Scytl
Innovating Democracy

sVote with Control Components Voting Protocol

Computational Proof of Complete Verifiability and Privacy

Enrique Larraia Tamara Finogina Nuria Costa
Scytl R&S
research@scytl.com

May 25, 2020

Abstract

sVote with Control components is a cryptographic voting protocol that provides complete verifiability and guarantees voting secrecy and the non-disclosure of early provisional results. This report demonstrates that sVote fulfills the requirements of the Swiss federal chancellery for completely verifiable E-voting systems. We extract precise requirements from the ordinance and the corresponding technical annex and model the sVote cryptographic voting protocol based on its design documents. Based on this model, we show in a detailed security analysis that an adversary cannot break the complete verifiability and voting secrecy properties of sVote without being detected by either the voter or by auditors.

Scytl

ACKNOWLEDGMENTS

This work has received funding from the European Commission under the auspices of PROMETHEUS Project, Horizon 2020 Innovation Action (Grant Agreement No. 780701).

Table of contents

List of figures

List of tables

Part I

Complete verifiability and voting secrecy

1 Electronic voting in Switzerland

Switzerland has a long history of the direct participation of its citizens in decision-making processes. Besides traditional elections where voters choose their representatives in the Federal Assembly, citizens can participate in several other voting events. Citizens can propose popular voting initiatives on their own (after having obtained enough popular support by collecting signatures), and the parties and governments themselves (at the communal, cantonal or federal level) can organize referendums in order to ask the citizens for their opinion on a new law or a modification of the Constitution, among others. Thus, on average Swiss citizens have the chance to participate in 3-4 voting processes a year.

Remote electronic voting was first introduced in Switzerland in three cantons: Geneva, Zurich and Neuchâtel [24]. The first binding trials were held in 2004. By 2019, 10 cantons will have offered the electronic voting channel to their electorates. However, until recently the participation rate has been restricted to up to 10% of the eligible voters. In 2011, the Federal Council of Switzerland launched a task force to study the security issues of electronic voting. As a result, the Federal Council published, in 2013, a report with the requirements for extending the use of the electronic voting systems to a larger part of the electorate. This framework [15], which became binding in January 2014, provides requirements of functionality, security, verifiability and testing/certification which allow the electronic voting systems to be extended to 30%, 50% or up to 100% of the electorate. More specifically, while current electronic voting systems may be allowed to be used for up to 30% of the electorate provided that they fulfil a certain set of functional and security requirements, systems to be used for up to 100% of the electorate are required to additionally provide verifiability features. Although the modality of electronic voting (DRE, remote) is not specified in the report, it refers to electronic voting systems where the vote is cast electronically. This paper specifically covers remote electronic voting systems.

Verifiability in remote electronic voting is traditionally divided into three types, which are related to the phase of the voting process which is verified [1]. The first step to audit is the vote preparation at the Voting Client application run in the voter's device. This application is usually in charge of encrypting the selections made by the voter prior to casting them to a remote server so that their secrecy is ensured. *Cast-as-intended* verification methods provide the voters with means to audit that the vote prepared and encrypted by the Voting Client application contains what they selected, and that no changes have been performed. *Recorded-as-cast* verification methods provide voters with mechanisms to ensure that, once cast, their votes have been correctly received and stored at the remote voting server. Finally, *counted-as-recorded* verification allows voters, auditors and third party observers to check that the result of the tally corresponds to the votes which were received and stored at the remote voting server during the voting phase.

Classically, cast-as-intended and recorded-as-cast verifiability are known as *individually verifiable* mechanisms, while counted-as-recorded is considered to be a *universally* verifiable method. The ex-

planation is simple: only the voter knows that she had actually cast a vote, and the intended content. On the other hand, anybody should be able to verify the correct outcome of the election given the votes in the ballot box.

However, the trust model and verifiability requirements defined by the Federal Council differ from these well-known properties. More specifically, the Federal Council defines two types of verifiability in the regulation for e-voting.

2 Threat model

We discus the model for complete verifiability and voting secrecy given by the Chancellery. Our interpretation of the threat model is supported by quotes from, and references to, relevant excerpts of the Chancellery's requirements [16]. The extraction of precise properties from the legislation's informally stated goals is an important step for justifying that the model matches the requirements.

After discussing the security model laid out by the Chancellery, we introduce the model used in sVote in Section 2.5 and relate both models in Section 2.6.

2.1 Individual verifiability

The security model for individual verifiability is the following.

- The voting server is trusted.
- A part of the voters may not be trustworthy.
- The Voting Client and the communication channel between the Voting Client and the voting server is not trusted.

Under this scope, the Federal Council requirement regarding verifiability is that an attacker cannot change the voter intention, prevent a vote from being stored, or cast a vote on its own, without detection from an honest voter that follows the verification protocol.

While this seems similar to the *usual* union of cast-as-intended and recorded-as-cast verifiability done in the literature, it differs from it due to the fact that in this model the voting server is trusted, which is not the case when talking in general about recorded-as-cast mechanisms [20]. We can refer to it as a "weak" recorded-as-cast verification.

2.2 Complete verifiability

Complete verifiability (individual and universal verifiability combined) assumes the following model:

- The voting server is not trusted. Instead, there exists a group of so called control components which interact with the voting server and which are trusted as a whole, under the assumption that at least

one of them is reliable (each sole control component is not trusted). The additional communication channel from the voting server to the control components is not trusted.

- The same assumptions than for individual verifiability apply to the voters, the print office, the Voting Client , and the channel between the Voting Client and the voting server.
- At least one of the auditors and her technical aids (software or hardware tools), who verify the proofs generated by the system, are trusted to behave properly.

Taking into account this trust model, the Federal Council verifiability requirements for this type are many: an attacker cannot change a vote before/after it is stored, or prevent a vote from being stored, delete it from the ballot box, as well as insert new votes, without voters or auditors noticing it. These correspond to the previous requirements for individual verifiability, taking into account that the trusted part of the system is not the server, but the control components which interact with it. Additionally, voters must have to be able to verify whether their voting credentials have been used to cast a vote in the system. Finally, auditors must receive a proof that the result of the election corresponds to the votes cast by eligible voters and accepted by the system during the voting phase. All these requirements have to be fulfilled while vote and intermediate results secrecy is preserved.

In this case, the requirements for complete verifiability cover the classic cast-as-intended, recorded-as-cast and counted-as-recorded concepts, plus additional features (such as that each voter can verify her participation or not in the election). Note that, by the definition provided, the recorded-as-cast verification may not be restricted to be verified by the voter, but also by auditors which inspect the votes registered by the trusted part of the system (the control components).

According to the report by the Federal Council, systems to be used for up to the 50% of electors are required to provide methods for individual verifiability, and systems for up to 100% of the electorate are required to provide complete verifiability, while also enforcing the separation of duties on operations impacting the privacy, integrity and verifiability of the system.

The certification process requires to provide security (cryptographic) and formal (symbolic) proofs, which demonstrate that the system fulfills the claimed security goals.

2.3 Voting secrecy

The trust model of the Federal Council regarding vote secrecy does not account for corrupting the user platform.

> **VEleS Annex Chapter 4.3:** *Under the trust assumptions for complete verifiability of the protocol, the attacker is unable to breach voting secrecy or to obtain early provisional results without changing* [1] *the voters or their user platforms maliciously.*

[1] In the French version the word corrupting is used: "compte tenu des hypothèses de confiance qui ont été formulées

Moreover, vote secrecy should be preserved only for trustworthy voters provided the server-side does not manipulate the application.

> **VEleS Annex Chapter 4.4, supplementary provision 4.4.8:** *[...] It must be ensured that the voting secrecy of trustworthy voters cannot be breached without maliciously changing their user platform through the server-sided manipulation of the application.[...]*

Our model assumes that the voters have the possibility to satisfy themselves with the correctness of the client-side application and the public key by comparing their hash values with a hash value published on trustworthy sources. This is inline with the model.

> **VEleS Annex Chapter 4.4, supplementary provision 4.4.8:** *[...] Voters should therefore be able, using a trustworthy platform, to satisfy themselves that the application is sending their vote in encrypted form with the correct key.[...]*

2.4 Parties and channels

2.4.1 Assumptions on parties

Chapter 4.3 of VEleS extends the reduced model defined in Chapter 4.3 with additional system components (control components, auditors and auditors' technical aid) and additional communication channels. We treat the complete abstract model defined of Chapter 4.3 of VEleS as the extension of the reduced abstract model.

We see the trusted system components as the combination of those present in the reduced model, and those in the extended model. The only exception is the Voting Server which is no longer trusted. Table 1 summarizes the system components in the complete abstract model.

2.4.2 Assumptions on communication channels

In the reduced abstract model (section 4.1 in the technical annex of the ordinance) all communication channels except the channel 'User platform ↔ system' and 'System ↔ print office' are trustworthy. The complete abstract model (section 4.3) extends the model with the additional communication channels 'Control component ↔ system', 'System ↔ Auditor's technical aid', 'Auditor's technical aid ↔ Auditor' and 'Control component ↔ Control component' Section 4.3 states: "*Of the **additional** communications channels, only those between the auditors and their technical aids may be deemed trustworthy.*"

Table 2 presents all possible communication channels.

à propos de la vérifiabilité complète du protocole, l'attaquant ne peut ni violer le secret du vote, ni établir des résultats partiels de manière anticipé sans corrompre les électeurs ou leurs plates-formes utilisatèurs respectives."

[2]This channel may only be regarded as trustworthy if the information has been sent by Swiss Post (section 4.2.9 page 23)

System components	Trust assumption	Section [16]
Voters	**significant proportion** of voters are **untrustworthy**	4.1
User platform	**untrustworthy** for individual and complete verifiability **trustworthy** for privacy	4.1 4.3
Trusted technical aids for voters	**trustworthy**	4.1
System (server-side)	**untrustworthy**	4.3
Print office	**trustworthy**	4.1
Control Components	**trustworthy only as the whole**	4.3
Auditors	at least **one is trustworthy**	4.3
Auditor's technical aid	at least one honest auditor has a **trustworthy** aid	4.3

Table 1: Assumption on parties of the complete abstract model defined in VEleS [16]

Communication channel	Trust assumption	Section
Voters ↔ user platform	**trustworthy**	4.1
Voters ↔ trustworthy technical aids	**trustworthy**	4.1
Trustworthy technical aids ↔ user platform	**trustworthy**	4.1
User platform ↔ system	**untrustworthy**	4.1
System ↔ print office	**untrustworthy**	4.1
Print office → voter	**trustworthy** [2]	4.1
Control component ↔ system	**untrustworthy**	4.3
System ↔ auditor's technical aids	**untrustworthy**	4.3
Auditors' technical aid ↔ auditors	**trustworthy**	4.3
Bidirectional channels for communication between control components	**untrustworthy**	4.3

Table 2: Assumption on communication channels of the complete abstract model defined in VEleS [16]

2.5 Trust model in sVote

2.5.1 Parties in sVote

The protocol specification [43] uses slightly different notations (Protocol Specifications section 1.1.1) and defines the participants of the voting protocol as follows:

- *Voter*: they participate in the election by choosing their preferred options.
- *Voting Client* : is the device used by the voter to cast their vote given the voting options selected

by the voters.

- *Voting Server*: it receives, processes and stores the votes cast by the voters in the ballot box bb.
- *Control Components* are separated in two groups; one group participates in the choice return codes and vote cast code generation, one group mixes and decrypts votes:

  Choice Return Codes Control Components (CCR): they collaborate with the Print Office indirectly (via the Voting Server) in the setup phase, and directly with the Voting Server in the voting phase, to compute the long Choice Return Codes and the long Vote Cast Code.

  Mixing Control Components (CCM): they mix and partially decrypt ciphertexts in the ballot box.

- *Print Office*: It is responsible for generating, printing and delivering the voting cards to the voters as well as for generating the required election keys [3].
- *Election Administrators*: they are responsible for generating the election configuration, verifying it, computing the results and publishing them. In the Protocol Specification this entity is divided into *Administration Board* and *Administration Portal* based on their ability to perform cryptographic operations, however, in our model we do not distinguish between those two.
- *Bulletin board*: is the entity used to store all the information generated during the election to verify the entire process. It stores the election configuration, the votes, the confirmations and keeps track of all the actions performed by each entity. The Bulletin Board is implemented as a distributed system, that includes: the election configuration (maintained by the *Print Office*), the Secure Logger (maintained by the *CCR*) and the Ballot Box (maintained by the *Voting Server*). In this document we refer to the Secure Logger as the CCR's logs.
- *Electoral Board*: This entity owns a key pair whose private key is shared among the Board members and is used to partially decrypt the votes in the last Control Component.
- *Auditors*: they verify the proofs and the information provided by the untrustworthy system components. They detect misbehavior of untrustworthy parties and, therefore, ensure the security goals of the system.
- *Verifier*: is the component used to verify the correctness of the entire election process, the integrity of the data processed through different voting system components, and that these processes are accurate and fair.

[3]For generating cryptographic material, *Print Office* runs a software called Secure Data Manger (SDM). This software is executed in a controlled, offline environment on the canton's premises. All operations on the SDM are subject to very strict 4-eyes principles and are executed on laptops with special access rights and hardened laptops.

2.5.2 Trust assumptions in sVote

Our model proves the desired security goals under the following assumptions:

- The *Voting Server* is not trusted. Instead, there are two groups of so called control components *CCM* and *CCR* that interact directly with the *Voting Server* and indirectly with the *Print Office* (via Voting Server). Each group of control components is trusted as a whole, under the assumption that at least one of them is reliable. However, each sole control component is not trusted.
- The credential delivery channel (postal channel between the *Print Office* and the voter) is considered to be trustworthy.
- The *Print Office* is trusted.
- The *Voting Client* of honest voters is trusted for privacy, and not trusted for individual and universal verifiability.
- The election configuration (number and names of the candidates, lists, questions, and answers, generated number of credentials, number of allowed options, details of electoral model) generated by the *Election Administrators* is correct. The auditors validate the number of generated voting cards using the information from the Secure Logs and the electoral board validates the correctness of the configuration using procedural means (test and control votes).
- The communication channel between the client side and the server side is not trusted.
- A significant proportion of the voters may not be trustworthy.
- The *Electoral Board* is treated as set of control component and therefore is trusted as whole, i.e. at least one Electoral Board member is assumed to be trustworthy.
- At least one of the auditors and her technical aids (software or hardware tools) are trusted to behave properly.

2.6 Correspondence between both security models

Table 3 and 4 relate the different parties and communication channels between the Chancellery and the sVote security models.

sVote's system component	Chancellery's system component	Trust assumption
Voters	Voters	**significant proportion** of voters are **non-trustworthy**
Voting Client	User platform	**untrustworthy** for individual and complete verifiability **trustworthy** for privacy
Voting Card	Trusted technical aids for voters	**trustworthy**
Voting Server	System (server-side)	**untrustworthy**
Print office	Print office	**trustworthy**
CCM CCR	Control Components	**trustworthy only as a whole**
Auditors	Auditors	at least **one is trustworthy**
Verifier	Auditor's technical aid	at least one honest auditor has a **trustworthy** aid

Table 3: Correspondence between sVote and the Chancellery assumptions made on the protocol's participants

sVote communication channels	Chancellery's communication channel	Trust assumption
Voters ↔ Voting Client	Voters ↔ user platform	**trustworthy**
Voters ↔ Voting Cards	Voters ↔ Trustworthy technical aids	**trustworthy**
no channel exists	Trustworthy technical aids ↔ User platform	**trustworthy**
Voting Client ↔ Voting Server	User platform ↔ system	**untrustworthy**
Voting Server ↔ Print office	System ↔ print office	**untrustworthy**
Print office → Voter	Print office → voter	**trustworthy** [a]
CCM ↔ Voting Server CCR ↔ Voting Server	Control component ↔ system	**untrustworthy**
Voting Server ↔ Verifier CCM ↔ Verifier CCR ↔ Verifier	System ↔ auditor's technical aids	**untrustworthy**
Verifier ↔ auditor	Auditors' technical aid ↔ auditors	**trustworthy**
no channel exists	Bidirectional channels for communication between control components	**untrustworthy**

Table 4: Correspondence between the sVote and the Chancellery assumptions made on the communication channels

[a] This channel may only be regarded as trustworthy if the information has been sent by Swiss Post (section 4.2.9 page 23)

Part II

Building blocks

3 ElGamal encryption scheme

The ElGamal [23] encryption scheme is a public-key encryption scheme instantiated over a cyclic group $\mathbb{G}_q$ of order q with generator g, where the Decision Diffie-Hellman (DDH) problem is believed to be hard.

The scheme has four algorithms. The first algorithm takes as input a security parameter λ and outputs group parameters. The remaining algorithms also take implicitly as input the security parameter and the group parameters.

$\mathsf{ParamsGen}(1^\lambda)$ This algorithm takes as input a security parameter λ and outputs $\mathtt{gparams}$ containing a description of the group $\mathbb{G}_q$, including the group order q and a group generator $g \in \mathbb{G}_q$.

$\mathsf{KeyGen}(\mathtt{gparams})$ This algorithm takes as input the group parameters and outputs a pair of secret/public keys $(sk, pk) \in \mathbb{Z}_q^* \times \mathbb{G}_q$. The secret key is randomly sampled $sk \xleftarrow{\$} \mathbb{Z}_q^*$, and the public key is set to $pk = g^{sk} \in \mathbb{G}_q$.

$\mathsf{Enc}(m, pk)$ This algorithm takes as input a message $m \in \mathbb{G}_q$ and a public key $pk \in \mathbb{G}_q$. It samples at random $r \xleftarrow{\$} \mathbb{Z}_q^*$ and outputs

$$\mathbf{c} = (c_0, c_1) = (g^r, pk^r \cdot m) \in \mathbb{G}_q^2.$$

$\mathsf{Dec}(c, sk)$ This algorithm takes as input a ciphertext $\mathbf{c} \in \mathbb{G}_q^2$, and the private key $sk \in \mathbb{Z}_q^*$ and outputs $m = c_1 \cdot c_0^{-sk} \in \mathbb{G}_q$.

