, it must be considered trustworthy.

A compromised third-party identity provider maliciously generates identity tokens for any voter identity
This scenario is relevant when voter authentication mode is **identity-based**.

This attack requires a third-party identity provider to become compromised so an attacker can generate identity tokens for any voter identity. Such an attack can be detected by that third-party provider, and then the voter authorization configuration can be updated to not rely on that identity provider any longer for voter authentication. The voter authorization process can continue with one less identity provider. If a single active identity provider is left in the process, it must be considered trustworthy.

The voter authorizer gets compromised, its private key gets leaked, or it authorizes voters without a successful authentication
This scenario can happen due to the voter authorizer service being compromised by an attacker by a malicious system administrator leaking the private key. To detect whether a key got compromised, an auditor can verify the activity of the voter authorizer and check whether the voter authorizations it performed are based on successful voter authentication.

When the voter authorizer is considered compromised, an election official can deauthorize that component and set up a new voter authorizer service by posting an actor configuration item with the new voter authorizer setup, including the new public key. From then on, the new voter authorizer will perform all voter authorizations.

An attacker eavesdrops on the communication between the election administrator and the voter authorizer
This scenario describes the fact that the data exchanged between the election administrator service and the other services (i.e., the voter authorizer) during the process of setting up actors on the bulletin board (section 3.2.2) can be read by an attacker. This data consists of public information that will eventually end up on the bulletin board, such as the public key of the voter authorizer. Therefore, listening in on this communication channel is harmless.

An attacker eavesdrops on the data of the configuration processes
This scenario describes the fact that the data exchanged between the election administrator service and the digital ballot box during the election configuration process (section 3.2.2) can be read by an attacker. This is public information that is supposed to be accessible through the bulletin board. Therefore, listening in on this communication channel is harmless.

An attacker tampers with the data of the configuration processes
This scenario describes an attempt by an external attacker to modify the election configuration by tampering with the data in transit. Recall from section 3.2.2 that the configuration is published on the bulletin board in the form of bulletin board items written by relevant actors, i.e., the election administrator service or the voter authorizer. Basically, the attacker would tamper with the data from protocol 1. The data in both the request and the response in protocol 1 is accompanied by a digital signature generated by the appropriate actor, i.e., the request is signed by the owner of the board item, while the response is signed by the digital ballot box. In consequence, tampering with the data in transit would invalidate the request and trigger a retry.

An attacker eavesdrops on the data of the voter authorization process
This scenario describes the fact that the data exchanged between the voter authorizer service and the digital ballot box during the voter authorization

procedure (section 3.3.1) can be read by an attacker. This is public information that is supposed to be accessible through the bulletin board. Therefore, listening in on this communication channel is harmless.

An attacker tampers with the data of the voter authorization process
This scenario describes an attempt of an external attacker to modify the configuration of a voter session by tampering with the data from protocol 1. Recall from section 3.3.1 that the voter data is published on the bulletin board as a voter session item written by the voter authorizer. The data in both the request and the response in protocol 1 is accompanied by a digital signature generated by the appropriate actor, i.e., the request is signed by the voter authorizer, while the response is signed by the digital ballot box. In consequence, tampering with the data in transit would invalidate the request and trigger a retry.

An attacker eavesdrops on the data of the voting process
This scenario describes the fact that the data exchanged during the vote cryptogram generation process (section 3.3.3) can be read by an attacker. We have already discussed that data transferred during the protocol 1 can be publicly readable. Here we discuss the implication of leaking the extra data transferred between the voting application and the digital ballot box.

During the protocol in figure 5, the response of the digital ballot box contains the empty cryptograms used in the computation of the voter's encrypted ballot. If a third-party actor can read and trust the authenticity of the empty cryptograms, combined with a voter willing to prove the vote content, the receipt freeness property of the election is broken on that particular ballot.

During the protocol in figure 6, the request of the voting application contains the proof of correct encryption, which is used to validate the request. Reading this proof is harmless, as it gives no advantage to an attacker.

An attacker tampers with the data during the voting process
This scenario describes an attempt of an external attacker to interfere with the data during the vote cryptogram generation process section 3.3.3. We have already discussed that tampering with the data during the protocol 1 results in rejecting the request. Here we cover the case of tampering with the extra data that is transferred between the voting application and the digital ballot box.

During the protocol in figure 5, the digital ballot box responds with the empty cryptograms used to encrypt the voter's ballot. Suppose a third-party attacker tampers with the empty cryptograms. In that case, a voter can detect the discrepancy by performing the process of challenging a vote cryptogram (section 3.3.4). During the challenging process, the voting application can verify whether the empty cryptograms are according to the encryption commitment published by the digital ballot box, therefore checking whether any interference happened in the transfer.

During the protocol in figure 6, the request of the voting application contains the proof of correct encryption used to validate the request. Tampering with the data in transit would invalidate the request and trigger a retry.

An attacker eavesdrops on the data of the threshold ceremony and result ceremony
Data that is generated and transferred during the threshold ceremony (section 3.2.5) and during the result ceremony (sections 3.4.2 and 3.4.3) is meant to be published on the bulletin board, either as the threshold configuration item (for the threshold ceremony data) or as the extraction confirmation item (for the mixing and partial decryption data). Therefore, eavesdropping on the threshold ceremony and result ceremony is harmless.

There is one exception regarding the data being published on the bulletin board. That is the encrypted partial secret shares sent by the trustees to the election administration service during the threshold ceremony. Recall from appendix A.5.3 and figure 12 that each trustee computes partial secret shares for the other trustees and then privately distributes them to the other trustees. That is achieved by encrypting and delivering them to the election administration service, where the other trustees can fetch them from. These partial secret shares are stored and transferred in encrypted form, therefore, eavesdropping on them is harmless.

An attacker tampers with the data of the threshold ceremony and result ceremony
This scenario resembles the case of a compromised trustee application that delivers incorrect data during the threshold ceremony or mixing and decryption phases. If the validations fail, there is actually no way for the election administration service to distinguish whether the trustee has wrongly computed the data or it has been tampered with in transfer.

Nevertheless, tampering with the data during the threshold ceremony leads to trustees failing to validate their partial secret share, therefore exposing the incorrect data. Tampering with the data during the mixing or decryption phases leads to the election administration service rejecting the request to submit a mixed board or a partial decryption. In that case, it is up to the election official that organizes the ceremony to redo the process or to exclude that trustee from the process.

5.4 Disruptive attack scenarios

This section describes some attacks that the protocol does not protect against. The protocol only supports detection mechanisms for them. If such an attack is detected, we recommend restoring the setup with new, healthy components, including the digital ballot box, and repeating the election event, if possible.

The election administrator service gets compromised, and its private key leaked
We present the scenario where the entire election administrator service gets compromised, therefore, no honest election official can access the system. In other words, the election configuration and management functionalities fall under the attacker's control.

The attack is detectable by election officials being unable to push election configuration updates or noticing unauthorized configuration published on the bulletin board. In such cases, the election data is considered unreliable as it is generated from unreliable sources.

A portion of the bulletin board considered authentic (i.e., up to the unreliable items published by the compromised election administrator service) can be used to extract a result in an offline result ceremony. Basically, trustees have to collaborate in the processes described in section 3.4, while offline scripts must be used to simulate the actions of the election administration service and the digital ballot box. Such a result loses the public verifiability property.

The digital ballot box gets compromised, and its private key leaked
We consider the case where the digital ballot box gets compromised. Therefore, all functionalities based on using the private key of the digital ballot box are considered untrustworthy. They include the generation of bulletin board items that facilitate the voting process described in section 3.3, and the computation of receipts returned during the protocol 1, which is a core building block of the entire election protocol.

The attack is detectable by observing deviations from the protocol made by the digital ballot box in terms of items written on the bulletin board. Another detection mechanism is the fact that other system components receive invalid receipts during their interactions with the digital ballot box. Another sign that the private key of the digital ballot box has been compromised is by someone owning a valid receipt of an item that is actually not on the bulletin board.

The bulletin board gets tampered with
Here, we describe a scenario where the bulletin board gets tampered with. This can happen because of the digital ballot box getting compromised through an external attack, a malicious system administrator, or even a bug in the system.

Tampering with the bulletin board implies that some items on the board are modified or removed, thus breaking the history property of the public bulletin board construction, described in section 2.4. The attack is detectable by an auditor running the integrity audit process, as described in section 4.3.1. The fraction of the bulletin board that has integrity preserved (i.e., from the genesis item up to the first tampered-with item) could be used to extract a result in an offline ceremony, as described above. However, such a result loses the public verifiability property.

5.5 Conclusion

We have described the threat model of our end-to-end verifiable election protocol and some potential attack scenarios that the system is designed to protect against. We have also listed some attacks that the system can only detect but not completely prevent from happening. We assume an attacker has a high level of technical knowledge but cannot break the cryptographic primitives used throughout the protocol. Some functionalities and responsibilities in the protocol are split amongst several entities in a threshold manner. In such cases, the attacker is assumed to control a limited amount of those entities.

The proposed election protocol provides security guarantees against the presented adversary model. The protocol offers, on one hand, integrity and privacy of the election data and, on the other hand, eligibility and anonymity for the voters. The protocol is end-to-end verifiable, combining individual and universal verification steps. The protocol ensures that no one, including insider and external attackers, can tamper with election data undetected.

However, it is essential to state that the protocol cannot defend against an attacker with close to infinite computation power or an attacker that can break the cryptographic primitives. We doubt such an attacker currently exists, but we are sure that such capabilities might arise in the near future. Therefore we are working on a future protocol version that relies on fewer assumptions about the adversary's capabilities. Moreover, we are aware that the protocol does not provide ever-lasting privacy. That means an attacker controlling more than a threshold of trustees can break anonymity and read sensitive information. This is another research area we are currently investigating to improve.

Overall, our proposed cryptographic protocol represents a strong foundation for building a secure, end-to-end verifiable digital voting solution.

References

[1] Tusk Philanthropies. Mobile voting project. `https://mobilevoting.org`.

[2] James Heather and David Lundin. The append-only web bulletin board. In Pierpaolo Degano, Joshua Guttman, and Fabio Martinelli, editors, *Formal Aspects in Security and Trust*, pages 242–256, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[3] Josh Benaloh. Simple verifiable elections. In *Proceedings of the USENIX Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, EVT'06, pages 5–5, Berkeley, CA, USA, 2006. USENIX Association.

[4] René Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, 44(170):483–494, 1985.

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

[6] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[7] Sherman S. M. Chow, Changshe Ma, and Jian Weng. Zero-knowledge argument for simultaneous discrete logarithms. In My T. Thai and Sartaj Sahni, editors, *Computing and Combinatorics*, pages 520–529, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[8] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In Donald W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, pages 522–526, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[9] Yvo G. Desmedt and Yair Frankel. Threshold cryptosystems. In *Proceedings on Advances in Cryptology*, CRYPTO '89, pages 307–315, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[10] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[11] National Institute of Standards and Technology. Advanced encryption standard. *NIST FIPS PUB 197*, 2001.

[12] C. P. Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York.

[13] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryp-*

tology — CRYPTO '91, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

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

[15] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. *IACR Cryptol. ePrint Arch.*, page 246, 2005.

[16] Kathleen Moriarty, Burt Kaliski, and Andreas Rusch. PKCS #5: Password Based Cryptography Specification Version 2.1. RFC 8018, January 2017.

[17] OWASP (Open Worldwide Application Security Project) Foundation. OWASP cheat sheet series. `https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#pbkdf2`, 2023.

[18] Dr. Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, May 2010.

A Theoretical background

A.1 Mathematics

A.1.1 Group

In mathematics, a group $\mathcal{G} = (\mathbb{G}, \circ, \text{inv}, e)$ is an algebraic structure consisting of a set $\mathbb{G}$ of elements, a binary operation indicated by the symbol $\circ$, a unary operation called **inv** and a neutral element $e \in \mathbb{G}$. The following properties must be satisfied by $\mathcal{G}$:

closure	$x \circ y \in \mathbb{G}$
associativity	$x \circ (y \circ z) = (x \circ y) \circ z$
identity element	$x \circ e = e \circ x = x$
inverse element	$x \circ \mathbf{inv}(x) = e$

for all $x, y, z \in \mathbb{G}$.

If $\mathcal{G}$ has a fifth property called *commutativity* (i.e. $x \circ y = y \circ x$), then $\mathcal{G}$ is an *abelian group*.

Moreover, if $\mathcal{G}$ is a *finite group*, then $\mathbb{G}$ has a finite number of elements, and we denote $q = |\mathbb{G}|$ as the order of the group. For example, a finite group would be $(\mathbb{Z}_q, +, -, 0)$, where $\mathbb{Z}_q = \{0, 1, ..., q-1\}$, the binary operation is addition modulo q, the inverse operation is negation, and the identity element is 0.

The binary operation can be applied on the same element, namely $x \circ x = [2]x$. We define $[k]x$ as the operation $\circ$ applied k times on the element x.

A finite group $\mathcal{G} = (\mathbb{G}, \circ, \text{inv}, e)$ of order q is called *cyclic group*, if there is a group element $g \in \mathbb{G}$, such that $\mathbb{G} = \{g, [2]g, [3]g, ..., [q]g\}$. In this case, the element g is called the *generator* of $\mathcal{G}$.

A.1.2 Finite Field

A field $\mathcal{F} = (\mathbb{F}, +, \cdot)$ consists of a set $\mathbb{F}$, which is an abelian group in respect to both operations: addition and multiplication. The following properties hold:

- $x + y \in \mathbb{F}$ and $x \cdot y \in \mathbb{F}$
- $(\mathbb{F}, +, -, 0)$ is an abelian group
- $(\mathbb{F}^*, \cdot, ^{-1}, 1)$ is an abelian group
- multiplication is distributive over addition: $x \cdot (y + z) = x \cdot y + x \cdot z$

for all $x, y, z \in \mathbb{F}$.

A finite field is a field with a finite number of elements, for example, the set of integers modulo p, denoted $\mathbb{F}_p$, where p is a prime number.

A.2 Elliptic Curve

A.2.1 Elliptic Curve over a Prime Field

We define the elliptic curve E over the prime field $\mathbb{F}_p$ as the set of points

$$E(\mathbb{F}_p) = \{(x, y) \in (\mathbb{F}_p)^2 \mid y^2 = x^3 + ax + b \pmod{p}\} \cup \{\mathcal{O}\}$$

where a tuple (x, y) represent the coordinates of a point, $\mathcal{O}$ is the point at infinity and $a, b \in \mathbb{F}_p$.

The elliptic curve $E(\mathbb{F}_p)$ follows a group structure with the following rules:

- $\mathcal{O}$ is the identity element, thus $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E(\mathbb{F}_p)$.
- The inverse operation is point negation noted $-$. For all $P = (P_x, P_y) \in E(\mathbb{F}_p)$, we define $-P = (P_x, -P_y)$ such that $P + (-P) = \mathcal{O}$.
- The binary operation is point addition noted $+$. Let $P, Q \in E(\mathbb{F}_p)$. The line through P and Q intersects the elliptic curve in a third point $R = (R_x, R_y) \in E(\mathbb{F}_p)$. The point addition is defined as $P + Q = -R$. The coordinates of R can be computed in the following way:

$$\begin{aligned} R_x &= \lambda^2 - P_x - Q_x \pmod{p} \\ R_y &= P_y + \lambda \cdot (R_x - P_x) \pmod{p} \end{aligned}$$

  where λ is the steep of line PQ. The steep can be computed in the following way:

$$\lambda = \begin{cases} (P_y - Q_y) \cdot (P_x - Q_x)^{-1} \pmod{p} & \text{, if } P \neq Q \\ (3 \cdot P_x^2 + a) \cdot (2 \cdot P_y)^{-1} \pmod{p} & \text{, if } P = Q \end{cases}$$

We define the total number of points on the $E(\mathbb{F}_p)$ as N, which can be calculated using *Schoof's algorithm* [4]. Any subgroup of $E(\mathbb{F}_p)$ has an order q, which is a divisor of N. In such a case, we define the *cofactor* of the subgroup as $h = \frac{N}{q}$. To find any generator of the subgroup, we perform the following:

- choose a random point $P \in E(\mathbb{F}_p)$,
- compute $G = [h]P$,
- if $G = \mathcal{O}$, repeat the process. Otherwise, G is a generator.

In conclusion, we can define our cryptographic cyclic subgroup as:

$$\mathbb{P} = \{P \in E(\mathbb{F}_p) \mid P = [k]G, k \in \mathbb{Z}_q\}$$

where G is the generator and q is the order of the subgroup. We call the integer k a *scalar*.

A.2.2 Elliptic Curve Discrete Logarithm Problem

The *Elliptic Curve Discrete Logarithm Problem* is defined in [5] the following way: Given the elliptic curve domain parameters (p, a, b, G, q, h) and a point $P \in \mathbb{P}$, find the scalar $k \in \mathbb{Z}_p$ such that $P = [k]G$. For an elliptic curve to be cryptographically strong, the ECDLP has to be *computationally infeasible*.

A.2.3 Supported Elliptic Curves

There is support for the following elliptic curves:

Secp256k1(Bitcoin Curve):

p	`ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffffe fffffc2f`
a	`00`
b	`07`
G	`02 79be667e f9dcbbac 55a06295 ce870b07 029bfcdb 2dce28d9 59f2815b 16f81798`
q	`ffffffff ffffffff ffffffff fffffffe baaedce6 af48a03b bfd25e8c d0364141`
h	`1`

Secp256r1(NIST P-256):

p	`ffffffff 00000001 00000000 00000000 00000000 ffffffff ffffffff ffffffff`
a	`ffffffff 00000001 00000000 00000000 00000000 ffffffff ffffffff fffffffc`
b	`5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b`
G	`03 6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296`
q	`ffffffff 00000000 ffffffff ffffffff bce6faad a7179e84 f3b9cac2 fc632551`
h	`1`

Secp384r1(NIST P-384):

p	ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffffe ffffffff 00000000 00000000 ffffffff
a	ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffffe ffffffff 00000000 00000000 fffffffc
b	b3312fa7 e23ee7e4 988e056b e3f82d19 181d9c6e fe814112 0314088f 5013875a c656398d 8a2ed19d 2a85c8ed d3ec2aef
G	03 aa87ca22 be8b0537 8eb1c71e f320ad74 6e1d3b62 8ba79b98 59f741e0 82542a38 5502f25d bf55296c 3a545e38 72760ab7
q	ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff c7634d81 f4372ddf 581a0db2 48b0a77a ecec196a ccc52973
h	1

Secp521r1(NIST P-521):

p	01ff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
a	01ff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffffc
b	51 953eb961 8e1c9a1f 929a21a0 b68540ee a2da725b 99b315f3 b8b48991 8ef109e1 56193951 ec7e937b 1652c0bd 3bb1bf07 3573df88 3d2c34f1 ef451fd4 6b503f00
G	0200c6 858e06b7 0404e9cd 9e3ecb66 2395b442 9c648139 053fb521 f828af60 6b4d3dba a14b5e77 efe75928 fe1dc127 a2ffa8de 3348b3c1 856a429b f97e7e31 c2e5bd66
q	01ff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffffa 51868783 bf2f966b 7fcc0148 f709a5d0 3bb5c9b8 899c47ae bb6fb71e 91386409
h	1

A.2.4 Elliptic Curve Point Encoding

Each point on the curve is represented by its x and y coordinate. As presented in [5], the y coordinate can be calculated based on the x coordinate. Note that the elliptic curve equation might spawn no valid values for y or two values for y. If y is invalid, it means x is not a valid coordinate to generate a point. Otherwise, the algorithm has to choose one of the two values for y and continue. Thus, one extra bit of information is required specifying which of the two values is to be used.

An elliptic curve point can be represented as a byte array in two ways: *compressed form* or *uncompressed form*. The compressed form contains the byte representation of only the x coordinate, which is prepended a byte $f \in \{02, 03\}$ as a flag to determine which value to choose for the y coordinate. Formally, $y \leftarrow \mathsf{RecoverY}(x, f)$ (algorithm 9).

The uncompressed form contains the byte representation of both x and y coordinates concatenated together, to which is prepended the byte 04.

In our system, when an elliptic curve point has to be stored in the database, or when it needs to be transferred over the network, or when it is used as input to a hash function, it is represented as a byte array in compressed form.