ElGamal has perfect correctness: for any given pair of keys (sk, pk) it holds $\mathsf{Dec}(\mathsf{Enc}(m, pk), sk) = m$ for all $m \in \mathbb{G}_q$.

3.1 Multi-recipient ElGamal

When a single encrypter sends multiple messages to different recipients (with different public keys) the encryption algorithm can be optimized by sharing the encryption randomness at no security loss. This optimization is known as multi-recipient ElGamal [4]. The encryption and decryption algorithms are defined as follows:

$\mathsf{MultiEnc}(\mathbf{m}, \mathbf{pk})$ This algorithm takes as input a vector of messages $\mathbf{m} \in \mathbb{G}_q^\ell$, and a vector of public keys $\mathbf{pk} \in \mathbb{G}_q^\ell$. It samples at random $r \xleftarrow{\$} \mathbb{Z}_q^*$ and outputs

$$\mathbf{c} = (c_0, c_1, \ldots, c_\ell) = (g^r, pk_1^r \cdot m_1, \ldots, pk_\ell^r \cdot m_\ell) \in \mathbb{G}_q^{\ell+1}.$$

$\mathsf{MultiDec}(\mathbf{c}, \mathbf{sk})$ This algorithm takes as input a ciphertext $\mathbf{c} \in \mathbb{G}_q^{\ell+1}$, and a vector of private keys $\mathbf{sk} \in (\mathbb{Z}_q^*)^\ell$ and outputs a vector of messages $\mathbf{m} \in \mathbb{G}_q^\ell$ such that $m_i = c_i \cdot c_0^{-sk_i} \in \mathbb{G}_q$.

3.2 ElGamal over multiplicative groups

ElGamal is often instantiated over the group of quadratic residues $\mathbb{Q}_p < \mathbb{Z}_p^*$ of prime order q, for modulus $p = 2q + 1$ being a safe prime large enough to preserve the hardness of DDH. The group description is $\texttt{gparams} = (p, q, g)$, where g generates $\mathbb{Q}_p$. We will take g to be the smallest prime which is a quadratic residue in $\mathbb{Z}_p^*$.

4 Symmetric encryption schemes

Let $\mathcal{K}$, $\mathcal{IV}$, $\mathcal{M}$ be a key space, IV space, and message space, respectively, and let $\mathsf{E} : \mathcal{K} \times \mathcal{IV} \times \mathcal{M} \to \mathcal{M}$ be a length-preserving permutation in $\mathcal{M}$, with inverse function D. An IV-based probabilistic symmetric encryption scheme [42] consists in three algorithms:

$\mathsf{KeyGen_s}(\lambda)$ This algorithm takes as input a security parameter λ and outputs a random key $K \xleftarrow{\$} \mathcal{K}$.

$\mathsf{Enc_s}(M, K)$ This algorithm takes as input a plaintext $M \in \mathcal{M}$ and a key $K \in \mathcal{K}$. Then it samples a random $IV \xleftarrow{\$} \mathcal{IV}$, computes $C' = \mathsf{E}(K, IV, M)$ and outputs a ciphertext $C = IV||C'$. Here C' is the core part of ciphertext produced using the IV.

$\mathsf{Dec_s}(C, K)$ This algorithm takes as input a ciphertext $C \in \mathcal{C}$ and a key $K \in \mathcal{K}$. Then it parses $IV||C' \leftarrow C$ and outputs plaintext $M = \mathsf{D}(K, IV, C')$.

The scheme is correct if for each $M \in \mathcal{M}$ it holds $\mathsf{Dec_s}(K, \mathsf{Enc_s}(K, M)) = M$. We also require the scheme to be semantically secure. Informally this means that it is not possible to tell apart ciphertexts encrypting different plaintexts. The following formal definition of semantic security is taken from [6].

Definition 1 (IND-CPA) *A symmetric encryption scheme given by the triplet* $\mathsf{SEnc} = (\mathsf{KeyGen_s}, \mathsf{Enc_s}, \mathsf{Dec_s})$ *is* IND-CPA *secure if for any PPT algorithm* $\mathcal{A}$ *it holds:*

$$Adv_{\mathcal{A}}^{\mathsf{SEnc},\mathsf{INDCPA}} = \left| Pr[1 \leftarrow \mathsf{INDCPA}_{\mathcal{A}}^{\mathsf{SEnc}}] - \frac{1}{2} \right| \approx 0$$

where the game $\mathsf{INDCPA}_{\mathcal{A}}^{\mathsf{SEnc}}$ *is defined in Figure 1.*

$\mathsf{INDCPA}_{\mathcal{A}}^{\mathsf{SEnc}}$:	Oracle $\mathsf{LR}_{K,b}(M_0, M_1)$:
1. Choose at random $K \xleftarrow{\$} \mathcal{K}$	1. $C \leftarrow \mathsf{Enc_s}(K, M_b)$
2. $b \xleftarrow{\$} \{0, 1\}$	2. Return C
3. $b' \leftarrow \mathcal{A}^{\mathsf{LR}_{K,b}(\cdot,\cdot)}$	
4. Return 1 iff $b = b'$	

Figure 1: IND-CPA game. The adversary is given access to a left-right encryption oracle of messages of its choosing and is asked to guess the bit b.

5 Pseudorandom functions

Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be an efficiently computable function where $\mathcal{K}$ is the key space, $\mathcal{X}$ the domain space, and $\mathcal{Y}$ the range space. The function is parameterized with public parameters pp. Informally, the function is pseudorandom if its outputs cannot be distinguished from the outputs of a truly random function.

Definition 2 (Pseudorandom) *A function $F_{\mathsf{pp}} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is* pseudorandom *if for all PPT algorithms $\mathcal{A}$ we have*

$$Adv_{\mathcal{A}}^{F,\mathsf{PRF}}(\mathsf{pp}) = \left| Pr[1 \leftarrow \mathsf{rPRF}_{\mathcal{A}}^{F}(\mathsf{pp})] - Pr[1 \leftarrow \mathsf{sPRF}_{\mathcal{A}}^{F}(\mathsf{pp})] \right| \approx 0$$

where the games $\mathsf{rPRF}_{\mathcal{A}}^{F}$ and $\mathsf{sPRF}_{\mathcal{A}}^{F}$ are defined in Figure 2.

$\mathsf{rPRF}_{\mathcal{A}}^{F}(\mathsf{pp})$:	$\mathsf{sPRF}_{\mathcal{A}}^{F}(\mathsf{pp})$:
1. Choose at random $k \xleftarrow{\$} \mathcal{K}$	1. Choose a random function $R : \mathcal{X} \to \mathcal{Y}$
2. $b \leftarrow \mathcal{A}^{F_{\mathsf{pp}}(k,\cdot)}$	2. $b \leftarrow \mathcal{A}^{R(\cdot)}$
3. Return b	3. Return b

Figure 2: **(Left)** Real game. The adversary queries adaptively the pseudorandom function F. **(Right)** Simulated game. The adversary queries adaptively the random function R.

If the key space $\mathcal{K}$ and the range space $\mathcal{Y}$ are both groups, then the function is key-homorphic if the group operation of the range space respects the group operation of the key space. See for example the formalization of Boneh et. al. [14].

Definition 3 (Key-homomorphic) *Let two groups $(\mathbb{G}_1, \star)$, and $(\mathbb{G}_2, \otimes)$, the pseudorandom function $F_{\mathsf{pp}} : \mathbb{G}_1 \times \mathcal{X} \to \mathbb{G}_2$ is* key-homomorphic *if*

$$F_{\mathsf{pp}}(k_1 \star k_2, x) = F_{\mathsf{pp}}(k_1, x) \otimes F(k_2, x) \quad \forall k_1, k_2 \in \mathbb{G}_1, \ \forall x \in \mathcal{X}.$$

Group Exponentiation as prf in rom. Naor, Pinkas, and Reingold [34] showed that the function $F_{\mathsf{pp}} : \mathbb{Z}_q \times \mathbb{G}_q \to \mathbb{G}_q$ given by $F_{\mathsf{pp}}(k, x) = x^k$ is a weak pseudorandom function under the DDH assumption over $\mathbb{G}_q$. Thus, it behaves as a pseudorandom function provided the inputs come from the random distribution. In ROM, randomness on input x (arbitrarily distributed) is enforced by feeding $\mathsf{H}(x)$ to F.

Here $\mathsf{H}(\cdot)$ denotes a cryptographic hash function with domain and range set to $\mathbb{G}_q$, and pp includes a description of $\mathbb{G}_q$. This function is clearly key-homomorphic. We will set $\mathbb{G}_q$ to be the group of quadratic residues $\mathbb{Q}_p \leq \mathbb{Z}_p^*$, hence pp includes $\mathtt{gparams} = (p, q, g)$, see Section 3.2.

6 Key derivation functions

A key derivation function (KDF) $\mathsf{KDF} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a function with which an input key from space $\mathcal{K}$ and other input data from $\mathcal{X}$ are used to generate (i.e. derive) key material in $\mathcal{Y}$ that can be employed

by cryptographic algorithms.

The protocol uses a secure KDF function that is constructed using a secure PRF function $F(k, x) \rightarrow \{0,1\}^h$ to derive L bits of keying material and defined as follows:

> $\mathsf{KDF}(k, s)$ takes as input a derivation key k that is a cryptographic key used for derivation of a key material and s is a unique value per KDF caller. Then it iterates a secure PRF $n = [\frac{L}{h}]$ times and concatenates the outputs until L bits of key material are generated. Note, that n shall not be large than $2^{32} - 1$. The key derivation is done as follows:
>
> - $n = [\frac{L}{h}]$.
> - If $n > 2^{32} - 1$, abort and return $\perp$.
> - $\mathsf{result}(0) = \emptyset$
> - For $i = 1, \ldots, n$:
>   - $K(i) = F(k, s||i)$.
>   - $\mathsf{result}(i) = \mathsf{result}(i-1)||K(i)$.
> - Return L leftmost bits of $\mathsf{result}(n)$.

Definition 4 (Pseudorandom) *A function* $\mathsf{KDF} : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ *is* pseudorandom *if for all PPT algorithms* $\mathcal{A}$ *we have*

$$Adv_{\mathcal{A}}^{\mathsf{KDF}}(k, s) = \left| Pr[1 \leftarrow \mathsf{rKDF}_{\mathcal{A}}^{\mathsf{KDF}}(k, s)] - Pr[1 \leftarrow \mathsf{sKDF}_{\mathcal{A}}^{\mathsf{KDF}}(k, s)] \right| \approx 0$$

where the games $\mathsf{rKDF}_{\mathcal{A}}^{\mathsf{KDF}}$ *and* $\mathsf{sKDF}_{\mathcal{A}}^{\mathsf{KDF}}$ *are defined in Figure 3.*

$\mathsf{rKDF}_{\mathcal{A}}^{\mathsf{KDF}}(s, k)$:	$\mathsf{sKDF}_{\mathcal{A}}^{\mathsf{KDF}}(k, s)$:
1. Compute $k' \leftarrow \mathsf{KDF}(k, s)$	1. Choose a random function $k' \xleftarrow{\$} \mathcal{Y}$
2. $b \leftarrow \mathcal{A}(k')$	2. $b \leftarrow \mathcal{A}(k')$
3. Return b	3. Return b

Figure 3: **(Left)** Real game. The adversary receives a derived key k'. **(Right)** Simulated game. The adversary receives a randomly sampled key k'.

The protocol additionally uses a password-based key derivation function specified in [26] and defined by the algorithm PBKDF as follows:

> $\mathsf{PBKDF}(\mathsf{pwd}, \mathsf{salt})$ receives as input a password string pwd and a salt salt. This algorithm derives the cryptographic key K of a length dkLen using PRF with iterN iterations.

The security of this primitive relies on the one-way and collision-resistance properties of the underlying hash function.

7 Proof systems

Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a polynomial time decidable binary relation. The language associated to $\mathcal{R}$ is the set of "yes" instances defined as:

$$\mathcal{L}_{\mathcal{R}} = \{x \mid \exists w : (x, w) \in \mathcal{R}\}.$$

For any pair $(st, w) \in \mathcal{R}$, the second element w is called a *witness* for the first element st, sometimes called the *statement*.

An interactive proof system for a language $\mathcal{L}$ is an interactive protocol between two parties, a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ where the prover aims to convince the verifier that a given string st belongs to $\mathcal{L}$ without revealing a witness for st. At the end of the interaction the verifier outputs a bit $b \in \{0,1\}$ indicating either acceptance or rejection, which will be denoted with $\langle \mathcal{P}, \mathcal{V} \rangle = b$. We will denote the public transcript generated during the execution of the protocol as $tr \leftarrow \langle \mathcal{P}, \mathcal{V} \rangle$.

An interactive proof system is *public coin* if the verifier chooses his messages uniformly at random and independently from the messages of the prover. The properties we ask the proof system to have are the following ones.

Definition 5 (Completeness) *The protocol $\langle \mathcal{P}, \mathcal{V} \rangle$ is* complete *for $\mathcal{R}$ if for any probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$ it holds:*

$$\begin{aligned} Pr[(st, w) \leftarrow \mathcal{A} : (st, w) \notin \mathcal{R} \\ \vee (tr \leftarrow \langle \mathcal{P}(st, w), \mathcal{V}(st) \rangle \wedge \langle \mathcal{P}(st, w), \mathcal{V}(st) \rangle = 1)] = 1 \end{aligned}$$

Definition 6 (Soundness) *The protocol $\langle \mathcal{P}, \mathcal{V} \rangle$ is* sound *for $\mathcal{R}$ if for any PPT algorithm $\mathcal{A}$ it holds:*

$$Pr[st \leftarrow \mathcal{A} \ : \ st \notin \mathcal{L}_{\mathcal{R}} \wedge \langle \mathcal{A}, \mathcal{V}(st) \rangle = 1] \approx 0.$$

Definition 7 (Special honest verifier zero knowledge) *The protocol $\mathsf{ps} = \langle \mathcal{P}, \mathcal{V} \rangle$ is* computational special honest verifier zero knowledge (SHVZK) *for relation $\mathcal{R}$ if there exists a PPT algorithm $\mathcal{S}_{\mathsf{zk}}$ such that for any PPT algorithm $\mathcal{A}$ it holds:*

$$Adv_{\mathcal{A}}^{\mathsf{ps},\mathsf{SHVZK}} = |Pr[1 \leftarrow \mathsf{rSHVZK}_{\mathcal{A}}^{\mathsf{ps}}] - Pr[1 \leftarrow \mathsf{sSHVZK}_{\mathcal{A}}^{\mathsf{ps}}]| \approx 0,$$

where the games rSHVZK, *and* sSHVZK *are defined in Figure 4.*

7.1 Σ-protocols

Σ-protocols are the subset of public-coin interactive proof systems that generate a three-move transcript $\pi = (c, e, z)$. The first message c, called the commitment, is sent by the prover. The second message e, called the challenge is sent by the verifier (chosen uniformly and independently from c). The third message z, called the response, is sent by the prover. In addition to the properties of public-coin proof systems outlined above, a Σ-protocol also has *special soundness*.

$\mathsf{rSHVZK}^{\mathsf{ps}}_{\mathcal{A}}$:	$\mathsf{sSHVZK}^{\mathsf{ps}}_{\mathcal{A}}$:
1. $(st, w) \leftarrow \mathcal{A}$	1. $(st, w) \leftarrow \mathcal{A}$
2. If $(st, w) \notin \mathcal{R}$ return 0	2. If $(st, w) \notin \mathcal{R}$ return 0
3. $tr \leftarrow \langle \mathcal{P}(st, w), \mathcal{V}(st) \rangle$	3. $tr \leftarrow \mathcal{S}_{\mathsf{zk}}(st)$
4. $b \leftarrow \mathcal{A}(tr)$	4. $b \leftarrow \mathcal{A}(tr)$
5. Return b	5. Return b

Figure 4: (left) real SHVZK game **(right)** simulated SHVZK game.

Definition 8 (Special soundness) *The protocol $\Sigma\langle\mathcal{P}, \mathcal{V}\rangle$ has* special soundness *for $\mathcal{R}$ if there exists a PPT algorithm $\mathcal{E}$, such that for any two accepting transcripts $\pi = (c, e, z)$, $\pi' = (c, e', z')$ for a statement st, with $e \neq e'$ then $w \leftarrow \mathcal{E}(\pi, \pi')$ with $(st, w) \in \mathcal{R}$.*

Special soundness implies soundness and proof of knowledge.

7.2 The random oracle model and the Fiat-Shamir transform

The random oracle model (ROM) is a security model that sees a cryptographic hash function as a black-box oracle that on fresh inputs returns uniform answers. This model has been used to argue security for a large variety of cryptographic schemes [8], [35].

The Fiat-Shamir heuristic (FS) [22] converts a public-coin interactive proof system into a non-interactive protocol by replacing the messages of the verifier with the output of a cryptographic hash function computed by the prover. Following [21] we denote with $\Sigma^{\mathsf{H}}(\mathcal{P}, \mathcal{V})$ the non-interactive protocol proof system (NIZK) resulting from applying the Fiat-Shamir transform to the proof system $\Sigma\langle\mathcal{P}, \mathcal{V}\rangle$ using the hash function H.

Formalizing the Fiat-Shamir transform is far from being trivial, and it was refined through a series of works [11] [12], [13], [21], [41], [45]. Indeed, if the transformation is not done properly it is unclear what type of security $\Sigma^{\mathsf{H}}(\mathcal{P}, \mathcal{V})$ guarantees [13]. We choose the formalization from [21] as it comes with rigorous security proofs.