Algorithm 9: $\mathsf{RecoverY}(x, f)$

```
Data: The field element $x \in \mathbb{F}_p$
      The flag $f \in \{02, 03\}$
$\{y_1, y_2\} \leftarrow \sqrt{x^3 + a \cdot x + b} \pmod p$
if $\{y_1, y_2\} \notin \mathbb{Z}$ then
    return failure
end
switch $f$ do
    case 02 do
        $y \leftarrow y_1$
    end
    case 03 do
        $y \leftarrow y_2$
    end
end
return $y$                                        // $y \in \mathbb{F}_p$
```

A.2.5 Mapping a message on the Elliptic Curve

An important use case of a cryptographic system is to interpret an arbitrary message (a plain text, a number, an id, or even a more complex construction). In the elliptic curve context, that means mapping a message into an elliptic curve point in a deterministic way. Additionally, this point must be able to be interpreted back as the original message. This section considers the message as an arbitrary byte array, and it is up to specific use cases to convert data to a byte array (i.e., UTF-8 encoding of text, the byte representation of an integer, or even custom encoding of complex JSON object).

Mapping a message $\vec{b} \in \mathbb{B}^*$ into an elliptic curve point is done by $M \leftarrow \mathsf{Bytes2Point}(\vec{b})$ (algorithm 10). 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 14), where p is the prime of the elliptic curve). The resulting byte array is interpreted as a field element $x \in \mathbb{F}_p$. Then, the algorithm checks whether x spawns a valid point $M = (x, y)$, where y is computed according to the elliptic curve equation $y \leftarrow \mathsf{RecoverY}(x, 02)$ (algorithm 9). If valid, 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 10) converts any message $\vec{b}$ of legal size (i.e., $|\vec{b}| \leq \ell - 1$, where ℓ is the byte size of the elliptic curve) into a valid elliptic curve point M with a negligible failure rate.

Recovering the message $\vec{b}$ from an elliptic curve point M can be done by calling $\vec{b} \leftarrow \mathsf{Point2Bytes}(M)$ (algorithm 11). It extracts the byte representation of the x coordinate, disregarding the rightmost 00 bytes and the adjusting byte b_0.

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**.

Algorithm 10: Bytes2Point($\vec{b}$)

```
Data: The byte array $\vec{b} = \{b_1, ..., b_n\} \in \mathbb{B}^n$
$\ell \leftarrow$ ByteLengthOf$(p)$
if $n > \ell - 1$ then
    return failure
end
$m \leftarrow \ell - n - 1$
for $i \leftarrow 0$ to $255$ by $1$ do
    $b_0 \leftarrow i$
    $\vec{b}\,' \leftarrow \{b_0, b_1, ..., b_n, \underbrace{00, ..., 00}_{m \text{ times}}\}$
    $x \leftarrow$ Bytes2Field$(\vec{b}\,')$                        // algorithm 12
    $y \leftarrow$ RecoverY$(x, 02)$                       // algorithm 9
    if $y$ is valid then
        $M \leftarrow (x, y)$
        if $M$ is valid then
            return $M$                                  // $M \in \mathbb{P}$
        end
    end
end
return failure
```

Algorithm 11: Point2Bytes(M)

```
Data: The point $M = (x, y) \in \mathbb{P} = \mathbb{F}_p \times \mathbb{F}_p$
$\ell \leftarrow$ ByteLengthOf$(p)$                                  // algorithm 14
$\vec{b}\,' = \{b_0, b_1, ..., b_n, \underbrace{00, ..., 00}_{m \text{ times}}\} \leftarrow$ Field2Bytes$(x)$     // algorithm 13
$\vec{b} \leftarrow \{b_1, ..., b_n\}$                                    // $\ell = m + n + 1$
return $\vec{b}$                                                  // $\vec{b} \in \mathbb{B}^*$
```

Algorithm 12: Bytes2Field($\vec{b}$)

```
Data: The byte array $\vec{b} = \{b_1, ..., b_n\} \in \mathbb{B}^n$
$\ell \leftarrow$ ByteLengthOf$(p)$                                  // algorithm 14
if $n \neq \ell$ then
    return failure
end
$x \leftarrow 0$
for $i \leftarrow 1$ to $n$ by $1$ do
    $x \leftarrow x * 256 + b_i$
end
if $x > p$ then
    return failure
end
return $x$                                                  // $x \in \mathbb{F}_p$
```

Algorithm 13: Field2Bytes(x)

```
Data: The field element $x \in \mathbb{F}_p$
$\ell \leftarrow$ ByteLengthOf($p$)                                  // algorithm 14
for $i \leftarrow \ell$ to $1$ by $1$ do
    $b_i \leftarrow x \mod 256$
    $x \leftarrow \lfloor x/256 \rfloor$
end
$\vec{b} \leftarrow \{b_1, ..., b_\ell\}$
return $\vec{b}$                                                      // $\vec{b} \in \mathbb{B}^\ell$
```

Algorithm 14: ByteLengthOf(x)

```
Data: The number $x \in \mathbb{Z}$
$n \leftarrow 0$
while $x \neq 0$ do
    $n \leftarrow n + 1$
    $x \leftarrow \lfloor x/256 \rfloor$
end
return $n$                                                            // $n \in \mathbb{N}$
```

A.3 Zero Knowledge Proofs

A *zero knowledge proof* (ZKP) is an algorithm by which one party (the *prover*) can prove to another party (the *verifier*) that she knows a secret value x, without disclosing any information about x. A ZKP can be *interactive*, where the prover and the verifier have to collaborate in a protocol for the verifier to get convinced of the proof. A ZKP can also be *non-interactive*. In this case, the prover alone generates a proof that is publicly verifiable, thus convincing any public verifier of its statement.

There exist two algorithms: one for generating a proof and another for verifying whether a proof is valid. A classic proof has a structure of a triple (commitment, challenge, and response). In an interactive zero-knowledge protocol, a prover commits to a value, the verifier independently and randomly generates a challenge, the prover computes a response based on the challenge received, and the verifier checks that the proof validates. The proof can be turned into a non-interactive one using the *Fiat-Shamir heuristic* as described in [6]. The prover computes alone the challenge, in a deterministic manner, based on the commitment, using a hash function.

A.3.1 Discrete Logarithm Proofs

A simple kind of ZKP is the *discrete logarithm proof* that proves knowledge of value x, such that $Y = [x]G$, 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.

The proof of multiple discrete logarithms can be turned into a non-interactive one by computing the challenge of the proof based on the commitment using a hash function. The proof is generated by the algorithm $PK \leftarrow \mathsf{DLProve}(x, \vec{G})$ (algorithm 15), where $\vec{G} = \{G_0, ...G_n\}$ is the list of generators.

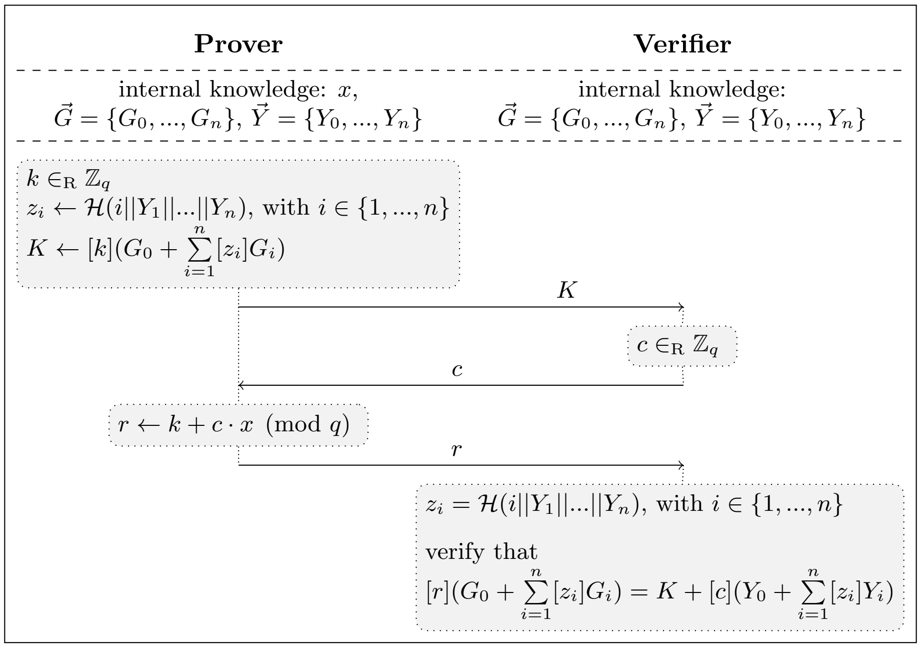

Prover
Verifier
internal knowledge: x, $\vec{G} = \{G_0, ..., G_n\}$, $\vec{Y} = \{Y_0, ..., Y_n\}$
internal knowledge: $\vec{G} = \{G_0, ..., G_n\}$, $\vec{Y} = \{Y_0, ..., Y_n\}$
$k \in_{\mathrm{R}} \mathbb{Z}_q$
$z_i \leftarrow \mathcal{H}(i||Y_1||...||Y_n)$, with $i \in \{1, ..., n\}$
$K \leftarrow [k](G_0 + \sum_{i=1}^{n} [z_i]G_i)$
K
$c \in_{\mathrm{R}} \mathbb{Z}_q$
c
$r \leftarrow k + c \cdot x \pmod q$
r
$z_i = \mathcal{H}(i||Y_1||...||Y_n)$, with $i \in \{1, ..., n\}$
verify that
$[r](G_0 + \sum_{i=1}^{n} [z_i]G_i) = K + [c](Y_0 + \sum_{i=1}^{n} [z_i]Y_i)$

Figure 11: Protocol for proving multiple discrete logarithms

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 16.