7.2.1 Non-interactive sigma protocols in ROM

In the explicitly programmable random oracle model (EPROM) [47], all the parties have access to the random oracle RO, however the zero-knowledge simulator $\mathcal{S}_{\mathsf{ps}}$ is allowed to choose the answers of the random oracle on adversarial queries as long as they look random. Wee writes $\mathsf{RO}[\ell]$ to emphasize that the random oracle is programmed (by $\mathcal{S}_{\mathsf{ps}}$) in ℓ different queries [47].

Faust et. al. [21] exploit the general forking lemma [7] in EPROM [47] to show that, if the input to the hash function H is the pair (st, c) then the NIZK $\Sigma^{\mathsf{H}}(\mathcal{P}, \mathcal{V})$ retains the security inherited from

the underlying Σ-protocol. More concretely, [21] shows that the FS transform enjoys zero-knowledge, simulation soundness and weak simulation extractability. Intuitively, simulation soundness says that even an adversary that sees simulated proofs cannot create a proof for a false statement st^* on its own; weak simulation extractability says that the probability of accepting a valid proof is close to the probability of extracting a witness for a valid proof, or in other words, that the FS transform is non-malleable.

Definition 9 (Non-interactive zero knowledge in EPROM) *Let the proof system $\Sigma\langle\mathcal{P},\mathcal{V}\rangle$ for relation $\mathcal{R}$. The NIZK proof system $\mathsf{ps} = \Sigma^{\mathsf{H}}(\mathcal{P},\mathcal{V})$ is* zero knowledge *in the random oracle model if there exists a PPT algorithm $\mathcal{S}_{\mathsf{ps}}$ such that for any PPT algorithm $\mathcal{A}$ it holds:*

$$Adv_{\mathcal{A}}^{\mathsf{ps},\mathsf{NIZK}} = |Pr[1 \leftarrow \mathsf{rNIZK}_{\mathcal{A}}^{\mathsf{ps}}] - Pr[1 \leftarrow \mathsf{sNIZK}_{\mathcal{A}}^{\mathsf{ps}}]| \approx 0$$

where the games rNIZK, *and* sNIZK *are defined in Figure 5.*

$\underline{\mathsf{rNIZK}_{\mathcal{A}}^{\mathsf{ps}}}$: 1. Let $b \leftarrow \mathcal{A}^{\mathsf{RO},\mathcal{P}^{\mathsf{RO}}(\cdot,\cdot)}$ 2. Return b	$\underline{\mathsf{sNIZK}_{\mathcal{A}}^{\mathsf{ps}}}$: 1. Let $b \leftarrow \mathcal{A}^{\mathsf{RO}[\ell],\mathcal{S}_{\mathsf{ps},\mathsf{nizk}}(\cdot,\cdot)}$ 2. Return b
$\underline{\mathcal{S}_{\mathsf{ps},\mathsf{nizk}}(st,w)}$: 1. If $(st,w) \notin \mathcal{R}$ return $\perp$ 2. $\pi \leftarrow \mathcal{S}_{\mathsf{ps}}(st)$ 3. Return π	$\underline{\mathcal{S}_{\mathsf{ps}}(st)}$: 1. $\pi = (c,e,z) \leftarrow \mathcal{S}_{\mathsf{zk}}^{\mathsf{RO}}(st)$ 2.Update $\mathsf{RO}[\ell]$ list $\mathcal{T}$ with pair $((st,c),e)$ as query/answer. If $\mathcal{T}$ is already defined in query (st,c) output failure. 3. Return π

Figure 5: (Upper left) Real NIZK game. If $(st,w) \notin \mathcal{R}$, then $\perp \leftarrow \mathcal{P}^{\mathsf{RO}}(st,w)$. **(Upper right)** Simulated NIZK game. **(Lower left)** Non-interactive zero-knowledge simulator $\mathcal{S}_{\mathsf{ps},\mathsf{nizk}}$ of $\Sigma^{\mathsf{H}}(\mathcal{P},\mathcal{V})$. **(Lower right)** Canonical simulator $\mathcal{S}_{\mathsf{ps}}$ in ROM using the zero-knowledge simulator $\mathcal{S}_{\mathsf{zk}}$ of the interactive proof system $\Sigma\langle\mathcal{P},\mathcal{V}\rangle$. $\mathcal{S}_{\mathsf{ps}}$ controls the programmable random oracle $\mathsf{RO}[\ell]$

For completeness, the *canonical* non-interactive zero-knowledge simulator $\mathcal{S}_{\mathsf{ps}}$ of $\mathsf{ps} = \Sigma^{\mathsf{H}}(\mathcal{P},\mathcal{V})$ using the zero-knowledge simulator $\mathcal{S}_{\mathsf{zk}}$ of the interactive proof system $\Sigma\langle\mathcal{P},\mathcal{V}\rangle$ of [21] is also described in Figure 5.

Definition 10 (Simulation soundness) *Let the proof system $\Sigma\langle\mathcal{P},\mathcal{V}\rangle$ for relation $\mathcal{R}$. The NIZK proof system $\Sigma^{\mathsf{H}}(\mathcal{P},\mathcal{V})$ is* simulation sound *with respect to $\mathcal{S}_{\mathsf{ps}}$ in the random oracle model if for any PPT algorithm $\mathcal{A}$ it holds*

$$Adv_{\mathcal{A}}^{\mathsf{ps},\mathsf{sSOUND}} = Pr[1 \leftarrow \mathsf{sSOUND}_{\mathcal{A}}^{\mathsf{ps}}] \approx 0,$$

where game $\mathsf{sSOUND}_{\mathcal{A}}^{\mathsf{ps}}$ *is given in Figure 6.*

Definition 11 (Weak simulation extractability) *Let the proof system $\Sigma\langle\mathcal{P},\mathcal{V}\rangle$ for relation $\mathcal{R}$. The NIZK proof system $\Sigma^{\mathsf{H}}(\mathcal{P},\mathcal{V})$ is* weak simulation extractable *in the random oracle model with respect*

$\underline{\mathsf{sSOUND}^{\mathsf{ps}}_{\mathcal{A}}}$:

1. Init an empty list $\mathcal{T}$ for oracle $\mathsf{RO}[\ell]$
2. $(st^*, \pi^*) \leftarrow \mathcal{A}^{\mathsf{RO}[\ell], \mathcal{S}'_{\mathsf{ps}}}$
3. $b \leftarrow 0$
4. If $(st^*, \pi^*) \notin \mathcal{T} \wedge \mathcal{V}^{\mathsf{RO}[\ell]}(st^*, \pi^*) = 1 \wedge st^* \notin \mathcal{L}_{\mathcal{R}}$ then $b \leftarrow 1$
5. Return b

$\underline{\text{Oracle } \mathcal{S}'_{\mathsf{ps}}(st)}$:

1. $\pi \leftarrow \mathcal{S}_{\mathsf{ps}}(st)$
2. Return π

Figure 6: (Left) Simulation soundness game of $\Sigma^{\mathsf{H}}(\mathcal{P}, \mathcal{V})$. **(Right)** Generation of simulated proofs, for either true or false statements. The canonical simulator $\mathcal{S}_{\mathsf{ps}}$ controls the programmable random oracle $\mathsf{RO}[\ell]$ and it is described in Figure 5.

to simulator $\mathcal{S}_{\mathsf{ps}}$ with extraction error ν, if for any PPT algorithm $\mathcal{A}$, there exists an extractor $\mathcal{E}_{\mathcal{A}}$ with access to the random coins ρ of $\mathcal{A}$ and to the queries $\mathcal{T}_{\mathsf{H}}$, $\mathcal{T}$ made by $\mathcal{A}$ to oracles $\mathsf{RO}[\ell]$ and $\mathcal{S}'_{\mathsf{ps}}$, such that whenever $acc \geq \nu$ it holds

$$ext \geq \frac{1}{p}(acc - \nu)^d,$$

where p and d are polynomially bounded (in the security parameter) and

$$\begin{aligned} acc &= Pr[(st^*, \pi^*) \leftarrow \mathcal{A}^{\mathsf{RO}[\ell], \mathcal{S}'_{\mathsf{ps}}}(\rho) \ : \ (st^*, \pi^*) \notin \mathcal{T}; \mathcal{V}^{\mathsf{RO}[\ell]}(st^*, \pi^*) = 1] \\ ext &= Pr[(st^*, \pi^*) \leftarrow \mathcal{A}^{\mathsf{RO}[\ell], \mathcal{S}'_{\mathsf{ps}}}(\rho) \ ; \\ &\qquad w^* \leftarrow \mathcal{E}_{\mathcal{A}}(st^*, \pi^*, \rho, \mathcal{T}, \mathcal{T}_{\mathsf{H}}) \ : \ (st^*, \pi^*) \notin \mathcal{T}; (st^*, w) \in \mathcal{R}]. \end{aligned}$$

Above, $\mathcal{S}'_{\mathsf{ps}}$ is the oracle given in Figure 6.

7.3 Non-interactive pre-image proof systems

Let two groups $(\mathbb{G}_1, \star)$, and $(\mathbb{G}_2, \otimes)$, and an efficiently computable function $\phi : \mathbb{G}_1 \to \mathbb{G}_2$ that is an homomorphism, thus $\phi(x_1 \star x_2) = \phi(x_1) \otimes \phi(x_2)$, one can define a binary relation $\mathcal{R}_\phi$ and the language $\mathcal{L}_\phi$ associated to it as:

$$\begin{aligned} \mathcal{R}_\phi &= \{((\phi, y), w) \mid y \in \mathbb{G}_2 \wedge w \in \mathbb{G}_1 \wedge y = \phi(w)\} \\ \mathcal{L}_\phi &= \{(\phi, y) \mid \exists w \in \mathbb{G}_1 \text{ s.t. } ((\phi, y), w) \in \mathcal{R}_\phi\} \end{aligned}$$

A pre-image proof system $\Sigma_\phi\langle\mathcal{P}, \mathcal{V}\rangle$ is a Σ-protocol for the language $\mathcal{L}_\phi$. The statement is the tuple $st = (\phi, y)$, where ϕ is (a description of) the homomorphism, and a witness for st is an element $w \in \mathbb{G}_1$ s.t. $\phi(w) = y$. In the random oracle model, these proof systems can be turned non-interactive.

Definition 12 (Non-interactive pre-image proof systems) *Let $\Sigma_\phi\langle\mathcal{P}, \mathcal{V}\rangle$ a pre-image proof system with group homormorphism ϕ, and let H a cryptographic hash function with range $\mathbb{Z}_s$ for some s. The*

NIZK proof system $\Sigma_\phi^{\mathsf{H}}(\mathcal{P},\mathcal{V}) = (\mathsf{prove},\mathsf{verify})$ *resulting from applying the Fiat-Shamir transform is defined as follows:*

$\mathsf{prove}(st, w, \mathsf{aux})$**:** *On input statement* $st = (\phi, y)$*, witness* w*, and auxiliary information* aux*, where* ϕ *is a description of the homomorphism,* $w \in \mathbb{G}_1$*,* $y = \phi(w) \in \mathbb{G}_2$ *do :*

1. *Sample* $b \in \mathbb{G}_1$ *at random and compute* $c = \phi(b)$
2. *Compute* $e = \mathsf{H}(\phi, y, c, \mathsf{aux})$ *and* $z = b \star w^e$
3. *Output* $\pi = (e, z)$

$\mathsf{verify}(st, \pi, \mathsf{aux})$**:** *On input statement* $st = (\phi, y)$*, proof* $\pi = (e, z)$*, and auxiliary information* aux*, where* ϕ *is a description of the homomorphism,* $y \in \mathbb{G}_2$*,* $z \in \mathbb{G}_1$ *and* $e \in \mathbb{Z}_s$ *do:*

1. *Compute* $x = \phi(z)$ *and* $c' = x \otimes y^{-e}$
2. *If* $\mathsf{H}(\phi, y, c', \mathsf{aux}) = e$ *output* 1*, otherwise output* 0*.*

Note that H is a hash function with domain separation, thus the list of variables is encoded uniquely into a binary string.

7.3.1 On the security of NIZK pre-image proof systems

Maurer [31] proved that the interactive version $\Sigma_\phi\langle\mathcal{P},\mathcal{V}\rangle$, where the challenge $e \in \mathbb{Z}_s$ is uniformly chosen by the verifier, is complete, SHVZK, and it has special soundness. The latter implies that the protocol is a proof of knowledge with knowledge error 2^{-s}. The assumptions for the above to hold[4] [31, Theorem 3] is that there must exist $\ell \in \mathbb{Z}$ relatively prime to the difference of any two challenges $e_1 - e_2$, and an element $u \in \mathbb{G}_1$ such that $\phi(u) = y^\ell$.

Furthermore, Faust et. al. [21] show that the Fiat-Shamir transform of any Σ-protocol that meets SHVZK and special soundness properties is non-interactive zero-knowledge in the random oracle model, simulation sound and weak simulation extractable.

Last, the simulator $\mathcal{S}_{\mathsf{zk}}$ for ϕ is standard, and can be found e.g. in [31].

Selecting the groups. The non-interactive pre-image proof systems appearing in the following subsections are instantiated over the group of quadratic residues $\mathbb{Q}_p < \mathbb{Z}_p^*$ (see Section 3.2). More concretely, in all the proof systems $\mathbb{G}_1 = \mathbb{Z}_q^{n_1}$ and $\mathbb{G}_2 = \mathbb{Q}_p^{n_2}$ with q prime, for some n_1, n_2; therefore the assumptions of [31, Theorem 3] trivially hold with $\ell = q$ and $u = 0 \in \mathbb{Z}_q$. This is done for the sake of concreteness, in particular, the description of all the group homomorphisms ϕ include a concrete choice of primes $q, p = 2q + 1$. However, we emphasize that the proof systems can be instantiated over any other suitable pair of groups $\mathbb{G}_1, \mathbb{G}_2$.

[4] As pointed out in [31], if ϕ is not one-way the properties still hold though the protocol might not be useful at all.

7.3.2 Exponentiation proof system.

The exponentiation proof system $\mathsf{Exp} = (\mathsf{ProveExp}, \mathsf{VerifyExp})$ is a generalization of the Chaum-Pedersen proof system [17]. It is a pre-image proof system with group homomorphism $\phi : \mathbb{Z}_q \to \mathbb{Q}_p^n$ given by $\phi(x) = (g_1^x, ..., g_n^x)$. The description of ϕ is given by the group parameters (p, q) and n group elements $\mathbf{g} = (g_1, \ldots, g_n)$ all in $\mathbb{Q}_p$.

Let a vector $\mathbf{y} \in \mathbb{Q}_p^n$, this proof system proves that $\mathbf{y} = \phi(x)$, that is, it proves that $\mathbf{y} \in \mathbb{Q}_p^n$ has been obtained by exponentiating the bases $\mathbf{g}$ to the same exponent x. The algorithms are as follows:

$\mathsf{ProveExp}((p, q, \mathbf{g}, \mathbf{y}), x, \mathtt{aux})$**:** Execute steps of algorithm prove from Defn. 12.

- Inputs: statement $st = ((p, q, \mathbf{g}), \mathbf{y})$ and witness $w = x$, where $x \in \mathbb{Z}_q$ and $\mathbf{y} \in \mathbb{Q}_p^n$.
- Output $\pi_{\mathsf{Exp}} = (e, z) \in \mathbb{Z}_s \times \mathbb{Z}_q$.

$\mathsf{VerifyExp}((p, q, \mathbf{g}, \mathbf{y}), \pi_{\mathsf{Exp}}, \mathtt{aux})$**:** Execute steps of algorithm verify from Definition 12.

- Inputs: statement $st = ((p, q, \mathbf{g}), \mathbf{y})$ and proof $\pi_{\mathsf{Exp}} = (e, z)$, where $\mathbf{y} \in \mathbb{Q}_p^n$ and $(e, z) \in \mathbb{Z}_s \times \mathbb{Z}_q$
- Output: either 1 (accept) or 0 (reject).

7.3.3 Schnorr proof system.

The Schnorr proof system $\mathsf{sch} = (\mathsf{ProveSch}, \mathsf{VerifySch})$ first appeared in [44]. It is a pre-image proof system with group homomorphism $\phi : \mathbb{Z}_q \to \mathbb{Q}_p$ given by $\phi(r) = g^r$. The description of ϕ is given by the group parameters (p, q), and a generator g of $\mathbb{Q}_p$.

Let an ElGamal ciphertext $\mathbf{c} = (c_0, c_1)$, where ElGamal is instantiated over $\mathbb{Q}_p$ with generator g. This proof system proves knowledge of some r such that $c_0 = \phi(r)$, that is, it proves knowledge of the randomness embedded in $\mathbf{c}$. To bind the generated proof to the ciphertext, the second group element c_1 of $\mathbf{c}$ is also included as auxiliary information. The algorithms are as follows:

$\mathsf{ProveSch}((p, q, g, \mathbf{c}), r, \mathtt{aux})$**:** Execute steps of algorithm prove from Defn. 12.

- Inputs: statement $st = ((p, q, g), c_0)$, witness $w = r$, and $\mathtt{aux}' = \mathtt{aux} \cup \{c_1\}$, where $c_0 \in \mathbb{Q}_p$ and $r \in \mathbb{Z}_q$.
- Output: $\pi_{\mathsf{sch}} = (e, z) \in \mathbb{Z}_s \times \mathbb{Z}_q$.

$\mathsf{VerifySch}((p, q, g, \mathbf{c}), \pi_{\mathsf{sch}}, \mathtt{aux})$**:** Execute steps of algorithm verify from Defn. 12.

- Inputs: statement $st = ((p, q, g), c_0)$, proof $\pi_{\mathsf{sch}} = (e, z)$, and $\mathtt{aux}' = \mathtt{aux} \cup \{c_1\}$, where $c_0 \in \mathbb{Q}_p$ and $(e, z) \in \mathbb{Z}_s \times \mathbb{Z}_q$.
- Output: either 1 (accept) or 0 (reject).

7.3.4 Plaintext Equality proof system

The plaintext equality proof system $\mathsf{EqEnc} = (\mathsf{ProveEqEnc}, \mathsf{VerifyEqEnc})$ is based on the Chaum-Pedersen proof system [17]. It is a pre-image proof system with group homomorphism $\phi : \mathbb{Z}_q^2 \to \mathbb{Q}_p^3$ given by $\phi(r, \bar{r}) = (g^r, g^{\bar{r}}, h^r/\bar{h}^{\bar{r}})$. The description of ϕ is given by the group parameters (p, q) and the group elements $g, h, \bar{h}$, all in $\mathbb{Q}_p$.

Let two ElGamal ciphertexts $\mathbf{c} = (c_0, c_1)$, $\bar{\mathbf{c}} = (\bar{c}_0, \bar{c}_1)$ under two public keys $h, \bar{h}$ of the same plaintext, where ElGamal is instantiated over $\mathbb{Q}_p$ with generator g. This proof system proves that $(c_0, \bar{c}_0, c_1/\bar{c}_1) = \phi(r, \bar{r})$, where r, and $\bar{r}$ are the random exponents used to encrypt $\mathbf{c}$ and $\bar{\mathbf{c}}$, respectively. That is, it proves that $\mathbf{c}, \bar{\mathbf{c}}$ encrypt the same plaintext. The algorithms are as follows:

$\mathsf{ProveEqEnc}((p, q, g, h, \bar{h}, \mathbf{c}, \bar{\mathbf{c}}), (r, \bar{r}), \mathsf{aux})$**:** Execute steps of algorithm prove from Defn. 12.

- Inputs: statement $st = ((p, q, g, h, \bar{h}), (c_0, \bar{c}_0, c_1/\bar{c}_1))$, witness $w = (r, \bar{r})$, where $(c_0, \bar{c}_0, c_1/\bar{c}_1) \in \mathbb{Q}_p^3$ and $(r, \bar{r}) \in \mathbb{Z}_q^2$.
- Output: $\pi_{\mathsf{EqEnc}} = (e, z, \bar{z}) \in \mathbb{Z}_s \times \mathbb{Z}_q^2$.

$\mathsf{VerifyEqEnc}((g, h, \bar{h}, \mathbf{c}, \bar{\mathbf{c}}), \pi_{\mathsf{EqEnc}}, \mathsf{aux})$**:** Execute steps of algorithm verify from Defn. 12.

- Inputs: statement $st = ((p, q, g, h, \bar{h}), (c_0, \bar{c}_0, c_1/\bar{c}_1))$, proof $\pi_{\mathsf{EqEnc}} = (e, z, \bar{z})$, where $(c_0, \bar{c}_0, c_1/\bar{c}_1) \in \mathbb{Q}_p^2$ and $(e, z, \bar{z}) \in \mathbb{Z}_s \times \mathbb{Z}_q^2$.
- Output: either 1 (accept) or 0 (reject).

7.3.5 Decryption proof system.

The correct decryption proof system $\mathsf{dec} = (\mathsf{ProveDec}, \mathsf{VerifyDec})$ is also based on the Chaum-Pedersen proof system [17]. It is a pre-image proof system with group homomorphism $\phi : \mathbb{Z}_q^n \to \mathbb{Q}_p^{2n}$ given by $\phi(\mathbf{x}) = (g^{x_1}, \ldots, g^{x_n}, c_0^{x_1}, \ldots, c_0^{x_n})$. The description of ϕ is given by the group parameters (p, q) and by g and c_0, both elements in $\mathbb{Q}_p$.

Let ElGamal instantiated over a cyclic group $\mathbb{Q}_p$ with generator g, and let a multi-recipient ciphertext $\mathbf{c} = (c_0, \bar{\mathbf{c}}) = (c_0, c_1, \ldots, c_n)$ under public keys $\mathbf{h} = (h_1, \ldots, h_n)$, with $h_i = g^{x_i}$, and plaintexts $\mathbf{m} = (m_1, \ldots, m_n)$ resulting from decrypting $\mathbf{c}$ using the secret keys $\mathbf{x} = (x_1, \ldots, x_n)$. This proof system proves that $(\mathbf{h}, \bar{\mathbf{c}}/\mathbf{m}) = \phi(\mathbf{x})$, that is, it proves correct decryption of $\mathbf{c}$ to $\mathbf{m}$ under secret keys $\mathbf{x}$. The algorithms are as follows:

$\mathsf{ProveDec}((p, q, g, \mathbf{h}, \mathbf{c}, \mathbf{m}), \mathbf{x}, \mathsf{aux})$**:** Execute steps of algorithm prove from Definition 12.

- Inputs: statement $st = ((p, q, g, c_0), (\mathbf{h}, \bar{\mathbf{c}}/\mathbf{m}))$, witness $w = \mathbf{x}$, where $(\mathbf{h}, \bar{\mathbf{c}}/\mathbf{m}) \in \mathbb{Q}_p^{2n}$ and $\mathbf{x} \in \mathbb{Z}_q^n$.
- Output: $\pi_{\mathsf{dec}} = (e, \mathbf{z}) \in \mathbb{Z}_s \times \mathbb{Z}_q^n$.

$\mathsf{VerifyDec}((p, q, g, \mathbf{h}, \mathbf{c}, \mathbf{m}), \pi_{\mathsf{dec}}, \mathtt{aux})$**:** Execute steps of algorithm verify from Definition 12.

- Inputs: statement $st = ((p, q, g, c_0), (\mathbf{h}, \bar{\mathbf{c}}/\mathbf{m}))$, proof $\pi_{\mathsf{dec}} = (e, \mathbf{z})$, where $(\mathbf{h}, \bar{\mathbf{c}}/\mathbf{m}) \in \mathbb{Q}_p^{2n}$ and $(e, \mathbf{z}) \in \mathbb{Z}_s \times \mathbb{Z}_q^n$.
- Output: either 1 (accept) or 0 (reject).

Observe that this proof system also proves correctness of *partial* decryptions: in this situation the ciphertext $\mathbf{c}$ is encrypted under public key $\prod_{j=1}^{m} \mathbf{h}_j$, and the partially decrypted ciphertext is $\mathbf{d} = (c_0, \mathbf{m})$. Then, the party P_j holding *shares* $(\mathbf{h}_j, \mathbf{x}_j)$ of the public and private keys proves that $(\mathbf{h}_j, \bar{\mathbf{c}}/\mathbf{m}) = \phi(\mathbf{x}_j)$.

8 Verifiable mixnet

8.1 Homomorphic commitment scheme

The verifiable proof of a shuffle correctness requires a computationally binding homomorphic commitment scheme i.e. a computationally binding commitment scheme for which it holds that $\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(a+b; r+s) = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(a; r)\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(b; s)$ for messages a, b, a commitment key $\mathsf{ck}_{\mathsf{mix}}$ and randomnesses r, s.

We use the generalization of the Pedersen commitment scheme [40] with a commitment key $\mathsf{ck}_{\mathsf{mix}} = (\mathbb{G}, G_1, \ldots, G_n, H)$, where $G_1, \ldots, G_n, H$ are the random generators of the group $\mathbb{G}$ generated by the key generation algorithm $\mathcal{K}$.

To commit to a vector of n elements $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_q^n$, we pick randomness $r \in \mathbb{Z}_q$ and compute $\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}; r) = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(a_1, \ldots, a_n; r) = H^r \prod_{i=1}^{n} G_i^{a_i}$. We can also commit to less than n elements; this is done by setting the remaining entries to 0.

A commitment to a matrix $A \in \mathbb{Z}_q^{n \times m}$ with columns $\mathbf{a}_1, \ldots, \mathbf{a}_m$ is computed as $\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(A; \mathbf{r}) = (\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}_1; r_1), \ldots, \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}_m; r_m))$. Similarly a commitment to a large vector $\mathbf{a} \in \mathbb{Z}_q^N$, where $N = m \times n$ (that is in fact a matrix converted into to vector), is computed as $\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}; \mathbf{r}) = (\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(a_1, \ldots, a_n; r_1), \ldots, \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(a_{(m-1)n+1}, \ldots, a_N; r_m))$.

Also we define $\mathbf{c}^{\mathbf{b}} = (c_1, \ldots, c_m)^{(b_1, \ldots, b_m)^T} = \prod_{j=1}^{m} c_j^{b_j}$ and for matrix B with columns $\mathbf{b_1}, \ldots, \mathbf{b_m}$ we define $\mathbf{c}^B = (\mathbf{c}^{\mathbf{b_1}}, \ldots, \mathbf{c}^{\mathbf{b_m}})$.

8.2 Mixing proof system

Informally, a shuffle or mixing of ciphertexts $C_1, \ldots, C_N$ is a set of ciphertexts $C'_1, \ldots, C'_N$ with the same plaintexts in permuted order and a different encryption randomness. Therefore, to prove the correctness of a shuffle , one needs to prove knowledge of the permutation π and the randomness ρ such that $\{C'_i\}_{i=1}^N = \{C_{\pi(i)} \cdot \mathsf{Enc}(1, \mathsf{pk}_{\mathsf{mix}}; \rho_i)\}_{i=1}^N$ without revealing π and ρ to the verifier.

In our protocol we use an honest verifier zero-knowledge argument for correctness of a shuffle of homomorphic encryptions proposed by Stephanie Bayer and Jens Groth [2]. Following their notation, we write $\mathbb{G}$ for the group used in the commitment scheme, and $\mathbb{H}$ for the ciphertext space.

8.2.1 $\mathsf{Mix}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C})$

This algorithm takes as input a mixing $\mathtt{pk}_{\mathsf{mix}}$ and commitment $\mathtt{ck}_{\mathsf{mix}}$ public keys and a vector of initial ciphertexts $\mathbf{C}$, then it permutes and re-encrypts ciphertexts to generate the resulting vector $\mathbf{C}'$ and proves shuffle correctness by computing a zero-knowledge proof of a shuffle $\pi_{\mathsf{mix},}$:

1. Sample permutation π and randomness ρ.
2. Re-encrypt and permute ciphertexts $\mathbf{C}$ to get $\mathbf{C}'$.
3. Compute proof of the shuffle $\pi_{\mathsf{mix},} \leftarrow \mathsf{ShuffleArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \pi, \rho)$.

Finally it returns the shuffled and permuted ciphertexts $\mathbf{C}'$ and a proof of shuffle correctness $\pi_{\mathsf{mix},}$.

8.2.2 $\mathsf{ShuffleArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \pi, \rho)$

This algorithm is a public coin perfect SHVZK argument of knowledge of a permutation $\pi \in \Sigma_N$ and randomness $\rho \in \mathbb{Z}_q^N$ such that for given ciphertexts $\mathbf{C} \in \mathbb{H}^N$ and $\mathbf{C}' \in \mathbb{H}^N$ it holds that $\mathbf{C}' = \mathsf{Enc}(\mathbf{1}, \mathsf{pk})\mathbf{C}_\pi$. An interactive proof is computed as described in Figure 7 with the following input:

- CRS: $\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}$
- Statement: $\mathbf{C}, \mathbf{C}'$
- Witness: π, ρ

Note, that for non-interactive case, challenges are computed as follows:

- $x \leftarrow \mathsf{H}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathsf{InitM}_{\mathsf{ShufArg}}, \mathbf{C}', \mathbf{C}, \mathsf{aux}) = \mathsf{H}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, \mathbf{C}', \mathbf{C}, \mathsf{aux})$.
- $y \leftarrow \mathsf{H}(\mathsf{Answer1}_{\mathsf{ShufArg}}, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathsf{InitM}_{\mathsf{ShufArg}}, \mathbf{C}', \mathbf{C}, \mathsf{aux}) = \mathsf{H}(\mathbf{c}_B, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, \mathbf{C}', \mathbf{C}, \mathsf{aux})$[5].
- $z \leftarrow \mathsf{H}(1, \mathsf{Answer1}_{\mathsf{ShufArg}}, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathsf{InitM}_{\mathsf{ShufArg}}, \mathbf{C}', \mathbf{C}, \mathsf{aux}) = \mathsf{H}(1, \mathbf{c}_B, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, \mathbf{C}', \mathbf{C}, \mathsf{aux})$[6], where $\mathsf{InitM}_{\mathsf{ShufArg}} = (\mathbf{c}_A)$ is an initial message of a shuffle argument and $\mathsf{Answer1}_{\mathsf{ShufArg}} = (\mathbf{c}_B)$ is a first answer of the Shuffle product argument computed as described in Figure 7.

The output is $(\mathbf{c}_A, \mathbf{c}_B, \pi_{\mathsf{Prod}}, \pi_{\mathsf{MultiExp}})$.

8.2.3 $\mathsf{MultiExpArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, (\mathbf{C_1}, \ldots, \mathbf{C_m}), C, \mathbf{c}_A, \{\mathbf{a}_j\}_{j=1}^m, \mathbf{r}, \rho)$

This algorithm is a public coin argument of knowledge of openings of commitments $\mathbf{c}_A$ to $A = \{\mathbf{a}_j\}_{j=1}^m$ such that $C = \mathsf{Enc}(1; \mathtt{pk}_{\mathsf{mix}}; \rho) \prod_{i=1}^m \mathbf{C}_i^{\mathbf{a}_i}$ and $\mathbf{c}_A = \mathsf{com}_{\mathtt{ck}_{\mathsf{mix}}}(A; \mathbf{r})$. An interactive proof is computed as described in Figure 8 with the following input:

[5] We don't need to include x explicitly since it deterministically depends on values already included in the hash.
[6] We don't need to include x and y explicitly since they deterministically depend on values already included in the hash.

- CRS: $\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}$
- Statement: $(\mathbf{C_1}, \ldots, \mathbf{C_m}), C, \mathbf{c}_A$
- Witness: $\{\mathbf{a_j}\}_{j=1}^{m}, \mathbf{r}, \rho$

Note, that for non-interactive case, the challenge are computed as follows:

- $x \leftarrow \mathsf{H}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathsf{InitM}_{\mathsf{MultiExp}}, \mathbf{c}_A, C, \mathbf{C_1}, \ldots, \mathbf{C_m}, \mathsf{aux}) = \mathsf{H}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \{E_k\}_{k=0}^{2m-1}, \{c_{B_k}\}_{k=0}^{2m-1}, c_{A_0}, \mathbf{c}_A, C, \mathbf{C_1}, \ldots, \mathbf{C_m}, \mathsf{aux})$, where $\mathsf{InitM}_{\mathsf{MultiExp}} = (\{E_k\}_{k=0}^{2m-1}, \{c_{B_k}\}_{k=0}^{2m-1}, c_{A_0})$ is an initial message of a multi-exponentiation argument computed as described in Figure 8.

The output is $\pi_{\mathsf{MultiExp}} = (c_{A_0}, \{c_{B_k}\}_{k=0}^{2m-1}, \{E_k\}_{k=0}^{2m-1}, \mathbf{a}, r, b, s, \tau)$.

8.2.4 ProductArg$(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_A, b, A, \mathbf{r})$

This algorithm is an argument that a set of committed values have a particular product i.e. given a commitment $\mathbf{c}_A$ to $A = \{a_{ij}\}_{i,j=1}^{n,m}$ and a value b it gives an argument of knowledge for $\prod_{i=1}^{n}\prod_{j=1}^{m} a_{ij} = b$. An interactive proof is computed as described in Figure 9 with the following input:

- CRS: $\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}$
- Statement: $\mathbf{c}_A, b$
- Witness: $A, \mathbf{r}$

Note, that this proof does not require a challenge.

The output is $\pi_{\mathsf{Prod}} = (c_b, \pi_{\mathsf{HadPA}}, \pi_{\mathsf{SingleVPA}})$.

8.2.5 HadamardProdArg$(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_A, c_b, (\mathbf{a_1}, \ldots, \mathbf{a_m}), \mathbf{r}, \mathbf{b}, s)$

This algorithm is an argument of knowledge of the openings $a_{11}, \ldots, a_{nm}$ and $b_1, \ldots, b_n$ to the commitments $\mathbf{c}_A$ and c_b s.t. $b_i = \prod_{j=1}^{m} a_{ij}$ for $i = 1, \ldots, n$. An interactive proof is computed as described in Figure 10 with the following input:

- CRS: $\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}$
- Statement: $\mathbf{c}_A, c_b$
- Witness: $(\mathbf{a_1}, \ldots, \mathbf{a_m}), \mathbf{r}, \mathbf{b}, s$

Note, that for non-interactive case, challenges are computed as follows:

- $x \leftarrow \mathsf{H}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathsf{InitM}_{\mathsf{HadPA}}, c_b, \mathbf{c}_A, \mathsf{aux}) = \mathsf{H}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_B, c_b, \mathbf{c}_A, \mathsf{aux})$
- $y \leftarrow \mathsf{H}(1, \mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathsf{InitM}_{\mathsf{HadPA}}, c_b, \mathbf{c}_A, \mathsf{aux}) = \mathsf{H}(1, \mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_B, c_b, \mathbf{c}_A, \mathsf{aux})$[5], where $\mathsf{InitM}_{\mathsf{HadPA}} = (\mathbf{c}_B)$ is an initial message of Hadamard product argument computed as described in Figure 10.

The output is $\pi_{\mathsf{HadPA}} = (\mathbf{c}_B, \pi_{\mathsf{ZeroArg}})$.