Algorithm 15: $\mathsf{DLProve}(x, \vec{G})$

```
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 16: $\mathsf{DLVer}(PK, \vec{G}; \vec{Y})$

```
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 $b$                                                     // $b \in \mathbb{B}$
```

A.4 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*. We define the hash function $\mathcal{H} : \mathbb{B}^* \leftarrow \mathbb{B}^\ell$, where $\mathbb{B}^\ell$ represents a bit array of length ℓ. In practice, hash algorithms work on byte arrays instead of bit arrays. Thus, the length of the input or output array is $\ell/8$.

A hash value can be computed for any data, such as a string, a number, or even an object with a complex structure. The hash value would result from the hash function applied to the byte representation of that particular input data. A hash value can be computed for an arbitrary number of inputs simultaneously. In that case, the hash function is applied to the concatenation of all byte representations of each input.

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*. That means finding two different input data with the same hash values is infeasible.

In our system, we will use the hash function called *SHA-256* that outputs bit arrays of 256 bits in length (32-byte array).

A.5 Elgamal cryptosystem

A.5.1 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 a triple (KeyGen, Enc, Dec) of algorithms that work on the elliptic curve described in appendix A.2.2. The scheme is considered secure under the *discrete logarithm assumption*.

An Elgamal key pair is a tuple $(x, Y) \leftarrow \mathsf{KeyGen}()$ (algorithm 17), 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.

Algorithm 17: KeyGen()

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

The encryption algorithm $e = (R, C) \leftarrow \mathsf{Enc}(Y, M; r)$ (algorithm 18) can be 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 19). 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 18: 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 19: 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 $M$                                        // $M \in \mathbb{P}$
```

A.5.2 Homomorphic Encryption

Elgamal encryption based on elliptic curve cryptographic primitive 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 20).

```
Algorithm 20: 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 21.

Algorithm 21: $\mathsf{ReEnc}(Y, e; r')$

```
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 18
$e' \leftarrow \mathsf{HomAdd}(e, e_2)$                       // algorithm 20
return $e'$                                          // $e' \in \mathbb{E}$
```

Usually, a re-encrypted cryptogram comes with a re-encryption proof to assure that the content of the cryptogram has not been changed. The proof is a non-interactive discrete logarithm equality proof (described in appendix A.3.1) $PK = (K, c, r) \leftarrow \mathsf{DLProve}(r', \{G, Y\})$ (algorithm 15), while the proof verification algorithm is $\mathsf{DLVer}(PK, \{G, Y\}; \{R' - R, C' - C\})$ (algorithm 16).

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 22).

Algorithm 22: $\mathsf{HomMul}(e; n)$

```
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 $e'$                                          // $e' \in \mathbb{E}$
```

A.5.3 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 [9, 10].

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 17) 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 17), 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 23), where $k_{i,j} \leftarrow \mathsf{DerSymKey}(x_i, Y_j)$ (algorithm 36). 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 24), where $k_{i,j} \leftarrow \mathsf{DerSymKey}(x_i, Y_j)$ (algorithm 36). Recall form appendix A.9.2 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 A.5.1: $e = (R, C) \leftarrow \mathsf{Enc}(Y, M; r)$.

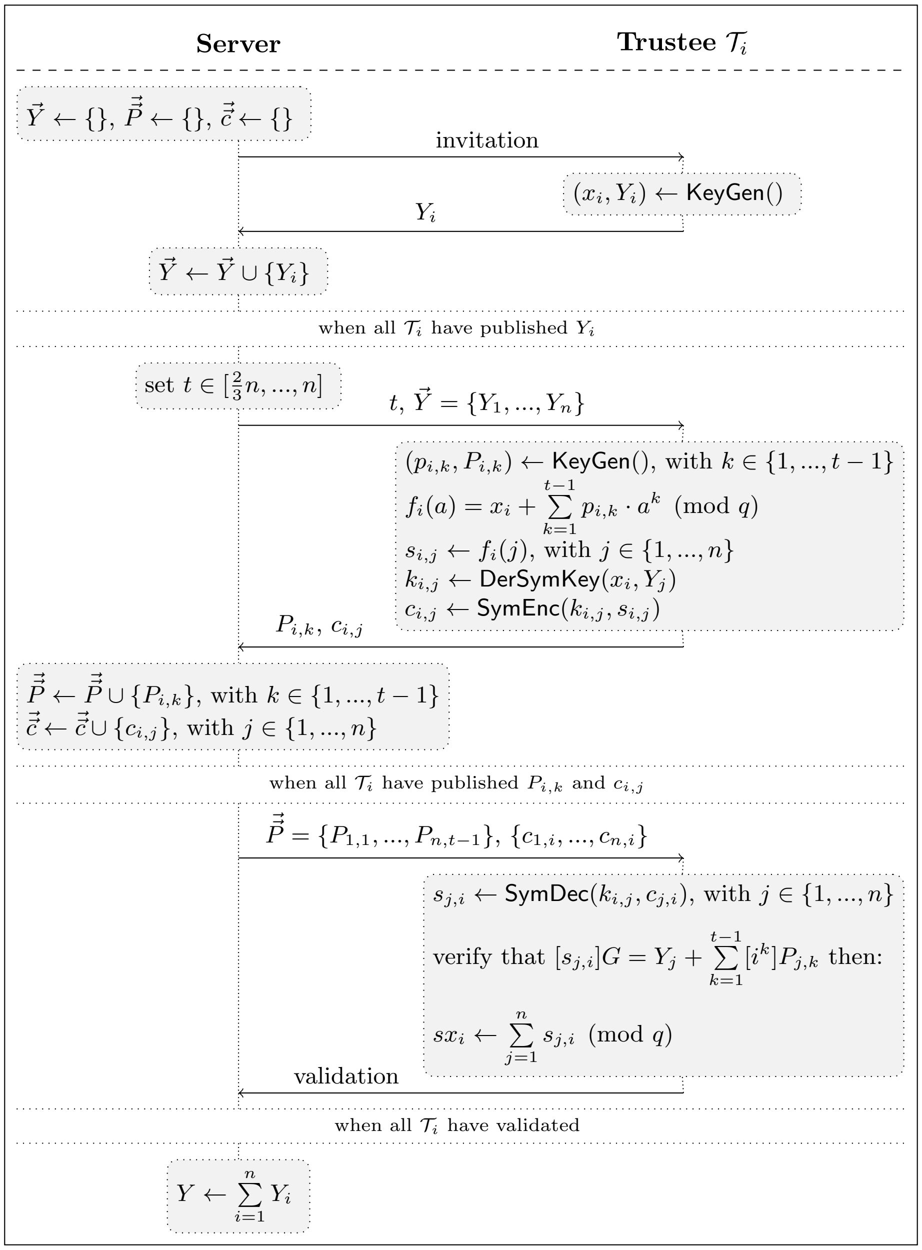

Server
Trustee $\mathcal{T}_i$
$\vec{Y} \leftarrow \{\}$, $\vec{\vec{P}} \leftarrow \{\}$, $\vec{c} \leftarrow \{\}$
invitation
$(x_i, Y_i) \leftarrow \mathsf{KeyGen}()$
Y_i
$\vec{Y} \leftarrow \vec{Y} \cup \{Y_i\}$
when all $\mathcal{T}_i$ have published Y_i
set $t \in [\frac{2}{3}n, ..., n]$
t, $\vec{Y} = \{Y_1, ..., Y_n\}$
$(p_{i,k}, P_{i,k}) \leftarrow \mathsf{KeyGen}()$, with $k \in \{1, ..., t-1\}$
$f_i(a) = x_i + \sum_{k=1}^{t-1} p_{i,k} \cdot a^k \pmod q$
$s_{i,j} \leftarrow f_i(j)$, with $j \in \{1, ..., n\}$
$k_{i,j} \leftarrow \mathsf{DerSymKey}(x_i, Y_j)$
$c_{i,j} \leftarrow \mathsf{SymEnc}(k_{i,j}, s_{i,j})$
$P_{i,k}$, $c_{i,j}$
$\vec{\vec{P}} \leftarrow \vec{\vec{P}} \cup \{P_{i,k}\}$, with $k \in \{1, ..., t-1\}$
$\vec{c} \leftarrow \vec{c} \cup \{c_{i,j}\}$, with $j \in \{1, ..., n\}$
when all $\mathcal{T}_i$ have published $P_{i,k}$ and $c_{i,j}$
$\vec{\vec{P}} = \{P_{1,1}, ..., P_{n,t-1}\}$, $\{c_{1,i}, ..., c_{n,i}\}$
$s_{j,i} \leftarrow \mathsf{SymDec}(k_{i,j}, c_{j,i})$, with $j \in \{1, ..., n\}$
verify that $[s_{j,i}]G = Y_j + \sum_{k=1}^{t-1} [i^k] P_{j,k}$ then:
$sx_i \leftarrow \sum_{j=1}^{n} s_{j,i} \pmod q$
validation
when all $\mathcal{T}_i$ have validated
$Y \leftarrow \sum_{i=1}^{n} Y_i$

Figure 12: Threshold ceremony

The decryption protocol of the threshold cryptosystem is inspired by paper [9]. 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 15).

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 16), 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 [9] 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., $|\tau| \geq t$.

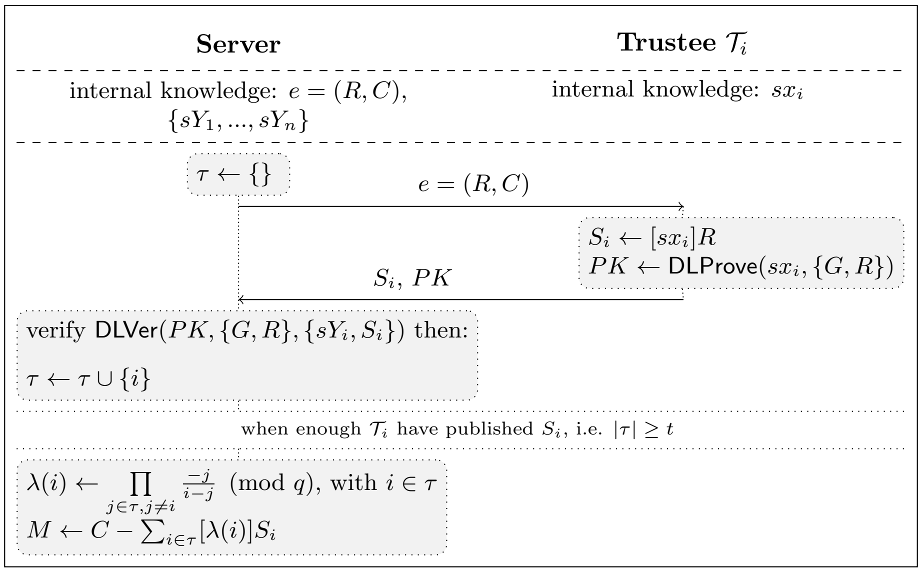

Server
Trustee $\mathcal{T}_i$
internal knowledge: $e = (R, C)$, $\{sY_1, ..., sY_n\}$
internal knowledge: sx_i
$\tau \leftarrow \{\}$
$e = (R, C)$
$S_i \leftarrow [sx_i]R$
$PK \leftarrow \mathsf{DLProve}(sx_i, \{G, R\})$
S_i, PK
verify $\mathsf{DLVer}(PK, \{G, R\}, \{sY_i, S_i\})$ then:
$\tau \leftarrow \tau \cup \{i\}$
when enough $\mathcal{T}_i$ have published S_i, i.e. $|\tau| \geq t$
$\lambda(i) \leftarrow \prod_{j\in\tau, j\neq i} \frac{-j}{i-j} \pmod q$, with $i \in \tau$
$M \leftarrow C - \sum_{i\in\tau}[\lambda(i)]S_i$

Figure 13: Threshold decryption

A.5.4 Proving the Content of a Cryptogram

Once a cryptogram is generated $e = (R, C) \leftarrow \mathsf{Enc}(Y, M; r)$, only the *sender* (the one who generated the cryptogram) and the *receiver* (the one in possession of the decryption key x) know the value of the message M. Both of them have the possibility to prove to somebody else (or publicly prove) the content of the cryptogram.

The one who generated the cryptogram can prove to a verifier that the cryptogram e contains message M by engaging in the protocol from figure 11 to prove the knowledge of the randomizer $PK[(r) : R = [r]G \wedge (C - M) = [r]Y]$. To generate a publicly verifiable proof, the *sender* can generate a non-interactive proof $PK \leftarrow \mathsf{DLProve}(r, \{G, Y\})$ (algorithm 15). Any public verifier is convinced that cryptogram e contains message M if the verification algorithm succeeds $\mathsf{DLVer}(PK, \{G, Y\}; \{R, C - M\})$ (algorithm 16).

At the same time, the one in possession of the decryption key x can prove the content of the cryptogram e to a verifier by engaging in the same protocol from figure 11 but this time for proving the knowledge of the decryption key $PK[(x) : Y = [x]G \wedge (C - M) = [x]R]$. To generate a publicly verifiable proof, the *receiver* of the cryptogram can generate a non-interactive proof $PK \leftarrow \mathsf{DLProve}(x, \{G, R\})$ (algorithm 15). Any public verifier is convinced that cryptogram e contains message M if the verification algorithm succeeds $\mathsf{DLVer}(PK, \{G, R\}; \{Y, C - M\})$ (algorithm 16).

A.5.5 Symmetric encryption

A particular 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).

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* as described in appendix A.9.2. Then, the symmetric key k is used to encrypt the message m using a standard *AES encryption* algorithm [11], resulting in the encryption $e \leftarrow \mathsf{SymEnc}(k, m)$ (algorithm 23). For decryption, the same symmetric key is derived and then used to decrypt the message $m \leftarrow \mathsf{SymDec}(k, e)$ (algorithm 24).

We use AES algorithms with 256-bit keys in *Galois Counter Mode* with a random 96-bit initialization vector iv, no authentication data (i.e., $ad \leftarrow \varnothing$), and a 128-bit tag t. Therefore, we define algorithm $\mathsf{AES} - \mathsf{GCM} - \mathsf{Encrypt}$ that returns the ciphertext c representing the encryption of message m with the key k after the AES-GCM cipher has been initialized with the initialization vector iv. Additionally, it returns tag t that authenticates the encryption. We consider as the encryption of message m the tuple $e = (iv, t, c)$.

We also define algorithm $\mathsf{AES-GCM-Decrypt}$ that returns plain text m as the decryption of ciphertext c with key k after initializing the AES-GCM cipher with the initialization vector iv. Note that m is returned only if the authentication tag t validates.

Algorithm 23: $\mathsf{SymEnc}(k, m)$