8.2.6 $\mathsf{ZeroArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, \mathbf{c}_B, \star, A, B, \mathbf{r}, \mathbf{s})$

This algorithm is an argument of knowledge of the committed values $\mathbf{a}_1, \mathbf{b}_0, \ldots, \mathbf{a}_m, \mathbf{b}_{m-1}$ s.t. $0 = \sum_{i=1}^{m} \mathbf{a}_i \star \mathbf{b}_{i-1}$. An interactive proof is computed as described in Figure 11 with the following input:

- CRS: $\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}$
- Statement: $\mathbf{c}_A, \mathbf{c}_B$ and a specification of a bilinear map $\star : \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_q$
- Witness: $A, B, \mathbf{r}, \mathbf{s}$

Note, that for non-interactive case, challenges are computed as follows:

- $x \leftarrow \mathsf{H}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathsf{InitM}_{\mathsf{ZeroArg}}, \mathbf{c}_B, \mathbf{c}_A, \mathtt{aux}) = \mathsf{H}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_D, c_{B_m}, c_{A_0}, \mathbf{c}_B, \mathbf{c}_A, \mathtt{aux})$,where $\mathsf{InitM}_{\mathsf{ZeroArg}} = (\mathbf{c}_D, c_{B_m}, c_{A_0})$ is an initial message of a zero argument computed as described in Figure 11.

The output is $\pi_{\mathsf{ZeroArg}} = (c_{A_0}, c_{B_m}, \mathbf{c}_D, \mathbf{a}, \mathbf{b}, r, s, t)$.

8.2.7 $\mathsf{SingleValueProdArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, c_a, b, \mathbf{a}, r)$

This algorithm is an argument of knowledge of the opening $a_1, \ldots, a_n, r$ s.t. $c_a = \mathsf{com}_{\mathtt{ck}_{\mathsf{mix}}}(\mathbf{a}; r)$ and $b = \prod_{i=1}^{n} a_i$. An interactive proof is computed as described in Figure 12 with the following input:

- CRS: $\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}$
- Statement: c_a, b
- Witness: $\mathbf{a}, r$

Note, that for non-interactive case, challenges are computed as follows:

- $x \leftarrow \mathsf{H}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathsf{InitM}_{\mathsf{SingleVPA}}, b, c_a, \mathtt{aux}) = \mathsf{H}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, c_\Delta, c_\sigma, c_d, b, c_a, \mathtt{aux})$, where $\mathsf{InitM}_{\mathsf{SingleVPA}} = (c_\Delta, c_\sigma, c_d)$ is an initial message of a single value product argument computed as described in Figure 12.

The output is $\pi_{\mathsf{SingleVPA}} = (c_d, c_\sigma, c_\Delta, \tilde{a}_1, \tilde{b}_1, \ldots, \tilde{a}_n, \tilde{b}_n, \tilde{r}, \tilde{s})$.

8.2.8 Fiat-Shamir heuristic in Shuffle Argument

In the Fiat-Shamir heuristic the prover computes the public-coin challenges with a cryptographic hash-function instead of interacting with a verifier. We apply the Fiat-Shamir heuristic of the following form: $x = \mathsf{H}(\mathsf{CRS}, \mathsf{Statement}, \mathsf{InitMessage}, \mathtt{aux})$ to each internal public-coin interactive sub-argument of the Shuffle Argument independently. Note that H is a hash function with domain separation, thus the list of variables is encoded uniquely into a binary string.

For the Hadamard Product Argument and the Shuffle Argument, where the Verifier sends two challenges x, y, the second challenge is generated as $y = \mathsf{H}(1, \mathsf{CRS}, \mathsf{InitMessage}, \mathsf{Statement}, \mathtt{aux})$ to ensure that y differs from x.

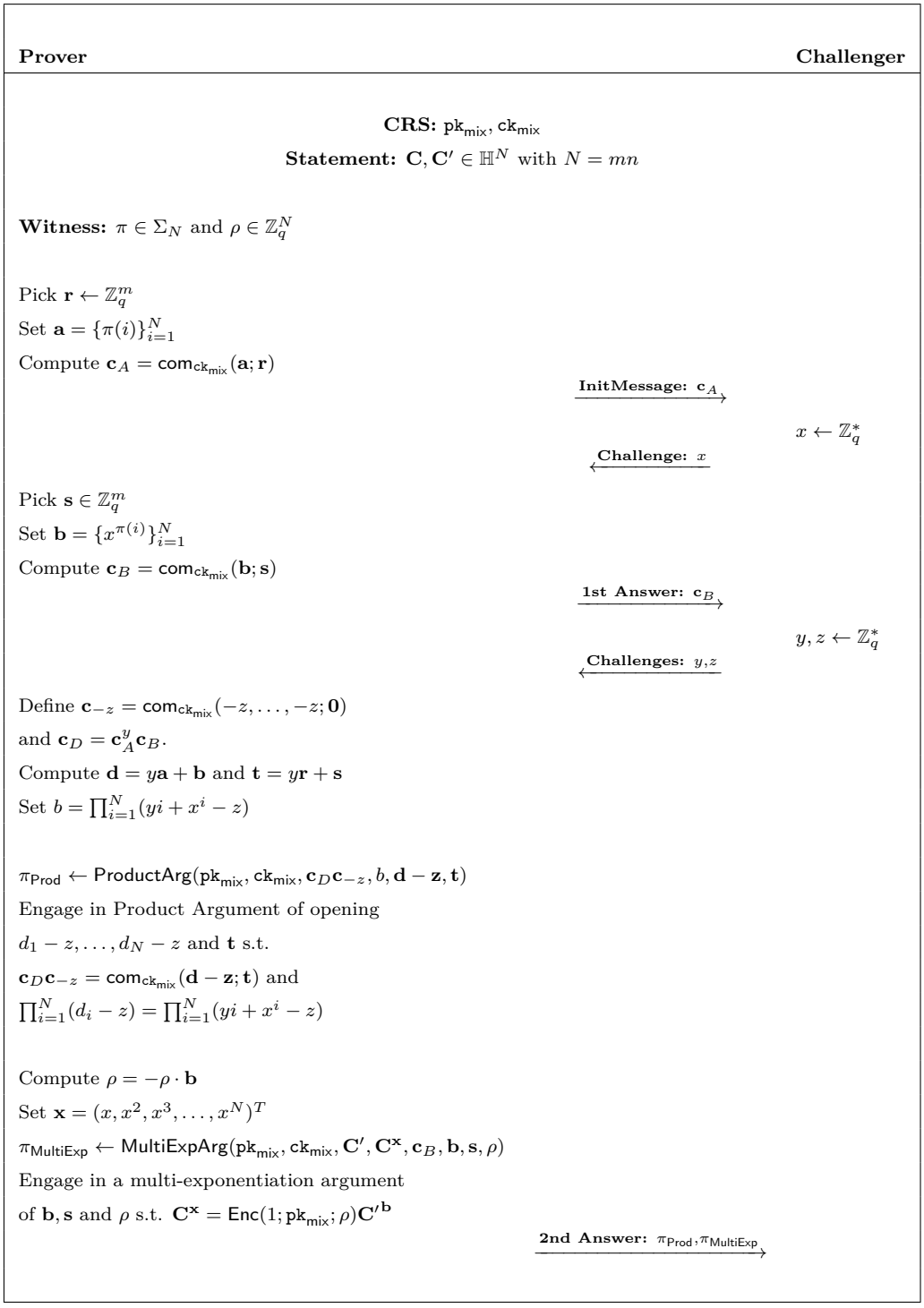

Figure 7: Shuffle argument: an argument of knowledge of a permutation π and randomness ρ s.t. $\mathbf{C}' = \mathsf{Enc}(\mathbf{1}; \mathsf{pk}_{\mathsf{mix}}; \rho)\mathbf{C}_\pi$.

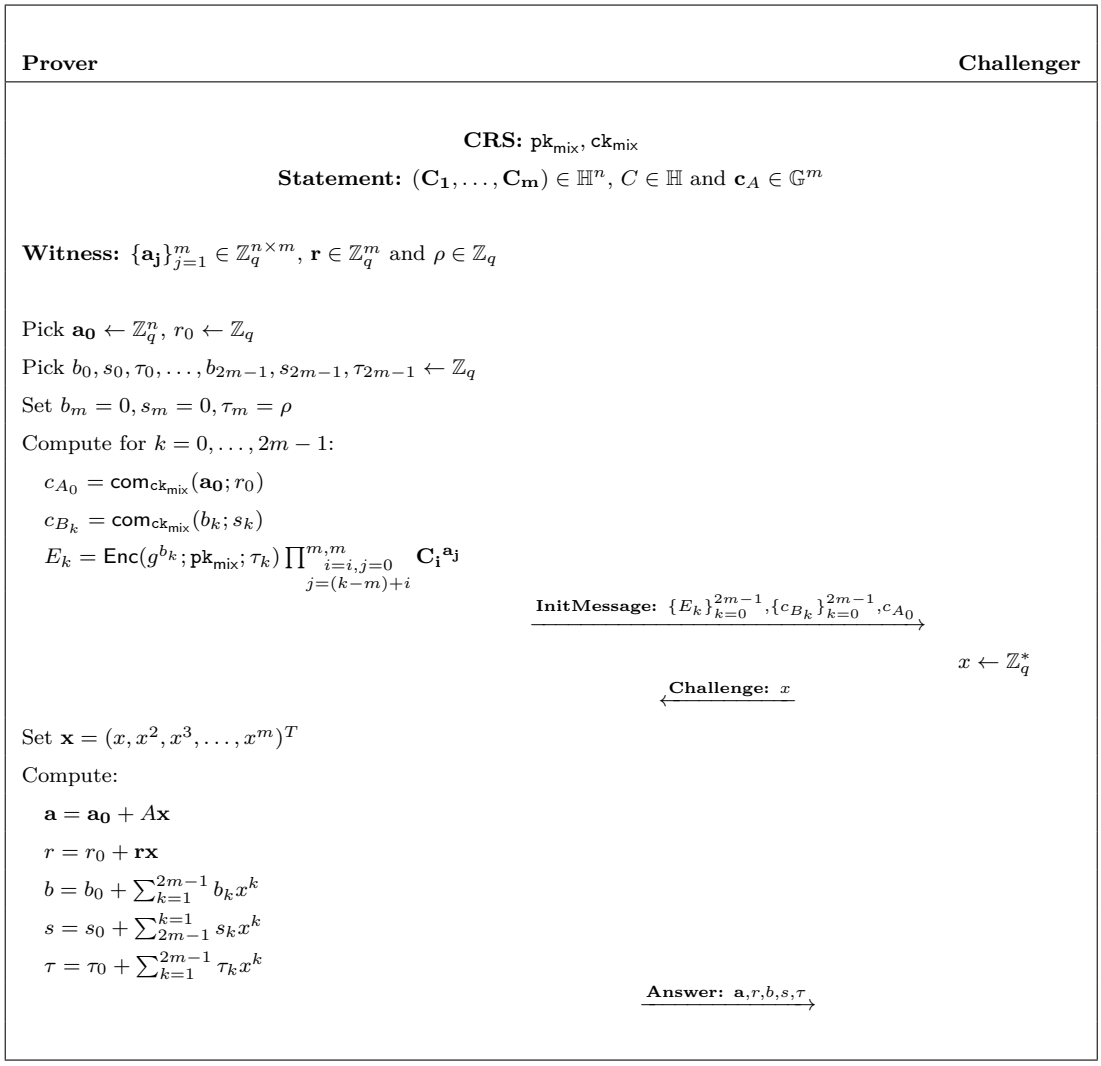

Figure 8: Multi-exponentiation argument: an argument of knowledge of openings of commitment $\mathbf{c}_A$ to $A = \{a_{ij}\}_{i,j=1}^{n,m}$ s.t. $C = \mathsf{Enc}(1; \mathsf{pk_{mix}}; \rho) \prod_{i=1}^m \mathbf{C_i}^{\mathbf{a_i}}$ and $\mathbf{c}_A = \mathsf{com}_{\mathsf{ck_{mix}}}(A; \mathbf{r})$.

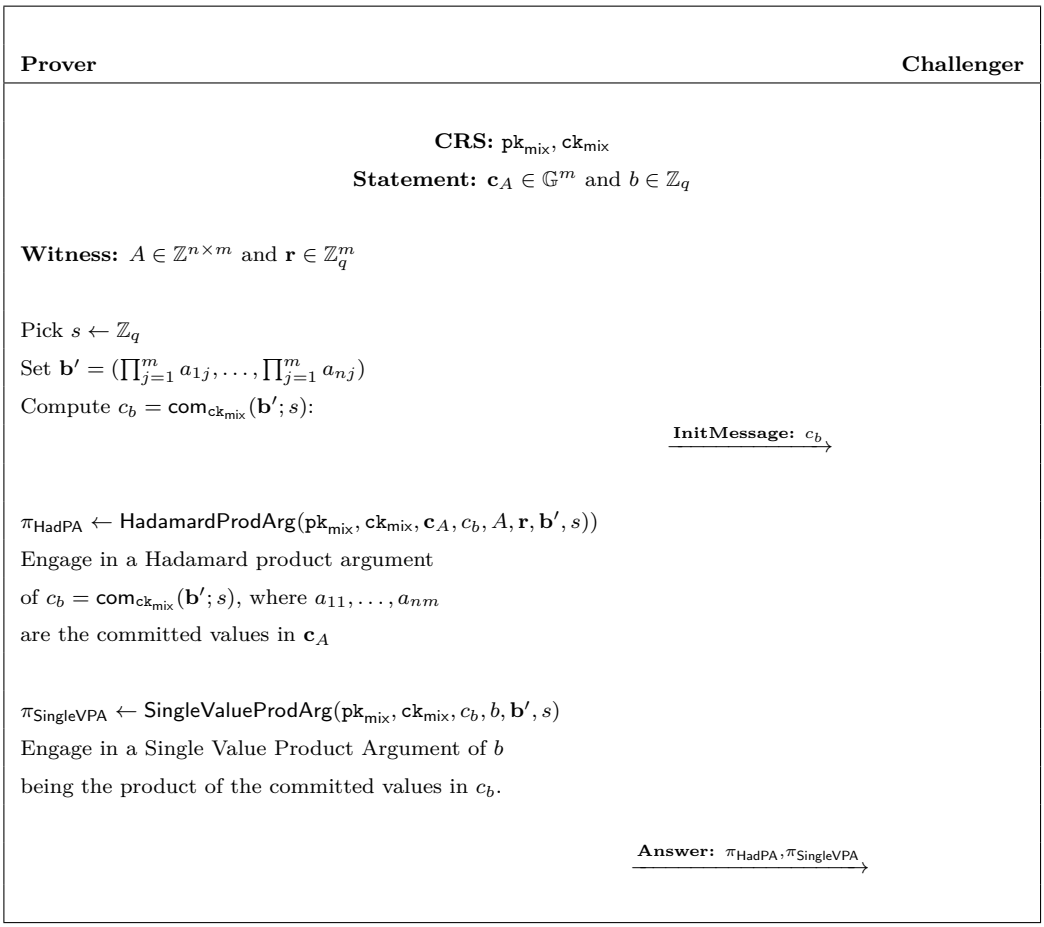

Figure 9: Product argument: an argument of knowledge of openings $a_{11}, \ldots, a_{nm}, r_1, \ldots, r_m$ to a given commitment $\mathbf{c}_A$ s.t. $\prod_{i=1}^n \prod_{j=1}^m a_{ij} = b$.

Prover	**Challenger**

CRS: $\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}$

Statement: $\mathbf{c}_A \in \mathbb{G}^m$ and $c_b \in \mathbb{G}$

Witness: $\mathbf{a_1}, \ldots, \mathbf{a_m}, \mathbf{b} \in \mathbb{Z}^n$, $\mathbf{r} \in \mathbb{Z}_q^m$ and $s \in \mathbb{Z}_q$

Define:

$\mathbf{b}_1 = \mathbf{a}_1$

$\mathbf{b}_2 = \mathbf{a}_1\mathbf{a}_2$

$\ldots$

$\mathbf{b_{m-1}} = \mathbf{a_1} \cdots \mathbf{a}_{m-1}$

$\mathbf{b_m} = \mathbf{b}$

Pick $s_2, \ldots, s_{m-1} \leftarrow \mathbb{Z}_q$

Compute:

$c_{B_2} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{b}_2; s_2)$

$\ldots$

$c_{B_{m-1}} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{b}_{m-1}; s_{m-1})$

Define $s_1 = r_1$ and $s_m = s$

Set $c_{B_1} = c_{A_1}$ and $c_{B_m} = c_b$

Set $\mathbf{c}_B = (c_{B_1}, \ldots, c_{B_m})$

$\xrightarrow{\textbf{InitMessage: } \mathbf{c}_B}$

$x, y \leftarrow \mathbb{Z}_q^*$

$\xleftarrow{\textbf{Challenges: } x,y}$

Define the bilinear map $\star : \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_q$

by $(a_1, \ldots, a_n)^T \star (d_1, \ldots, d_n)^T = \sum_{j=1}^n a_j d_j y^j$

Define $c_{D_i} = c_{B_i}^{x^i}$, $c_D = \prod_{i=1}^{m-1} c_{B_{i+1}}^{x^i}$

and $c_{-1} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(-\mathbf{1}; 0)$

Set:

$\mathbf{d}_1 = x\mathbf{b}_1, t_1 = xs_1$

$\ldots$

$\mathbf{d}_{m-1} = x^{m-1}\mathbf{b}_{m-1}, t_{m-1} = x^{m-1}s_{m-1}$

$\mathbf{d} = \sum_{i=1}^{m-1} x^i \mathbf{b}_{i+1}, t = \sum_{i=1}^{m-1} x^i s_{i+1}$

Define:

$D = (\mathbf{d}_1, \ldots, \mathbf{d}_{m-1}, \mathbf{d})$

$\mathbf{t} = (t_1, \ldots, t_{m-1}, t)$

$F = (\mathbf{a}_2, \ldots, \mathbf{a}_m, -\mathbf{1})$

$\tau = (r_2, \ldots, r_m, 0)$

$\mathbf{c}_f = (c_{A_2}, \ldots, c_{A_m}, c_{-1})$

$\mathbf{c}_d = (c_{D_1}, \ldots, c_{D_{m-1}, c_D})$

$\pi_{\mathsf{ZeroArg}} \leftarrow \mathsf{ZeroArg}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_f, \mathbf{c}_d, \star, F, D, \tau, \mathbf{t})$

Engage in a Zero argument for

the committed values in $\mathbf{c}_f, \mathbf{c}_d$ satisfying

$0 = \sum_{i=1}^{m-1} \mathbf{a}_{i+1} \star \mathbf{d}_i - \mathbf{1} \star \mathbf{d}$

$\xrightarrow{\textbf{Answer: } \pi_{\mathsf{ZeroArg}}}$

Figure 10: Hadamard argument: an argument of knowledge of the openings $a_{11}, \ldots, a_{nm}$ and $b_1, \ldots, b_n$ to the commitments $\mathbf{c}_A$ and c_b s.t. $b_i = \prod_{j=1}^m a_{ij}$ for $i = 1, \ldots, n$.

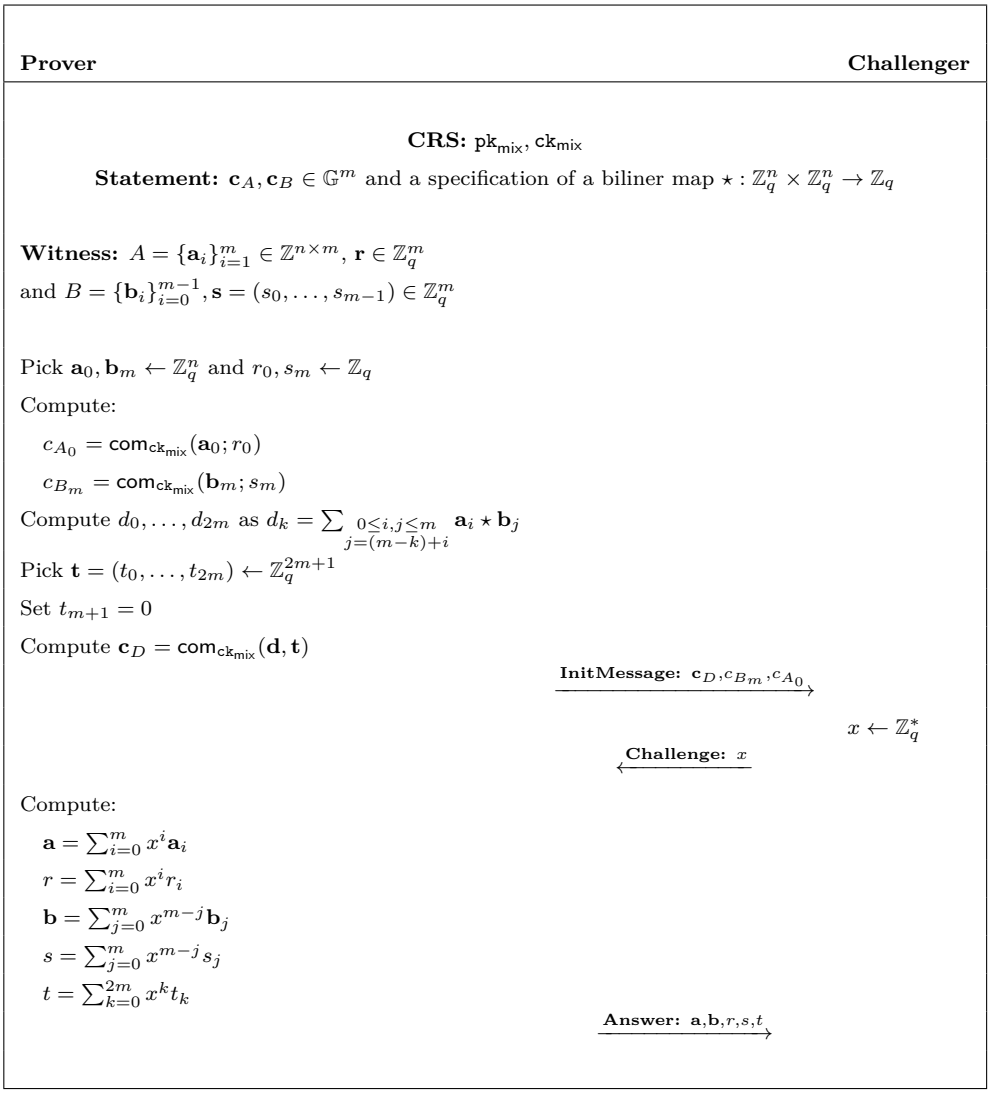

Figure 11: Zero argument: an argument of knowledge of the committed values $\mathbf{a}_1, \mathbf{b}_0, \ldots, \mathbf{a}_m, \mathbf{b}_{m-1}$ s.t. $0 = \sum_{i=1}^m \mathbf{a}_i \star \mathbf{b}_{i-1}$.

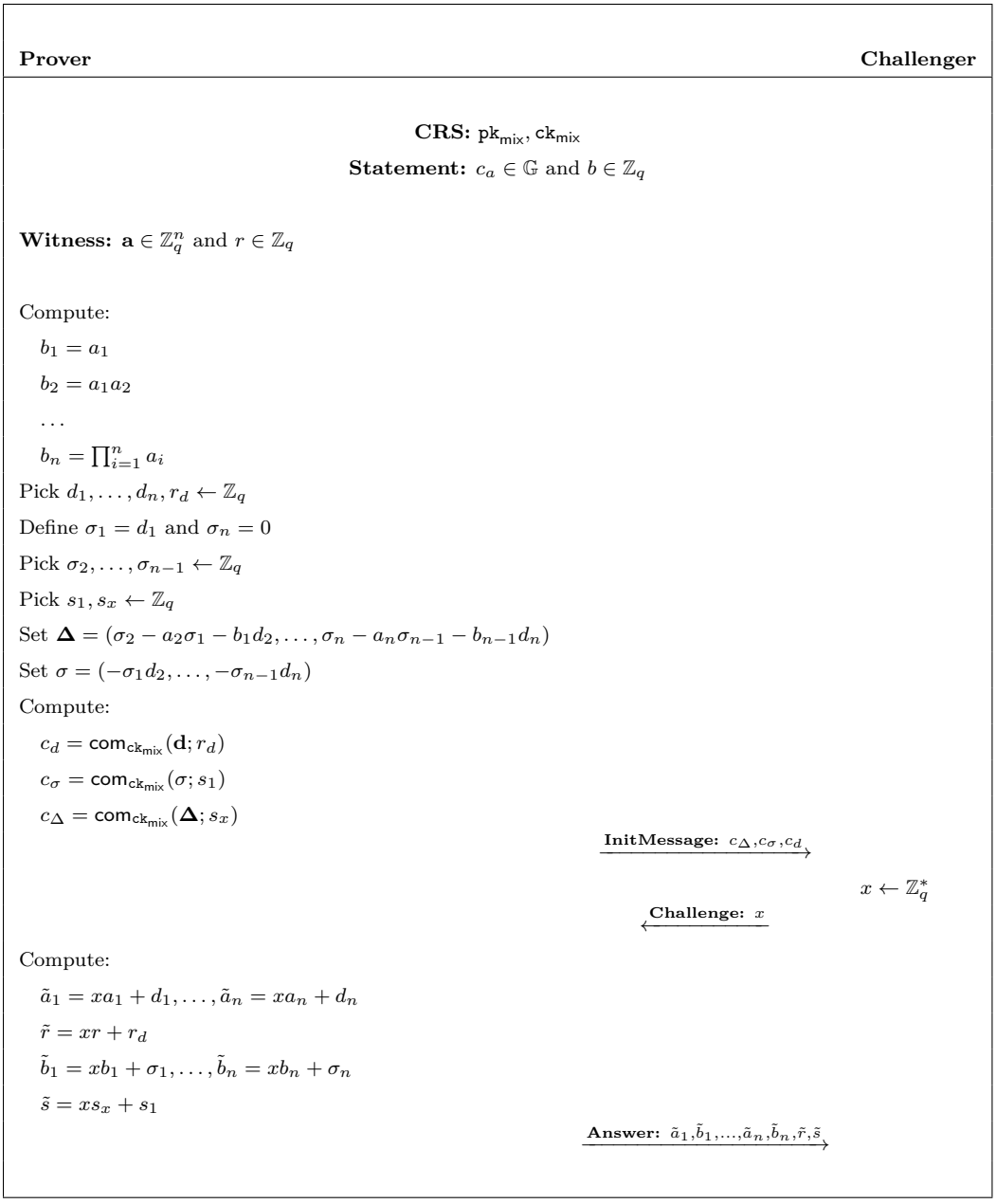

Figure 12: Single value product argument: an argument of knowledge of the opening $a_1, \ldots, a_n, r$ s.t. $c_a = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}; r)$ and $b = \prod_{i=1}^n a_i$.

8.2.9 Special case of mixing

If the ciphertexts cannot be arranged in a matrix (e.g. the number of votes N is a prime number), then the mixing process is executed for $n = N$ and $m = 1$ parameters. In such cases, the Hadamard Product argument becomes redundant and the single value product argument can be executed directly. Alternatively, one can add a vector containing unity elements and compute commitment to that vector $\mathsf{com}_{\mathtt{ck}_{\mathsf{mix}}}(\mathbf{1}; 0)$, then set $m = 2$ and execute the Hadamard Product as usual for $A = (\mathbf{a_1}, \mathbf{1})$.

8.3 $\mathsf{MixVerify}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \pi_{\mathsf{mix},})$

This algorithm takes as input a mixing $\mathtt{pk}_{\mathsf{mix}}$ and commitment $\mathtt{ck}_{\mathsf{mix}}$ public keys, a vector of initial ciphertexts $\mathbf{C}$ and permuted ciphertexts $\mathbf{C}'$ and verifies the mixing procedure as follows:

1. If $\mathsf{verifyShuffleArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \pi_{\mathsf{mix},})$ outputs 0, aborts and returns $\perp$.
2. Otherwise outputs $\top$.

8.4 $\mathsf{verifyShuffleArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \pi_{\mathsf{mix},})$

This algorithm takes as input a mixing $\mathtt{pk}_{\mathsf{mix}}$ and commitment $\mathtt{ck}_{\mathsf{mix}}$ public keys, a vector of initial ciphertexts $\mathbf{C}$ and permuted ciphertexts $\mathbf{C}'$ and verifies the proof of the shuffle correctness $\pi_{\mathsf{mix},}$ as follows:

1. Parses $\pi_{\mathsf{mix},}$ as $(\mathbf{c}_A, \mathbf{c}_B, \pi_{\mathsf{Prod}}, \pi_{\mathsf{MultiExp}})$.
2. Computes challenges:
   - $x \leftarrow \mathsf{H}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, \mathbf{C}', \mathbf{C}, \mathtt{aux})$.
   - $y \leftarrow \mathsf{H}(\mathbf{c}_B, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, \mathbf{C}', \mathbf{C}, \mathtt{aux})$.
   - $z \leftarrow \mathsf{H}(1, \mathbf{c}_B, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, \mathbf{C}', \mathbf{C}, \mathtt{aux})$.
3. Computes the following values:
   (a) $\mathbf{c}_D = \mathbf{c}_A^y \mathbf{c}_B$.
   (b) $\mathbf{c}_{-z} = \mathsf{com}_{\mathtt{ck}_{\mathsf{mix}}}(-z, \ldots, -z; \mathbf{0})$.
   (c) $\mathbf{C}^{\mathbf{x}}$.
   (d) $b = \prod_{i=1}^{N}(yi + x^i - z)$.
4. Verifies:
   (a) $\mathbf{c}_A, \mathbf{c}_B \in \mathbb{G}^m$.
   (b) checks that $\mathsf{verifyMultiExpArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}', \mathbf{C}^{\mathbf{x}}, b, \pi_{\mathsf{MultiExp}})$ outputs 1.

(c) checks that $\mathsf{verifyProductArg}(\mathsf{pk_{mix}}, \mathsf{ck_{mix}}, \mathbf{c}_D\mathbf{c}_{-z}, b, \pi_{\mathsf{Prod}})$ outputs 1.

If and only if all verifications above are successful, the algorithm outputs 1. Otherwise, it outputs 0.

8.5 $\mathsf{verifyMultiExpArg}(\mathsf{pk_{mix}}, \mathsf{ck_{mix}}, (\mathbf{C_1}, \ldots, \mathbf{C_m}), C, \mathbf{c}_A, \pi_{\mathsf{MultiExp}})$

This algorithm verifies a public coin argument of knowledge of openings of commitments $\mathbf{c}_A$ to $A = \{\mathbf{a}_j\}_{j=1}^m$ such that $C = \mathsf{Enc}(1; \mathsf{pk_{mix}}; \rho) \prod_{i=1}^m \mathbf{C}_i^{\mathbf{a}_i}$ and $\mathbf{c}_A = \mathsf{com}_{\mathsf{ck_{mix}}}(A; \mathbf{r})$.

1. Parses π_{MultiExp} as $(c_{A_0}, \{c_{B_k}\}_{k=0}^{2m-1}, \{E_k\}_{k=0}^{2m-1}, \mathbf{a}, r, b, s, \tau)$.
2. Computes challenge $x \leftarrow \mathsf{H}(\mathsf{pk_{mix}}, \mathsf{ck_{mix}}, \{E_k\}_{k=0}^{2m-1}, \{c_{B_k}\}_{k=0}^{2m-1}, c_{A_0}, \mathbf{c}_A, C, \mathbf{C_1}, \ldots, \mathbf{C_m}, \mathsf{aux})$
3. Checks that:
   (a) $c_{A_0}, \mathbf{c}_{B_0}, \ldots, \mathbf{c}_{B_{2m-1}} \in \mathbb{G}$.
   (b) $E_0, \ldots, E_{2m-1} \in \mathbb{H}$.
   (c) $\mathbf{a} \in \mathbb{Z}_q^n$.
   (d) $r, b, s, \tau \in \mathbb{Z}_q$.
   (e) $c_{B_m} = \mathsf{com}_{\mathsf{ck_{mix}}}(0; 0)$.
   (f) $E_m = C$.
   (g) $c_{A_0}\mathbf{c}_A^{\mathbf{x}} = \mathsf{com}_{\mathsf{ck_{mix}}}(\mathbf{a}; r)$.
   (h) $c_{B_0} \prod_{k=1}^{2m-1} c_{B_k}^{x^k} = \mathsf{com}_{\mathsf{ck_{mix}}}(b; s)$.
   (i) $E_0 \prod_{k=1}^{2m-1} E_k^{x^k} = \mathsf{Enc}(G^b; \mathsf{pk_{mix}}; \tau) \prod_{i=1}^m \mathbf{C}_i^{x^{m-i}\mathbf{a}}$.

If and only if all verifications above are successful, the algorithm outputs 1. Otherwise, it outputs 0.

8.6 $\mathsf{verifyProductArg}(\mathsf{pk_{mix}}, \mathsf{ck_{mix}}, \mathbf{c}_A, b, \pi_{\mathsf{Prod}})$

This algorithm verifies an argument that a set of committed values have a particular product i.e. given a commitment $\mathbf{c}_A$ to $A = \{\mathbf{a}_{ij}\}_{i,j=1}^{n,m}$ and a value b it gives an argument of knowledge for $\prod_{i=1}^n \prod_{j=1}^m a_{ij} = b$.

1. Parses π_{Prod} as $(c_b, \pi_{\mathsf{HadPA}}, \pi_{\mathsf{SingleVPA}})$.
2. Verifies:
   (a) $c_b \in \mathbb{G}$.
   (b) Checks that $\mathsf{verifyHadamardProdArg}(\mathsf{pk_{mix}}, \mathsf{ck_{mix}}, \mathbf{c}_A, c_b, \pi_{\mathsf{HadPA}})$ outputs 1.
   (c) Checks that $\mathsf{verifySingleValueProdArg}(\mathsf{pk_{mix}}, \mathsf{ck_{mix}}, c_b, b, \pi_{\mathsf{SingleVPA}})$ outputs 1.

If and only if all verifications above are successful, the algorithm outputs 1. Otherwise, it outputs 0.

8.7 verifyHadamardProdArg($\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_A, c_b, \pi_{\mathsf{HadPA}}$)

This algorithm verifies an argument of knowledge of the openings $a_{11}, \ldots, a_{nm}$ and $b_1, \ldots, b_n$ to the commitments $\mathbf{c}_A$ and c_b s.t. $b_i = \prod_{j=1}^{m} a_{ij}$ for $i = 1, \ldots, n$.

1. Parses π_{HadPA} as $(\mathbf{c}_B, \pi_{\mathsf{ZeroArg}})$.
2. Parses $\mathbf{c}_B = (c_{B_1}, \ldots, c_{B_m})$.
3. Verifies that $c_{B_2}, \ldots, c_{B_{m-1}} \in \mathbb{G}$, $c_{B_1} = c_{A_1}$ and $c_{B_m} = c_b$.
4. Computes challenges:
   (a) $x \leftarrow \mathsf{H}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_B, c_b, \mathbf{c}_A, \mathtt{aux})$.
   (b) $y \leftarrow \mathsf{H}(1, \mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_B, c_b, \mathbf{c}_A, \mathtt{aux})$.
5. Computes $c_{D_i} = c_{B_i}^{x^i}$, $c_D = \prod_{i=1}^{m-1} c_{B_{i+1}^{x^i}}$ and $c_{-1} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(-\mathbf{1}; 0)$.
6. Defines the bilinear map $\star : \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_q$ by $(a_1, \ldots, a_n)^T \star (d_1, \ldots, d_n)^T = \sum_{j=1}^{n} a_j d_j y^j$
7. Sets $\mathbf{c}_f = (c_{A_2}, \ldots, c_{A_m}, c_{-1})$ and $\mathbf{c}_d = (c_{D_1}, \ldots, c_{D_{m-1}, c_D})$.
8. Verifies $\mathsf{verifyZeroArg}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_f, \mathbf{c}_d, \star, \pi_{\mathsf{ZeroArg}})$.