```
Data: The symmetric key $k \in \mathbb{B}^{256}$
      The message $m \in \mathbb{B}^*$
$iv \in_R \mathbb{B}^{96}$
$ad \leftarrow \varnothing$
$tl \leftarrow 128$
$(c, t) \leftarrow \mathsf{AES-GCM-Encrypt}(k, m, iv, ad, tl)$
$e \leftarrow (iv, t, c)$
return $e$                                   // $e \in \mathbb{B}^{96} \times \mathbb{B}^{128} \times \mathbb{B}^*$
```

Algorithm 24: $\mathsf{SymDec}(k, e)$

```
Data: The symmetric key $k \in \mathbb{B}^{256}$
      The encryption $e = (iv, t, c) \in \mathbb{B}^{96} \times \mathbb{B}^{128} \times \mathbb{B}^*$
$ad \leftarrow \varnothing$
$m \leftarrow \mathsf{AES-GCM-Decrypt}(k, c, iv, ad, t)$
if $m = failure$ then
    return $failure$                         // invalid authentication tag
else
    return $m$                               // $m \in \mathbb{B}^*$
end
```

A.6 Schnorr digital signature

The *Schnorr digital signature scheme*, introduced in [12], 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 17), 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 25. 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 26.

Algorithm 25: $\mathsf{Sign}(x, m)$

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

Algorithm 26: $\mathsf{SigVer}(Y, \sigma; m)$

```
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 $b$                                         // $b \in \mathbb{B}$
```

A.7 Pedersen commitment scheme

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 [13]. Later, the scheme has been updated in [14] 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 28), where $r \in_R \mathbb{Z}_q$ 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 27), 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 29).

Algorithm 27: $\mathsf{BaseGen}(i)$

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

Algorithm 28: $\mathsf{Com}(\vec{m}; r)$

```
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 27
end
$C \leftarrow [r]G + \sum_{i=1}^{n} [m_i]G_i$
return $C$                                                  // $C \in \mathbb{P}$
```

Algorithm 29: $\mathsf{ComVer}(C, \vec{m}; r)$

```
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 27
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 $b$                                                  // $b \in \mathbb{B}$
```

A.8 Groth's 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 30) where each $e'_{i,j} = (R'_{i,j}, C'_{i,j}) \leftarrow \mathsf{ReEnc}(Y, e_{k,j}; r_{i,j})$ (algorithm 21) for $k = \psi(i)$.

Algorithm 30: $\mathsf{Shuffle}(Y, \vec{\vec{e}}, \vec{\vec{r}}, \psi)$