If and only if all verifications above are successful, the algorithm outputs 1. Otherwise, it outputs 0.

8.8 verifyZeroArg($\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_A, \mathbf{c}_B, \star, \pi_{\mathsf{ZeroArg}}$)

This algorithm verifies an argument of knowledge of the committed values $\mathbf{a}_1, \mathbf{b}_0, \ldots, \mathbf{a}_m, \mathbf{b}_{m-1}$ s.t. $0 = \sum_{i=1}^{m} \mathbf{a}_i \star \mathbf{b}_{i-1}$.

1. Parses π_{ZeroArg} as $(c_{A_0}, c_{B_m}, \mathbf{c}_D, \mathbf{a}, \mathbf{b}, r, s, t)$.
2. Computes challenge $x \leftarrow \mathsf{H}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_D, c_{B_m}, c_{A_0}, \mathbf{c}_B, \mathbf{c}_A, \mathtt{aux})$.
3. Checks that:
   (a) $c_{A_0}, c_{B_m} \in \mathbb{G}$.
   (b) $\mathbf{c}_D \in \mathbb{G}^{2m+1}$.
   (c) $c_{D_{m+1}} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(0; 0)$.
   (d) $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_q^n$.
   (e) $r, s, t \in \mathbb{Z}_q$.
   (f) $\prod_{i=0}^{m} c_{A_i}^{x^i} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}; r)$.
   (g) $\prod_{j=0}^{m} c_{B_j}^{x^{m-j}} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{b}; s)$.
   (h) $\prod_{k=0}^{2m} c_{D_k}^{x^k} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a} \star \mathbf{b}; t)$.

If and only if all verifications above are successful, the algorithm outputs 1. Otherwise, it outputs 0.

8.9 verifySingleValueProdArg($\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, c_a, b, \pi_{\mathsf{SingleVPA}}$)

This algorithm verifies an argument of knowledge of the opening $a_1, \ldots, a_n, r$ s.t. $c_a = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}; r)$ and $b = \prod_{i=1}^{n} a_i$.

1. Parses $\pi_{\mathsf{SingleVPA}}$ as $(c_d, c_\sigma, c_\Delta, \tilde{a}_1, \tilde{b}_1, \ldots, \tilde{a}_n, \tilde{b}_n, \tilde{r}, \tilde{s})$.
2. Computes challenge $x \leftarrow \mathsf{H}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, c_\Delta, c_\sigma, c_d, b, c_a, \mathsf{aux})$.
3. Check that:
   (a) $c_d, c_\sigma, c_\Delta \in \mathbb{G}$.
   (b) $\tilde{a}_1, \tilde{b}_1, \ldots, \tilde{a}_n, \tilde{b}_n, \tilde{r}, \tilde{s} \in \mathbb{Z}_q$.
   (c) $c_a^x c_d = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\tilde{a}_1, \ldots, \tilde{a}_n; \tilde{r})$.
   (d) $c_\Delta^x c_\sigma = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(x\tilde{b}_2 - \tilde{b}_1\tilde{a}_2, \ldots, x\tilde{b}_n - \tilde{b}_{n-1}\tilde{a}_n; \tilde{s})$.
   (e) $\tilde{b}_1 = \tilde{a}_1$.
   (f) $\tilde{b}_n = xb$.

If and only if all verifications above are successful, the algorithm outputs 1. Otherwise, it outputs 0.

9 Hard problems

9.1 The Decisional Diffie-Hellman problem (DDH)

Definition 13 (DDH) *The Decisional Diffie-Hellman problem in a cyclic group $\mathbb{G}$ of order q with a generator g, informally, requires the indistinguishability of g_1^s from random element, given (g^s, g_1) for random $s \in \mathbb{Z}_q^*$ and $g_1 \in \mathbb{G}$. Figure 13 describes the* DDH *game formally for L different instances.*

The group $\mathbb{G}$ is DDH*-hard if it holds:*

$$Adv_{\mathcal{A}}^{\mathbb{G},\mathsf{DDH}} = |Pr\left[1 \leftarrow \mathsf{DDH}_{\mathcal{A}}^{\mathbb{G}}(\lambda)\right] - \frac{1}{2}| \approx 0.$$

DDH OVER QUADRATIC RESIDUES.. The group of quadratic residues $\mathbb{Q}_p < \mathbb{Z}_p^*$ of prime order q, for modulus $p = 2q + 1$ a safe prime of length λ is believed to be DDH-hard. In this case, in step 1 of Figure 13 the group parameters (sampled at random) are set to $\mathtt{gparams} = (p, q, g)$.

9.2 Subgroup Generated by Small Primes (SGSP)

This problem was first introduced in [25]. It is similar to the DDH problem but weaker.

Definition 14 (SGSP) *The SGSP problem in the group of quadratic residues $\mathbb{Q}_p$ informally, requires the indistinguishability of ℓ_1^s from a random element, given (g^s, ℓ_1) for random $s \in \mathbb{Z}_q^*$, and $\ell \in \mathbb{Q}_p$ the*

$\underline{\mathsf{DDH}^{\mathbb{G}}_{\mathcal{A}}(\lambda)}$:

1. $\mathtt{gparams} \leftarrow \mathsf{GParamsGen}(\lambda)$ //Sample at random
2. $b \leftarrow \{0,1\}$
3. $s \leftarrow \mathbb{Z}_q^*$
4. $s_i \leftarrow s \; \forall \; 0 \leq i \leq L$
5. $g_1, \ldots, g_L \leftarrow \mathbb{G}$ // Sampled at random
6. if $b = 1$ then
   6.1 $s_i \leftarrow \mathbb{Z}_q^* \forall 0 \leq i \leq L$
7. $b' \leftarrow \mathcal{A}(\mathtt{gparams}, g^{s_0}, g_1^{s_1}, \ldots, g_L^{s_L})$
8. Output 1 if $b = b'$. Else output 0.

Figure 13: DDH game. GParamsGen samples at random the group parameters gparams, including the order q and a generator g of the cyclic group $\mathbb{G}$.

first prime that is a quadratic residue. Figure 14 describes the SGSP *game formally for the first L primes that are quadratic residues.*

The group $\mathbb{Q}_p$ is SGSP*-hard if it holds:*

$$Adv_{\mathcal{A}}^{\mathbb{Q}_p,\mathsf{SGSP}} = |Pr\left[1 \leftarrow \mathsf{SGSP}_{\mathcal{A}}^{\mathbb{Q}_p}(\lambda)\right] - \frac{1}{2}| \approx 0.$$

$\underline{\mathsf{SGSP}^{\mathbb{Q}_p}_{\mathcal{A}}(\lambda)}$:

1. $\mathtt{gparams} \leftarrow \mathsf{GParamsGen}(\lambda)$ // Sampled at random
2. $\ell_1, \ldots, \ell_L \leftarrow \mathbb{Q}_p$ // Find first L smallest primes that are quadratic residues
2. $b \leftarrow \{0,1\}$
3. $s \leftarrow \mathbb{Z}_q^*$
4. $s_i \leftarrow s \; \forall 0 \leq i \leq L$
5. if $b = 1$ then
   5.1 $s_i \leftarrow \mathbb{Z}_q^* \forall \; 0 \leq i \leq L$
6. $b' \leftarrow \mathcal{A}(\mathtt{gparams}, g^{s_0}, \ell_1^{s_1}, \ldots \ell_L^{s_L})$
7. Output 1 if $b = b'$. Else output 0.

Figure 14: SGSP game. GParamsGen samples at random the group parameters $\mathtt{gparams} = (p, q, g)$, including the order q and a generator g of $\mathbb{Q}_p$.

The following problem combines DDH and SGSP problems.

Definition 15 (ESGSP) *The extended SGSP problem in the group of quadratic residues $\mathbb{Q}_p$ informally, requires the indistinguishability of (g_1^s, ℓ_1^s) from random elements, given (g^s, g_1, ℓ_1) for random $s \in \mathbb{Z}_q^*$, random $g_1 \in \mathbb{Q}_p$ and $\ell \in \mathbb{Q}_p$ the first prime that is a quadratic residue. Figure 15 describes the* ESGSP *game formally for L_1 random group elements and the first L_2 primes that are quadratic residues.*

The group $\mathbb{Q}_p$ *is* ESGSP*-hard if it holds:*

$$Adv_{\mathcal{A}}^{\mathbb{Q}_p,\mathsf{ESGSP}} = |Pr\left[1 \leftarrow \mathsf{SGSP}_{\mathcal{A}}^{\mathbb{Q}_p}(\lambda)\right] - \frac{1}{2}| \approx 0.$$

$\underline{\mathsf{ESGSP}_{\mathcal{A}}^{\mathbb{Q}_p}(\lambda)}$:

1. $\mathtt{gparams} \leftarrow \mathsf{GParamsGen}(\lambda)$
2. $g_1, \ldots, g_{L_1} \leftarrow \mathbb{Q}_p$ // Sampled at random
3. $\ell_1, \ldots, \ell_{L_2} \leftarrow \mathbb{Q}_p$ // Find first L_2 smallest primes that are quadratic residues
2. $b \leftarrow \{0, 1\}$
3. $s \leftarrow \mathbb{Z}_q^*$
4. $s_i \leftarrow s \; \forall 0 \leq i \leq L_1 + L_2$
5. if $b = 1$ then
   5.1 $s_i \leftarrow \mathbb{Z}_q^* \forall \; 0 \leq i \leq L_1 + L_2$
6. $b' \leftarrow \mathcal{A}(\mathtt{gparams}, g^{s_0}, g_1^{s_1}, \ldots g_{L_1}^{s_{L_1}}, \ell_1^{s_{L_1+1}}, \ldots \ell_{L_2}^{s_{L_1+L_2}})$
7. Output 1 if $b = b'$. Else output 0.

Figure 15: ESGSP game. GParamsGen samples at random the group parameters $\mathtt{gparams} = (p, q, g)$, including the order q and a generator g of $\mathbb{Q}_p$.

HARDNESS OF ESGSP PROBLEM.. It is not difficult to see that if $\mathbb{Q}_p$ is DDH-hard and SGSP-hard, then it is also ESGSP-hard.

Part III

sVote voting system

10 General aspects

sVote is a e-voting system that allows to vote for up to ψ options out of n voting options.

The system consists of three phases: the configuration phase, the voting phase and the tally phase. Each phase has several protocols (interactive algorithms) and procedures (local algorithms). Phases and flow diagrams are covered in Section 11. Descriptions of the algorithms are covered in Section 12.

The tally phase outputs plaintexts of the confirmed votes in random order while maintaining vote privacy, individual verifiability (cast-as-intended and recorded-as-cast), and universal verifiability (counted-as-recorded). Section 10.2 discusses the techniques for achieving these security objectives.

10.1 Public system parameters

Some of the system parameters are generated before the configuration phase begins. These parameters, jointly denoted as `crs`, form the common reference string that all parties in the system know. Thus, implicitly all algorithms take `crs` as input. Public parameters are generated in a verifiable way.

10.1.1 ElGamal parameters

To encrypt the voting options in the Voting Client , sVote uses ElGamal over the multiplicative group of quadratic residues $\mathbb{Q}_p$. The group parameters $\texttt{gparams} = (p, q, g)$, namely the modulus p, the order q and the generator g are generated in advance. See Section 3.2 for more details.

10.1.2 Pedersen commitment key

A Pedersen commitment is instantiated over $\mathbb{Q}_p$, the same group used for ElGamal encryptions. The pair of commitment keys is generated in advance.

10.1.3 Voting options

Voting options are encoded as elements of $\mathbb{Q}_p$. They are chosen as the first n primes that are quadratic residues modulo p. We denote them with vector $\mathbf{v} = (v_1, \ldots, v_n)$.

We use the operations of multiplication and factorization for encoding/decoding the selected voting options: a voter selects ψ voting options v'_i out of n possibilites, and the Voting Client encodes them as the group element $\nu = \prod_{i=1}^{\psi} v'_i \in \mathbb{Q}_p$. During tally, the votes are eventually decrypted, and the voter selections v'_i recovered by factorizing the plaintext ν in basis $\mathbf{v}$. Therefore, it has to be ensured that the product of any subset of ψ primes is smaller than p.

10.1.4 Voter pseudonyms

A list $\mathcal{ID}$ of N pseudonyms (identifiers) of eligible voters is available prior to the configuration phase. This list is provided by an external authority.

10.1.5 Return codes spaces

To derive human-readable return codes and keys, we use different value spaces.

- Start voting keys are derived from $\mathcal{C}_{svk}$. These keys are used to send a vote.
- Ballot casting keys are derived from $\mathcal{C}_{bck}$. These keys are used to confirm a vote .
- Short choice return codes are derived from $\mathcal{C}_{cc}$. These codes are used to verify cast-as-intended.
- Short vote cast return codes are derived from $\mathcal{C}_{vcc}$. These codes is used to verify recorded-as-cast.

10.2 Achieving verifiability and privacy

10.2.1 Individual verifiability

sVote uses a two-round protocol between the voter, the Voting Client and the server to provide individual verifiability.

Before the voting phase is open, an eligible voter with identifier $\mathtt{id}$ receives a paper-based voting card $\mathtt{vcd}_{\mathtt{id}}$[7] with a one-to-one correspondence between the voting options $\mathbf{v}$ and a set of *Choice Return Codes* $\mathbf{cc}_{\mathtt{id}} = (\mathtt{CC}_{1,\mathtt{id}}, \ldots, \mathtt{CC}_{n,\mathtt{id}})$.

During the voting phase, the voter selects ψ voting options, then her voting client prepares the ballot and sends it to the server. The server jointly with CCRs derives a set of choice return codes $\mathbf{cc}_{\mathtt{id}} = (\mathtt{CC}_{1,\mathtt{id}}, \ldots, \mathtt{CC}_{\psi,\mathtt{id}})$ that correspond to the ψ selected voting options and sends them back to Voting Client. The Voting Client shows the received Choice Return Codes $\mathbf{cc}_{\mathtt{id}}$ to the voter so that she can cross-check them against the Choice Return Codes $\mathbf{cc}_{\mathtt{id}}$ printed on her voting card. If the Choice Return Codes $\mathbf{cc}_{\mathtt{id}}$ match, the voter confirms the vote by entering into the Voting Client a Ballot Casting Key $\mathtt{BCK}_{\mathtt{id}}$ that is also printed on her voting card. The Voting Client derives the confirmation message $\mathtt{CM}^{\mathtt{id}}$ from the Ballot Casting Key $\mathtt{BCK}_{\mathtt{id}}$ and sends the confirmation message $\mathtt{CM}^{\mathtt{id}}$ to the voting server. Subsequently, the voting server retrieves, in collaboration with the CCR, the Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$ and sends it back to the voting client to allow the voter to cross-check the received Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$ against the Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$ printed on the voting card. If the voter receives the correct Choice Return Codes $\mathbf{cc}_{\mathtt{id}}$ and Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$, the system has correctly registered the vote. Otherwise, the voter complains to the authorities to invalidate the vote.

[7] We do not cover the provision of voting cards to the voters; it is assumed they are delivered by a secure channel, e.g. postal channel.

GENERATION OF CHOICE RETURN CODES AND VOTE CAST RETURN CODE. During the configuration phase, the Print Office generates the short Choice Return Code $\texttt{CC}_{i,\texttt{id}}$ and the short Vote Cast Return Code $\texttt{VCC}_{\texttt{id}}$ for each voter with identifier $\texttt{id}$ and each voting option v_i. Furthermore, the Print Office collaborates with the Choice Return Codes control components (CCR) to generate the long Choice Return Code $\texttt{lCC}_{i,\texttt{id}}$ and the long Vote Cast Return Code $\texttt{lVCC}_{\texttt{id}}$. The Print Office symmetrically encrypts the short Choice Return Codes $\texttt{CC}_{i,\texttt{id}}$ and the Vote Cast Return Code $\texttt{VCC}_{\texttt{id}}$. Then, it maps the hash of the long Choice Return Code $\texttt{lCC}_{i,\texttt{id}}$ to the encrypted short Choice Return Codes $\texttt{CC}_{i,\texttt{id}}$ and the hash of the long Vote Cast Return Code $\texttt{lVCC}_{\texttt{id}}$ to the encrypted short Vote Cast Return Code $\texttt{VCC}_{\texttt{id}}$ and stores the result in the *codes mapping table* CMtable[8].

The Print Office sends the mapping table CMtable to the voting server.

RETRIEVAL OF CHOICE RETURN CODES AND VOTE CAST RETURN CODE. During the voting phase, the voting client sends $(\texttt{E1}, \texttt{E2}, \boldsymbol{\pi}_{\mathsf{ballot}})$ to the voting server, where $\texttt{E1}$ is the ciphertext used for the tally phase, $\texttt{E2} = (c_{2,0}, c_{2,1}, \ldots, c_{2,\psi})$ is a multi-recipient ElGamal ciphertext containing encryptions of the Partial Choice Return Codes $\texttt{pCC}_{i,\texttt{id}}$, and $\boldsymbol{\pi}_{\mathsf{ballot}}$ is a set of three NIZK proofs. The voting server and the CCR collaborate to retrieve the long Choice Return Code $\texttt{lCC}_{i,\texttt{id}}$ and to retrieve the corresponding encrypted short Choice Return Code $\texttt{CC}_{i,\texttt{id}}$ from the mapping table CMtable. Then, the voting server decrypts the short Choice Return code $\texttt{CC}_{i,\texttt{id}}$ and sends it to the voting client. A similar mechanism is applied in the second round of the protocol to retrieve the short Vote Cast Return Code $\texttt{VCC}_{\texttt{id}}$; first by collaboratively generating the long Vote Cast Return code $\texttt{lVCC}_{\texttt{id}}$ and then by retrieving the encrypted short Vote Cast Return Code $\texttt{VCC}_{\texttt{id}}$ from the mapping Table CMtable.