```
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 $\ell$ by $1$ do
        $e'_{i,j} \leftarrow \mathsf{ReEnc}(Y, e_{\psi(i),j}; r_{i,j})$          // algorithm 21
    end
end
$\vec{\vec{e}}' \leftarrow \{e'_{1,1}, ..., e'_{n,\ell}\}$
return $\vec{\vec{e}}'$                                          // $\vec{\vec{e}}' \in \mathbb{E}^{n\times\ell}$
```

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 has been changed. Our mixing proof is based on an algorithm presented by Jens Groth in [15]. 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.

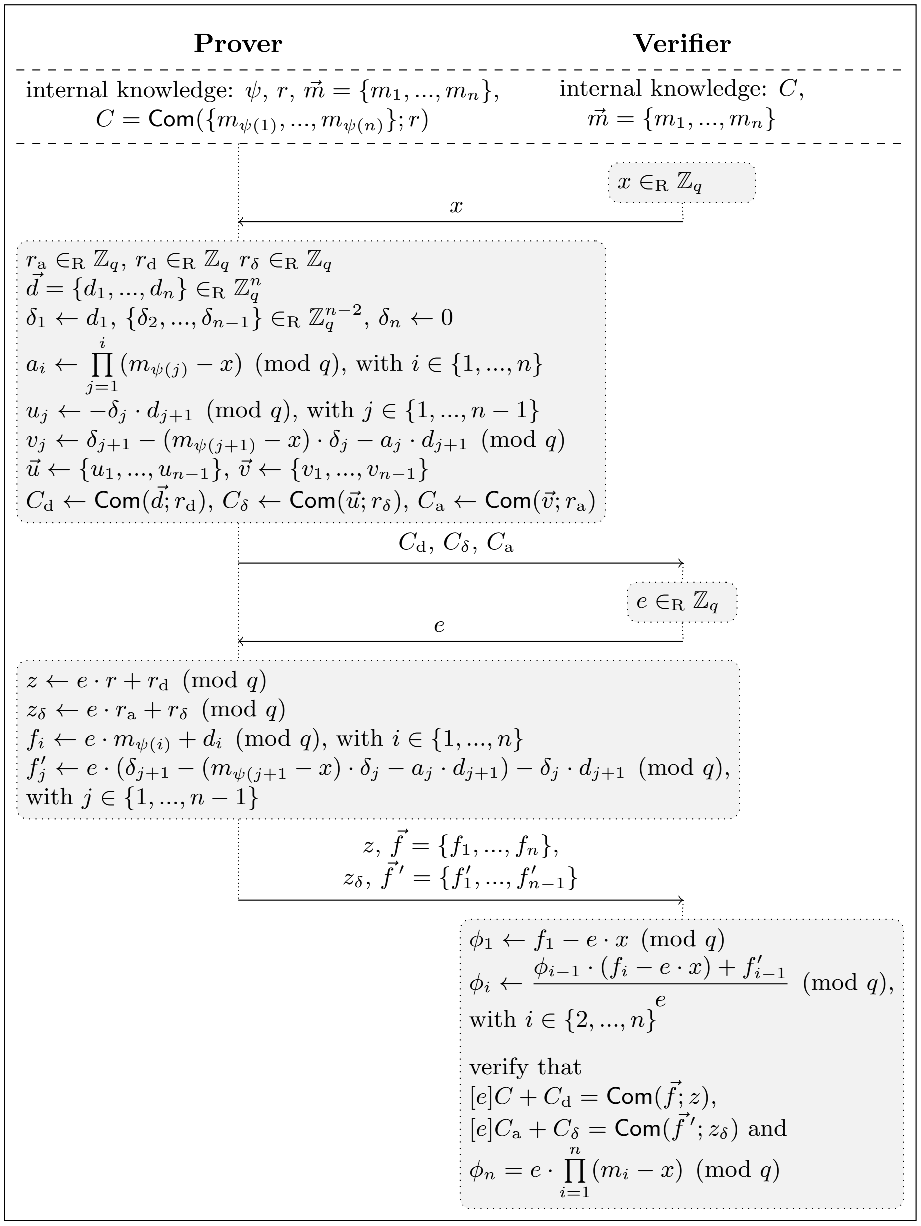
The protocol for the Argument of Shuffle of Known Contents is presented in figure 14. During this protocol, the *Prover* convinces the *Verifier* that C is a commitment to a set of known messages $\vec{m} = \{m_1, ..., m_n\}$ that are shuffled by a secret permutation ψ. Note that, in this protocol, the *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 20) as the homomorphic addition of multiple cryptograms, with each $e_i \in \mathbb{E}$.

Jens Groth suggests in [15] that the protocols can be turned into non-interactive algorithms by using the *Fiat-Shamir heuristic* strategy [6] 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 33), 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 34).



Prover | **Verifier**

Prover internal knowledge: ψ, r, $\vec{m} = \{m_1, ..., m_n\}$, $C = \mathsf{Com}(\{m_{\psi(1)}, ..., m_{\psi(n)}\}; r)$

Verifier internal knowledge: C, $\vec{m} = \{m_1, ..., m_n\}$

Verifier: $x \in_{\mathrm{R}} \mathbb{Z}_q$

Verifier → Prover: x

Prover:

$r_{\mathrm{a}} \in_{\mathrm{R}} \mathbb{Z}_q$, $r_{\mathrm{d}} \in_{\mathrm{R}} \mathbb{Z}_q$ $r_\delta \in_{\mathrm{R}} \mathbb{Z}_q$
$\vec{d} = \{d_1, ..., d_n\} \in_{\mathrm{R}} \mathbb{Z}_q^n$
$\delta_1 \leftarrow d_1$, $\{\delta_2, ..., \delta_{n-1}\} \in_{\mathrm{R}} \mathbb{Z}_q^{n-2}$, $\delta_n \leftarrow 0$
$a_i \leftarrow \prod_{j=1}^{i} (m_{\psi(j)} - x) \pmod q$, with $i \in \{1, ..., n\}$
$u_j \leftarrow -\delta_j \cdot d_{j+1} \pmod q$, with $j \in \{1, ..., n-1\}$
$v_j \leftarrow \delta_{j+1} - (m_{\psi(j+1)} - x) \cdot \delta_j - a_j \cdot d_{j+1} \pmod q$
$\vec{u} \leftarrow \{u_1, ..., u_{n-1}\}$, $\vec{v} \leftarrow \{v_1, ..., v_{n-1}\}$
$C_{\mathrm{d}} \leftarrow \mathsf{Com}(\vec{d}; r_{\mathrm{d}})$, $C_\delta \leftarrow \mathsf{Com}(\vec{u}; r_\delta)$, $C_{\mathrm{a}} \leftarrow \mathsf{Com}(\vec{v}; r_{\mathrm{a}})$

Prover → Verifier: C_{d}, C_δ, C_{a}

Verifier: $e \in_{\mathrm{R}} \mathbb{Z}_q$

Verifier → Prover: e

Prover:

$z \leftarrow e \cdot r + r_{\mathrm{d}} \pmod q$
$z_\delta \leftarrow e \cdot r_{\mathrm{a}} + r_\delta \pmod q$
$f_i \leftarrow e \cdot m_{\psi(i)} + d_i \pmod q$, with $i \in \{1, ..., n\}$
$f'_j \leftarrow e \cdot (\delta_{j+1} - (m_{\psi(j+1} - x) \cdot \delta_j - a_j \cdot d_{j+1}) - \delta_j \cdot d_{j+1} \pmod q$, with $j \in \{1, ..., n-1\}$

Prover → Verifier: z, $\vec{f} = \{f_1, ..., f_n\}$, z_δ, $\vec{f}\,' = \{f'_1, ..., f'_{n-1}\}$

Verifier:

$\phi_1 \leftarrow f_1 - e \cdot x \pmod q$
$\phi_i \leftarrow \dfrac{\phi_{i-1} \cdot (f_i - e \cdot x) + f'_{i-1}}{e} \pmod q$, with $i \in \{2, ..., n\}$

verify that
$[e]C + C_{\mathrm{d}} = \mathsf{Com}(\vec{f}; z)$,
$[e]C_{\mathrm{a}} + C_\delta = \mathsf{Com}(\vec{f}\,'; z_\delta)$ and
$\phi_n = e \cdot \prod_{i=1}^{n} (m_i - x) \pmod q$

Figure 14: Argument of Shuffle of Known Contents

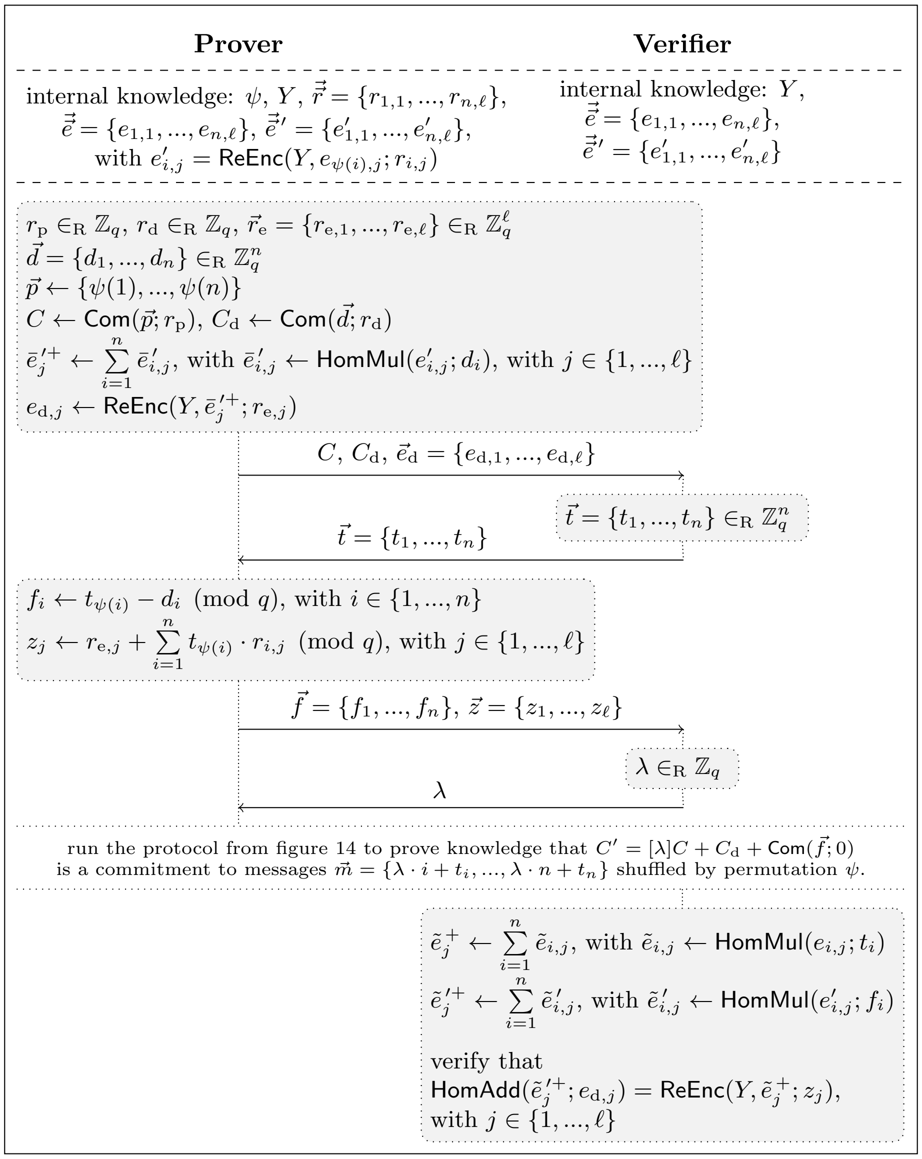

Prover
Verifier
internal knowledge: ψ, Y, $\vec{r} = \{r_{1,1}, ..., r_{n,\ell}\}$, $\vec{e} = \{e_{1,1}, ..., e_{n,\ell}\}$, $\vec{e}' = \{e'_{1,1}, ..., e'_{n,\ell}\}$, with $e'_{i,j} = \mathsf{ReEnc}(Y, e_{\psi(i),j}; r_{i,j})$
internal knowledge: Y, $\vec{e} = \{e_{1,1}, ..., e_{n,\ell}\}$, $\vec{e}' = \{e'_{1,1}, ..., e'_{n,\ell}\}$
$r_\mathrm{p} \in_\mathrm{R} \mathbb{Z}_q$, $r_\mathrm{d} \in_\mathrm{R} \mathbb{Z}_q$, $\vec{r}_\mathrm{e} = \{r_{\mathrm{e},1}, ..., r_{\mathrm{e},\ell}\} \in_\mathrm{R} \mathbb{Z}_q^\ell$
$\vec{d} = \{d_1, ..., d_n\} \in_\mathrm{R} \mathbb{Z}_q^n$
$\vec{p} \leftarrow \{\psi(1), ..., \psi(n)\}$
$C \leftarrow \mathsf{Com}(\vec{p}; r_\mathrm{p})$, $C_\mathrm{d} \leftarrow \mathsf{Com}(\vec{d}; r_\mathrm{d})$
$\bar{e}'^{+}_j \leftarrow \sum_{i=1}^{n} \bar{e}'_{i,j}$, with $\bar{e}'_{i,j} \leftarrow \mathsf{HomMul}(e'_{i,j}; d_i)$, with $j \in \{1, ..., \ell\}$
$e_{\mathrm{d},j} \leftarrow \mathsf{ReEnc}(Y, \bar{e}'^{+}_j; r_{\mathrm{e},j})$
C, C_d, $\vec{e}_\mathrm{d} = \{e_{\mathrm{d},1}, ..., e_{\mathrm{d},\ell}\}$
$\vec{t} = \{t_1, ..., t_n\} \in_\mathrm{R} \mathbb{Z}_q^n$
$\vec{t} = \{t_1, ..., t_n\}$
$f_i \leftarrow t_{\psi(i)} - d_i \pmod q$, with $i \in \{1, ..., n\}$
$z_j \leftarrow r_{\mathrm{e},j} + \sum_{i=1}^{n} t_{\psi(i)} \cdot r_{i,j} \pmod q$, with $j \in \{1, ..., \ell\}$
$\vec{f} = \{f_1, ..., f_n\}$, $\vec{z} = \{z_1, ..., z_\ell\}$
$\lambda \in_\mathrm{R} \mathbb{Z}_q$
λ
run the protocol from figure 14 to prove knowledge that $C' = [\lambda]C + C_\mathrm{d} + \mathsf{Com}(\vec{f}; 0)$ is a commitment to messages $\vec{m} = \{\lambda \cdot i + t_i, ..., \lambda \cdot n + t_n\}$ shuffled by permutation ψ.
$\tilde{e}^{+}_j \leftarrow \sum_{i=1}^{n} \tilde{e}_{i,j}$, with $\tilde{e}_{i,j} \leftarrow \mathsf{HomMul}(e_{i,j}; t_i)$
$\tilde{e}'^{+}_j \leftarrow \sum_{i=1}^{n} \tilde{e}'_{i,j}$, with $\tilde{e}'_{i,j} \leftarrow \mathsf{HomMul}(e'_{i,j}; f_i)$
verify that $\mathsf{HomAdd}(\tilde{e}'^{+}_j; e_{\mathrm{d},j}) = \mathsf{ReEnc}(Y, \tilde{e}^{+}_j; z_j)$, with $j \in \{1, ..., \ell\}$

Figure 15: Argument of Shuffle of Cryptograms

```
Algorithm 31: $\mathsf{ASKCProve}(\psi; r; \vec{m}; C)$
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 28
$e \leftarrow \mathcal{H}(\vec{m}||C||C_{\mathrm{d}}||C_\delta||C_{\mathrm{a}})$
$z \leftarrow e \cdot r + r_{\mathrm{d}} \pmod q$, $z_\delta \leftarrow e \cdot r_{\mathrm{a}} + r_\delta \pmod q$
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 32: $\mathsf{ASKCVer}(AS; \vec{m}; C)$

```
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
    $\phi_i \leftarrow \frac{\phi_{i-1} \cdot (f_i - e \cdot x) + f'_{i-1}}{e} \pmod q$
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 28
 then
    $b \leftarrow 1$                                      // argument is valid
else
    $b \leftarrow 0$                                      // argument is invalid
end
return $b$                                                // $b \in \mathbb{B}$
```

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

```
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 $\ell$ by 1 do
        $\bar{e}'_{i,j} \leftarrow \mathsf{HomMul}(e'_{i,j}; d_i)$                  // algorithm 22
    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 28
for $j \leftarrow 1$ to $\ell$ 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 21
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 $\ell$ 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 28
$AS \leftarrow \mathsf{ASKCProve}(\psi; r'; \vec{m}'; C')$                        // algorithm 31
$PM \leftarrow (C, C_\mathrm{d}, \vec{e}_\mathrm{d}, \vec{t}, \vec{f}, \vec{z}, \lambda)$
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 34: $\mathsf{MixVer}(PM, AS, Y, \vec{\vec{e}}, \vec{\vec{e}}')$

```
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 $\ell$ by $1$ do
        $\tilde{e}_{i,j} \leftarrow \mathsf{HomMul}(e_{i,j}; t_i)$                  // algorithm 22
        $\tilde{e}'_{i,j} \leftarrow \mathsf{HomMul}(e'_{i,j}; f_i)$                 // algorithm 22
    end
    $m_i \leftarrow \lambda \cdot i + t_i \pmod q$
end
for $j \leftarrow 1$ to $\ell$ 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 28
$\vec{m} \leftarrow \{m_1, ..., m_n\}$
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 20, 21 and 32
 then
    $b \leftarrow 1$                                     // proof is valid
else
    $b \leftarrow 0$                                     // proof is invalid
end
return $b$                                               // $b \in \mathbb{B}$
```

A.9 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.

A.9.1 Password-based key derivation function

PBKDF2 is a standard algorithm, described in [16], 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.

We use PBKDF2 as a building block for Pass2Key(m) (algorithm 35) that converts password m into a key pair (x, Y). This can be seen as an alternative algorithm to KeyGen() (algorithm 17) 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 [17]. 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 35: Pass2Key(m)

```
Data: A text $m \in \mathbb{B}^*$
$salt \leftarrow \varnothing$
$iterations \leftarrow 600.000$
$\ell \leftarrow$ ByteLengthOf($q$)                              // algorithm 14
$i \leftarrow 0$
repeat
    $x \leftarrow$ PBKDF2($\mathcal{H}, m||i, salt, iterations, \ell$)
    if $x \geq q$ then
        $i \leftarrow i + 1$
        continue
    end
until $x < q$
$Y \leftarrow [x]G$
return $(x, Y)$                                    // $(x, Y) \in \mathbb{Z}_q \times \mathbb{P}$
```

A.9.2 Diffie Hellman key derivation function

To use symmetric encryption for encrypting arbitrary amounts of data, a symmetric key needs to be derived from a private-public key environment. Algorithm $\mathsf{DerSymKey}(x_1, Y_2)$ (algorithm 36) deterministically computes a 256-bit symmetric key $k \in \mathbb{B}^{256}$, given a private key and a public key that are not related (i.e., $Y_2 \neq [x_1]G$).

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 (described in [18]) 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).

Algorithm 36: $\mathsf{DerSymKey}(x, Y)$

```
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 $k$                                        // $k \in \mathbb{B}^{256}$
```

B 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:

Configuration items

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 $\{\mathcal{I}_1, ..., \mathcal{I}_{n_\text{i}}\}$	latest configuration item	The first item defines the voter authorization configuration. The following items update the configuration. The configuration of Identity Providers is included only if voter authorization mode is **identity-based**.
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.

Voting 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.

Item	**Writer**	**Content**	**Parent type**	**Validation rules**
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.
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.

Hidden 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.

Result 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 $\vec{\vec{e}}_0 = \{e_1, ..., e_{n_\mathrm{e}}\}$.
extraction confirmation	$\mathcal{E}$	list of trustees that participated in the result ceremony $\tau \subset \{1, ..., n_\mathrm{t}\}$, 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.

C Extra features

C.1 Affidavit document extension

C.1.1 General description

The affidavit document extension is an optional feature that introduces an extra authentication factor for voters in the form of a document hand signed by the voter. The affidavit is submitted encrypted together with the digital ballot. Affidavits are manually checked by election officials, which can either accept or reject them. The affidavit verification process can be done continuously throughout the election phase or as a bulk process before the cleansing procedure. During the post-election phase, ballots get included in the tally only if they have been marked as accepted after the affidavit verification process.

The affidavit document comes in the form of a PDF document. It gets encrypted by the voting application and stored privately by the digital ballot box. During the affidavit verification process, the affidavit is decrypted by the election administrator service and presented to an election official for assessment.

A fingerprint of the encrypted affidavit is included in the ballot submission, signed by the voting application, and published on the bulletin board for trackability and integrity reasons. The encrypted affidavit is not publicly accessible on the bulletin board for privacy concerns.

C.1.2 Election protocol modification

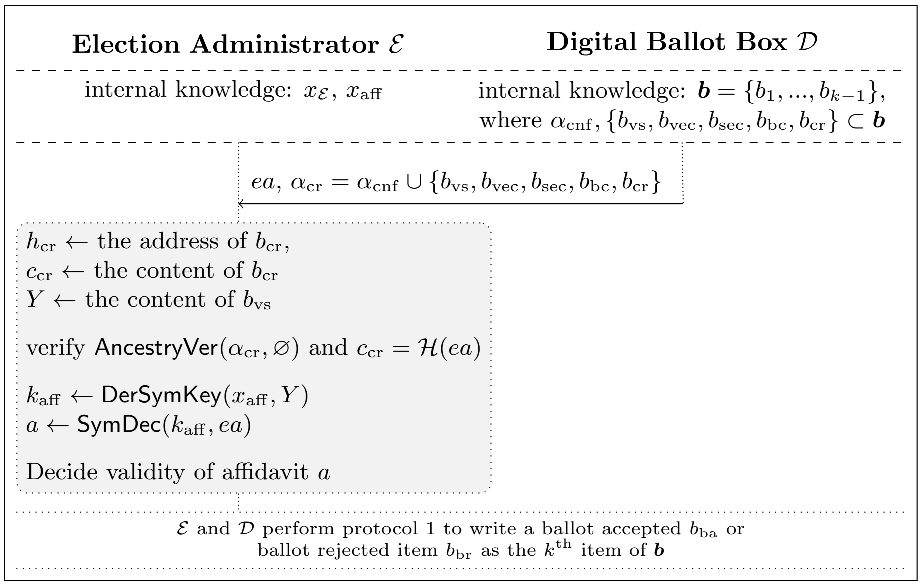
In the pre-election phase, the election administrator $\mathcal{E}$ generates another key pair that is used for encrypting/decrypting the affidavit documents $(x_{\text{aff}}, Y_{\text{aff}}) \leftarrow$ $\mathsf{KeyGen}()$ (algorithm 17), where x_{aff} is the private key used for decryption and will be kept secret by the election administrator, and Y_{aff} is the public key that voters will use to encrypt affidavits. The public key Y_{aff} is included in the election configuration item. Thus it is publicly accessible on the bulletin board.

During the election phase, a voter $\mathcal{V}_i$ must provide the affidavit document to the voting application. Before publishing the cast request item, as described in section 3.3.5, the voting application computes the encryption of the affidavit document $ea \leftarrow \mathsf{SymEnc}(k_{\text{aff}}, a)$ (algorithm 23), where $a \in \mathbb{B}^*$ is the byte representation of the affidavit document. The symmetric key k_{aff} is derived from the Diffie-Hellman key exchange mechanism $k_{\text{aff}} \leftarrow \mathsf{DerSymKey}(x_i, Y_{\text{aff}})$ (algorithm 36), where x_i is the voter's private key (as in section 3.3.1) and Y_{aff} is the affidavit public encryption key, defined in the election configuration item.

A fingerprint of the encrypted affidavit (i.e., $\mathcal{H}(ea)$) is included in the content of the cast request item. The encrypted affidavit is submitted alongside the cast request item to the digital ballot box, which validates the request and appends the item on the bulletin board if the fingerprint matches the encrypted affidavit. Note that the digital ballot box cannot read the contents of the affidavit document, as it does not know the affidavit decryption key x_{aff}.

To perform the affidavit verification process (figure 16), an election official interacts with the election administrator service, which gets the encrypted affidavit ea of the voter $\mathcal{V}_i$, together with the ancestry of items $\alpha_{\text{cr}} = \alpha_{\text{cnf}} \cup \{b_{\text{vs}}, b_{\text{vec}}, b_{\text{sec}}, b_{\text{bc}}, b_{\text{cr}}\}$ from the digital ballot box. The election administrator validates the ancestry by $\mathsf{AncestryVer}(\alpha_{\text{cr}}, \varnothing)$ (algorithm 1), and checks that $c_{\text{cr}} = \mathcal{H}(ea)$, where c_{cr} is the content of the cast request item b_{cr}. If all validations succeed, it decrypts the affidavit by $a \leftarrow \mathsf{SymDec}(k_{\text{aff}}, ea)$ (algorithm 24). The symmetric key is derived by $k_{\text{aff}} \leftarrow \mathsf{DerSymKey}(x_{\text{aff}}, Y_i)$, where the public key Y_i is defined in the voter session item b_{vs}. Value a is decoded as a PDF file and rendered to the election official for assessment.

The election official decides whether to accept or reject the affidavit. If accepted, the election administration $\mathcal{E}$ interacts with the digital ballot box to append a ballot accepted item b_{ba} on the bulletin board by running protocol 1 `WriteOnBoard`$(\mathcal{E}, m_{\text{ba}}, c_{\text{ba}}, p_{\text{ba}})$, where $m_{\text{ba}} =$ "ballot accepted", the content is empty, and the parent is the address of the cast request item b_{cr}. If rejected, the same interaction happens to append a ballot rejected item b_{br} by `WriteOnBoard`$(\mathcal{E}, m_{\text{br}}, c_{\text{br}}, p_{\text{br}})$, where $m_{\text{br}} =$ "ballot rejected", c_{br} contains the rejection reason, and the parent is the address of the cast request item b_{cr}.


Election Administrator $\mathcal{E}$ | **Digital Ballot Box $\mathcal{D}$**

internal knowledge: $x_{\mathcal{E}}$, x_{aff} | internal knowledge: $\boldsymbol{b} = \{b_1, ..., b_{k-1}\}$, where $\alpha_{\text{cnf}}, \{b_{\text{vs}}, b_{\text{vec}}, b_{\text{sec}}, b_{\text{bc}}, b_{\text{cr}}\} \subset \boldsymbol{b}$

ea, $\alpha_{\text{cr}} = \alpha_{\text{cnf}} \cup \{b_{\text{vs}}, b_{\text{vec}}, b_{\text{sec}}, b_{\text{bc}}, b_{\text{cr}}\}$

$h_{\text{cr}} \leftarrow$ the address of b_{cr},
$c_{\text{cr}} \leftarrow$ the content of b_{cr}
$Y \leftarrow$ the content of b_{vs}

verify $\mathsf{AncestryVer}(\alpha_{\text{cr}}, \varnothing)$ and $c_{\text{cr}} = \mathcal{H}(ea)$

$k_{\text{aff}} \leftarrow \mathsf{DerSymKey}(x_{\text{aff}}, Y)$
$a \leftarrow \mathsf{SymDec}(k_{\text{aff}}, ea)$

Decide validity of affidavit a

$\mathcal{E}$ and $\mathcal{D}$ perform protocol 1 to write a ballot accepted b_{ba} or ballot rejected item b_{br} as the k^{th} item of $\boldsymbol{b}$

Figure 16: Affidavit verification process

The affidavit extension also impacts the cleansing procedure (section 3.4.1), such that only ballots that have been marked as accepted (i.e., that are followed by a ballot accepted item) will be included in the *initial mixed board*.

C.1.3 Impact on election properties

The affidavit extension impacts the eligibility property of the election system. This feature acts as an extra authentication step for voters. The authentication validation is a manual process done by election officials, at a later time than the ballot submission. Moreover, the election officials' decisions to accept or reject affidavits cannot be audited. Therefore, the system achieves the eligibility property on an extra assumption that the election officials are trustworthy for the affidavit verification process.

C.2 Multiple result extractions

C.2.1 General description

The multiple result extractions is an optional feature that gives the election officials the ability to extract partial results during the election phase. The extracted votes follow the same processes described in section 3.4 i.e., cleansing, mixing, and decryption. The publication of each partial result is postponed until after the election phase is closed.

C.2.2 Election protocol modification

The multiple result extractions feature introduces an extra parameter in the election configuration item, called the *extraction threshold* t_{e}. This allows an extraction to happen only of more than t_{e} ballots. This is important because the anonymity property of a result is bound to the number of votes being mixed and decrypted together.

When an election official requests a partial result to be computed, the following process is followed:

- the election administrator requests a partial result to be computed by interacting with the digital ballot box in $\texttt{WriteOnBoard}(\mathcal{E}, m_{\mathrm{ei}}, c_{\mathrm{ei}}, p_{\mathrm{ei}})$ (protocol 1) to publish an extraction intent item b_{ei} on the bulletin board, according to the rules from appendix B,
- the digital ballot box identifies all the valid ballots that have not been extracted in a previous extraction $\vec{\vec{e}}' = \{\vec{e}_1, ..., \vec{e}_{n'_{\mathrm{e}}}\}$, according to section 3.4.1. Then, it checks whether the amount of valid ballots is at least double the extraction threshold, i.e., $n'_{\mathrm{e}} \geq 2 \cdot t_{\mathrm{e}}$. Otherwise, the extraction is aborted,
- the digital ballot box extracts as the initial mixed board the cryptograms $\vec{\vec{e}}_0 = \{\vec{e}_1, ..., \vec{e}_{n_{\mathrm{e}}}\}$, where $n_{\mathrm{e}} = n'_{\mathrm{e}} - t_{\mathrm{e}}$, such that there are t_{e} ballots left unextracted,
- a subset of all trustees $\mathcal{T}_i$, with $i \in \tau$ and $\tau \subset \{1, ..., n_{\mathrm{t}}\}$, collaborate in the mixing process to anonymize the encrypted ballots, as described in

section 3.4.2, where n_t is the total number of trustees and $t \leq |\tau| \leq n_\mathrm{t}$ (recall from section 3.2.5 that t is threshold decryption value),

- the same subset of trustees collaborate in the decryption process (section 3.4.3) of the anonymized votes,
- finally, the election administrator publishes a modified extraction confirmation item as described in section 3.4.4, that contains fingerprints of the mixed boards of cryptograms, proofs, partial decryptions, and proofs of correct decryption. The item does not include the actual values, so the partial result is not publicly released.

After the election phase, an election official can request the final partial result to be computed. This follows the same process as above, only that, this time, all remaining valid ballots are extracted.

C.2.3 Impact on election properties

Two properties are affected by the multiple result extractions feature, namely privacy, and anonymity. Their definitions are listed in section 2.6.3.

The privacy property is affected because partial results are computed before the election is closed. That means votes submitted after a partial result has been extracted might be influenced based on the knowledge of the result. Even though results are not publicly released, trustees and election officials do know the outcome of each partial result, which is enough. The more partial results are computed, the more election privacy is affected.

The anonymity property is affected as ballots are anonymous in regards to how many 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). However, anonymity is still bound by the number of ballots in the mix. For example, assuming there is only one result extraction at the end of the election, but there are only two ballots in the ballot box, and both contain a vote for the same candidate, the anonymity is reduced to nothing, as both ballots are identifiable.

For that reason, it is essential to allow result extractions to happen only on a list with a substantial amount of ballots. Therefore we introduce the extraction threshold t_e as a configuration value.

C.2.4 Analysis on extraction threshold

As discussed in the previous section, the strength of the anonymity property is related to how many ballots are mixed together and how identifiable a ballot could be. In this section, we describe how a ballot could be identified and give recommendations for the extraction threshold t_e for a reasonable anonymity level. We consider three scenarios that could lead to a ballot being identifiable.

The analysis is also based on how many unique vote options exist on a particular ballot. For example, in a referendum, there are two vote options (i.e., any valid vote can be either a 'yes/for' or a 'no/against'). In a candidate election, there are as many vote options as there are candidates. In multiple-choice and ranked elections, the amount of unique vote options is a combination of the amount of allowed choices from the number of candidates. We consider a ballot that allows a write-in vote to have an infinite amount of vote options. Therefore, we exclude it from the analysis. The analysis assumes that each vote option has an equal probability of appearing (i.e., voters vote randomly).

All ballots are identical is the scenario where we consider all voters voting on the same option. By doing so, anonymity is broken as it is visible that each voter voted for that option. The probability of this scenario happening is

$$\left(\frac{1}{x}\right)^n$$

where x is the number of vote options, and n is the number of ballots. As this vulnerability is critical because it would affect all voters included in that result, it is recommended that the chance of this scenario to happen to be maximum one in a million (i.e., the probability should be lower than 0.0001%).

Voting options	Amount of ballots	Scenario probability
2	20	0.00009%
5	9	0.00005%
10	6	0.00010%
100	3	0.00010%
1000	2	0.00010%

One unique ballot is the scenario where a voter submits a ballot uniquely identifiable/distinguishable from the other ballots. We see this scenario as a malicious intent of the voter to bypass the receipt freeness property of the system, such that the voter can identify the ballot after it has been mixed and decrypted. This kind of attack is also known as the Sicilian attack.

A ballot that supports write-in votes specifically introduces this vulnerability, as a voter could use the write-in vote as a marking mechanism for the ballot. Therefore, the following analysis covers only ballots without write-in possibilities.

The probability of this scenario happening is

$$\left(\frac{x-1}{x}\right)^{n-1}$$

where x is the number of vote options, and n is the number of ballots. As this vulnerability would affect one single ballot in case of successful performance, it is recommended that the probability of this scenario to happen to be lower than 1%.

Voting options	Amount of ballots	Scenario probability
2	8	0.78%
5	22	0.92%
10	45	0.97%
100	460	0.99%
1000	4600	1.00%

All ballots are unique is the extreme scenario where all ballots included in an extraction are distinctly identifiable. This scenario can happen only if the amount of extracted ballots is less than the number of voting options. Assuming recommendations from the previous scenario are met, this scenario is also protected against.

Considering all scenarios described above, we conclude that the extraction threshold t_e is dependent on the amount of unique vote options of a ballot, therefore, it should be configurable. Nevertheless, we recommend $t_\mathrm{e} \geq 300$.

C.2.5 Multiple voting anomaly

The multiple result extractions feature is incompatible with the feature of overwriting your vote. After an extraction has happened and a partial result has been computed, voters that have ballots included in that result are not allowed to cast a vote any longer. This measure prevents double voting.