Three NIZK proofs $\boldsymbol{\pi}_{\mathsf{ballot}}$ ensure that the encoded voting options $\texttt{E1}$ correspond to the encrypted Partial Choice Return Codes $\texttt{E2}$.

10.2.2 Universal verifiability and voter privacy

The protocol uses verifiable mixnets [2] to ensure voter privacy. These mixnets are enhanced with zero-knowledge proofs of correct shuffling and decryption which allow to check the correctness of the output of the tally whilst maintaining the privacy of each voter and which exclude the possibility of obtaining early provisional results.

11 Phases

[8]The Print Office permutes the codes mapping table CMtable before sending it to the voting server. This prevents the voting server from correlating the voting options to the position in the mapping table in order to break vote privacy

11.1 Configuration phase

In the configuration phase the control components and the Print Office generate the Election key pair and the voting cards of the voters.

It consists of two interactive protocols SetupTally and SetupVoting. In addition, at the end of the phase the auditors execute a verification algorithm VerifyConfigPhase to ensure everything was generated properly. The public parameters `crs` are known to all system parties. Refer to section 12.1 for the complete list of algorithms executed during configuration.

Protocol Setup Tally. The Print Office and the Mixing control components generate the global Election key pair.

- Each control component CCM_j for $j \leq m' - 1$ generates a share of the Election key pair.
- The Print Office generates the last share and combines all the Election public key shares.

See Figure 16 for an overview of the interaction and Section 12.1.2 for the description of the algorithms.

Protocol Setup Voting. The Print Office and the Choice Return Codes control components generate the Global CCR encryption key pair and credential and verification material for each voter.

- Each CCR_j generates an encryption key pair, and Print Office combines the encryption keys of each CCR_j.
- The Print Office generates the Verification Card key pairs for each voter, and sends encryptions (under the Secure Data Manager encryption key) of the Partial Choice Return Codes and Confirmation Key (voting options and Ballot Casting Key raised to the Verification Card Private key of each voter, respectively) to the CCRs.
- The CCRs exponentiate the two sets of received ciphertexts to their Voter Choice Return Code Generation and Voter Vote Cast Return Code Generation private keys, respectively. These are the (encrypted) long Choice Return Code shares and long Vote Cast Return Code shares that each control component generates.
- The Print Office generates the short Choice Return Codes and short Vote Cast Return Code. Then it combines and decrypts the ciphertexts returned by the CCRs and obtains the two sets of long return codes. These codes are used to derive the symmetric keys to encrypt the short return codes in the Codes mapping table.

See Figure 17 for an overview and Section 12.1.1 for the description of the algorithms.

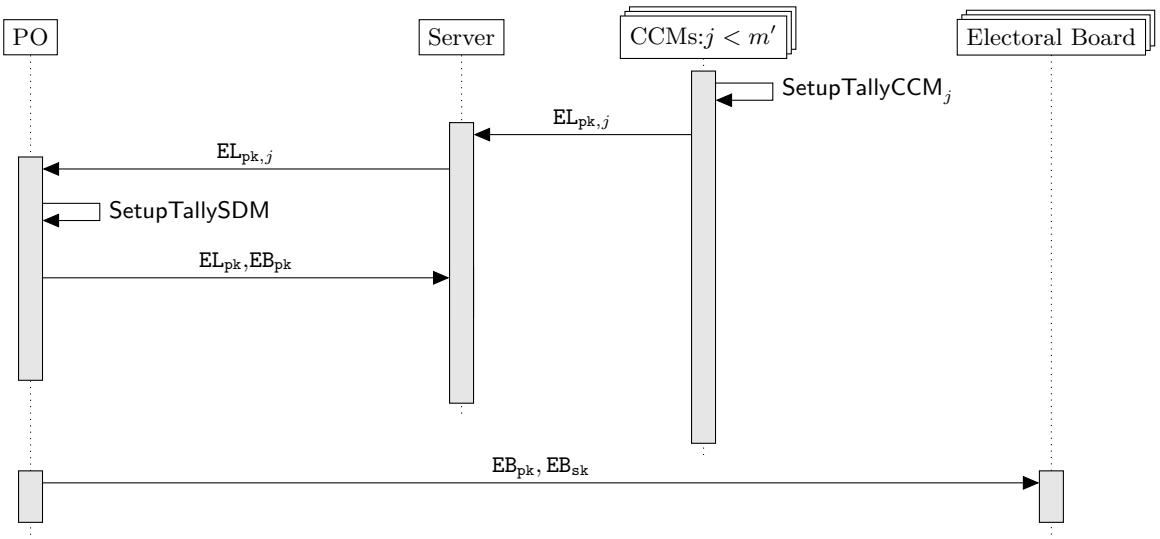

Figure 16: Protocol SetupTally, executed between the Print Office and the Mixing control components. Each control component, except the last one, generates its own share of the Election key pair and send it back to the Print Office, who generates the last Election key pair (the Electoral Board key pair) share and combines all shares.

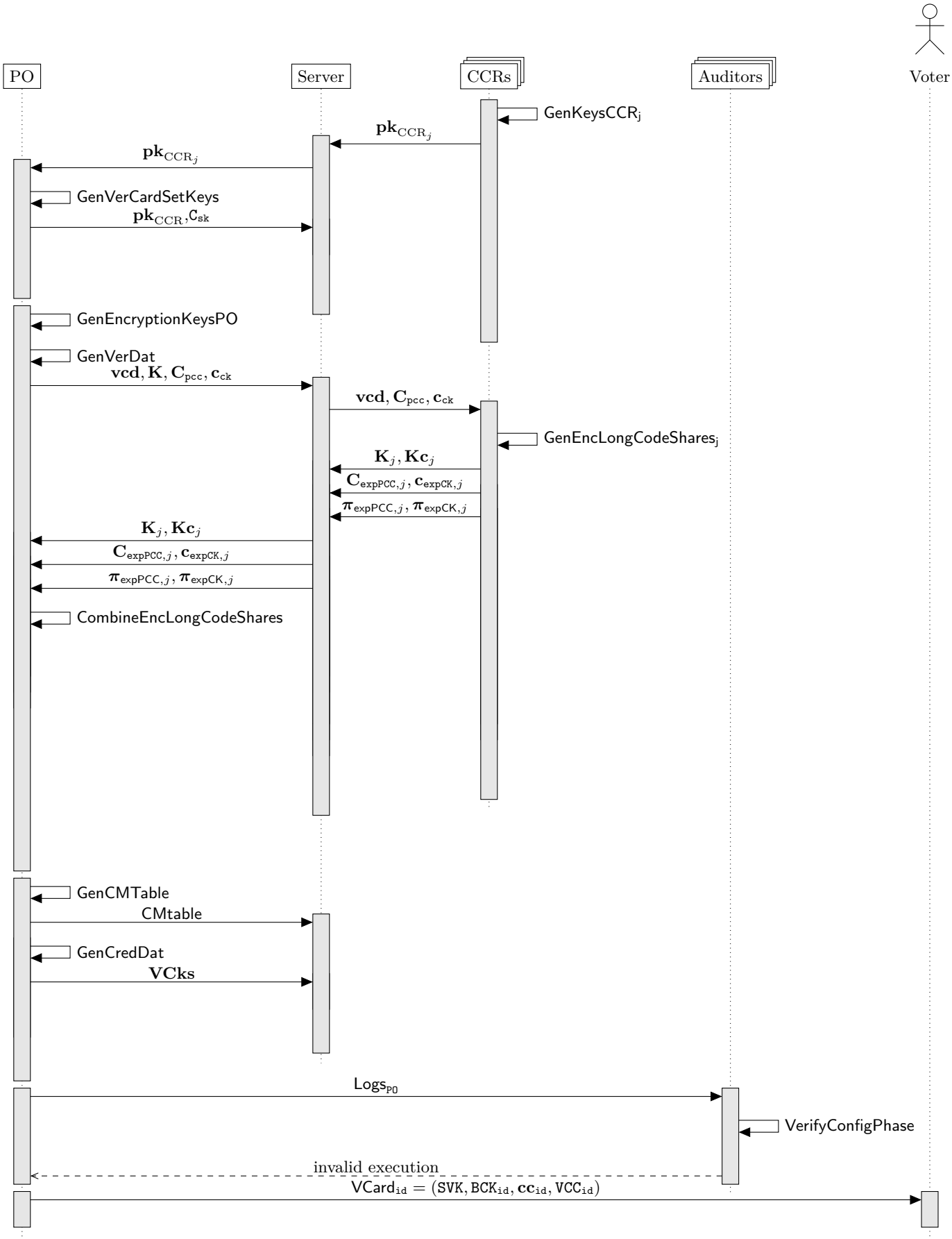

Figure 17: Protocol SetupVoting executed between the Print Office and the Choice Return Code control components CCRs. Upon termination, auditors verify execution and the voters receive the Voting Card with the Start Voting Key, the Ballot Casting Key, the voting options and the short Return Codes.

11.2 Voting phase

In the voting phase a Voter sends and confirms her vote with the help of a Voting Client and her voting card.

It consists in two interactive protocols SendVote and ConfirmVote. In addition, at the end of the phase the auditors execute a verification procedure VerifyVotingPhase to ensure the ballot box bb is consistent. Refer to section 12.2 for the complete list of algorithms executed during the voting phase.

Protocol Send Vote. The Voter starts the protocol introducing the Start Voting Key in the Voting Client , who creates the ballot and send it to the Server. Then, the Server and the CCRs interact to retrieve the short Choice Return Codes from the Codes mapping table using information encrypted in the ballot. The Server sends the return codes to the Voting Client and these are displayed to the Voter.

- The Voter enters the Start Voting Key $\texttt{SVK}_{\texttt{id}}$, and her selected voting options $\mathbf{v}_{\texttt{id}} = (v_{j_1}, \ldots, v_{j_\psi})$ printed on her voting card.
- The Voting Client creates the ballot and sends it to the Server.
- The Server and the CCRs validate the ballot.
- The Server and the CCRs interact to decrypt the Partial Choice Codes embedded in the ballot.
- The Server and the CCRs interact to derive the long Choice Return Codes using the Partial Choice Codes. During the process the CCRs validates Server's requests (for a given Voter's identifier, decryption has been done and no previous request has been answered).
- The Server extracts the short Choice Return Codes from the mapping table and send them back to the Voting Client .
- The Voter verifies the short Choice Return Codes.

For an overview of the interaction see Figure 18 and for a description of the algorithms see Section 12.2.1.

Protocol Confirm Vote. The Voter starts the protocol introducing the Ballot Casting Key in the Voting Client , who creates the Confirmation Key and sends it to the Server. The Server and the CCRs interact to retrieve the short Vote Cast Return Code from the mapping table.

- The Voter enters the Ballot Casting Key in the Voting Client , who sends the Confirmation Key to the Server.
- The CCRs validate Server's request and exponentiate the Confirmation Key to obtain the long Vote Cast Return Code shares.
- The Server combines the shares of the CCRs and extracts the short Vote Cast Return Code from the Codes mapping table and sends them back to the Voting Client.

- The Voter verifies the Vote Cast Return Code. If the verification is successful, the voter has concluded the vote cast successfully. Otherwise, she notifies the authorities and uses another channel to cast her vote.

For an overview of the interaction see Figure 19 and for a description of the algorithms see Section 12.2.2.

11.3 Tally phase

In the tally phase the Server and the Mixing control components compute the result of the election.

It consists in two protocols, MixOnline and MixOffline. In addition, two audits are in place to ensure no party deviate from the protocol:

1. After completion of the online mix protocol, the auditors run algorithm VerifyOnlineTally (section 12.3.3.1) to ensure the protocol has been executed correctly.

2. After completion of the offline mix, the auditors run algorithm VerifyOfflineTally (section 12.3.3.2) to ensure the output of the last Mixing control component $\text{CCM}_{m'}$ is correct. If this last audit is successful, the result is deemed trustworthy.

Online mix protocol. First, the server first cleans the ballot box removing all information related to the voters and keeping only the ciphertexts for mixing. Then, all but the last Mixing control components shuffle and partially decrypt the list of ciphertext received from the preceding control component (or from the server in case of the first control component). They also produce NIZK proofs for correct shuffling and correct partial decryption that are described in sections 8 and 7.3.5.

Offline mix protocol. The last Mixing control component $\text{CCM}_{m'}$ shuffles and decrypt the permuted ballot box received from $\text{CCM}_{m'-1}$. It also produces NIZK proofs for correct shuffling and correct decryption. Finally, it decodes the decrypted votes and outputs the result of the election.

Finally, the auditors *may* also run algorithm Auditors.VerifyElection to verify the entire election. Refer to Section 12.3 for the complete list of algorithms of the tally phase. Figure 20.

12 Protocols and procedures

This section is devoted to list the algorithms executed at each phase of the sVote protocol.

Unless specified otherwise, the public system parameters `crs` are understood to be implicit inputs to those algorithms that use them. Recall that `crs` includes a description $\texttt{gparams} = (p, q, g)$ of $\mathbb{Q}_p$, a list of voter pseudonyms $\mathcal{ID}$, the number of voting options and maximum of choices (n, ψ), and a set of voting options $\mathbf{v} = \{v_1, \ldots, v_n\}$. See Section 10.1 for more details.

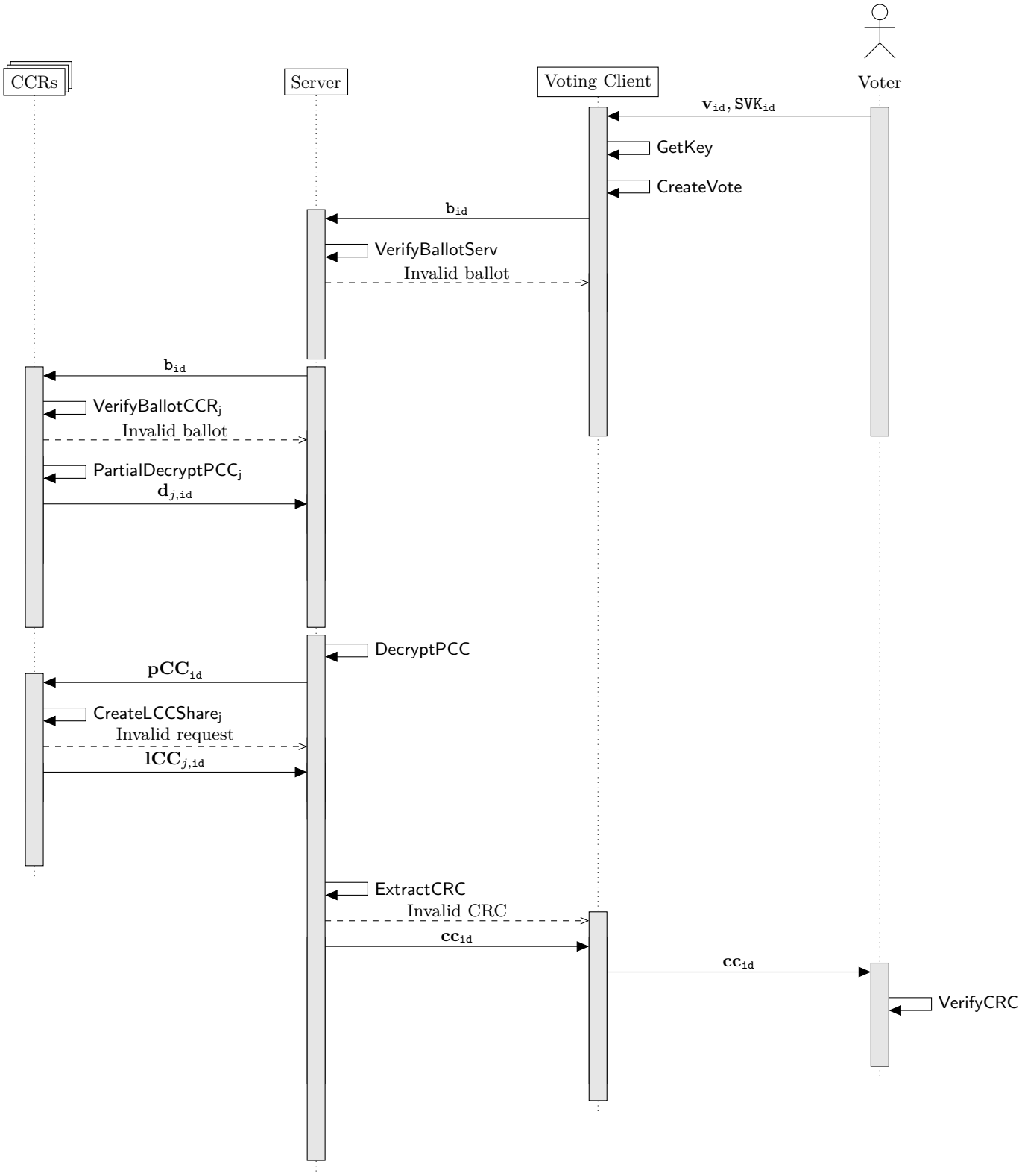

Figure 18: Protocol SendVote executed between the Voting Server and the Choice Return Code control components CCRs. Upon termination, voters receive short Choice Return Codes and perform verification.

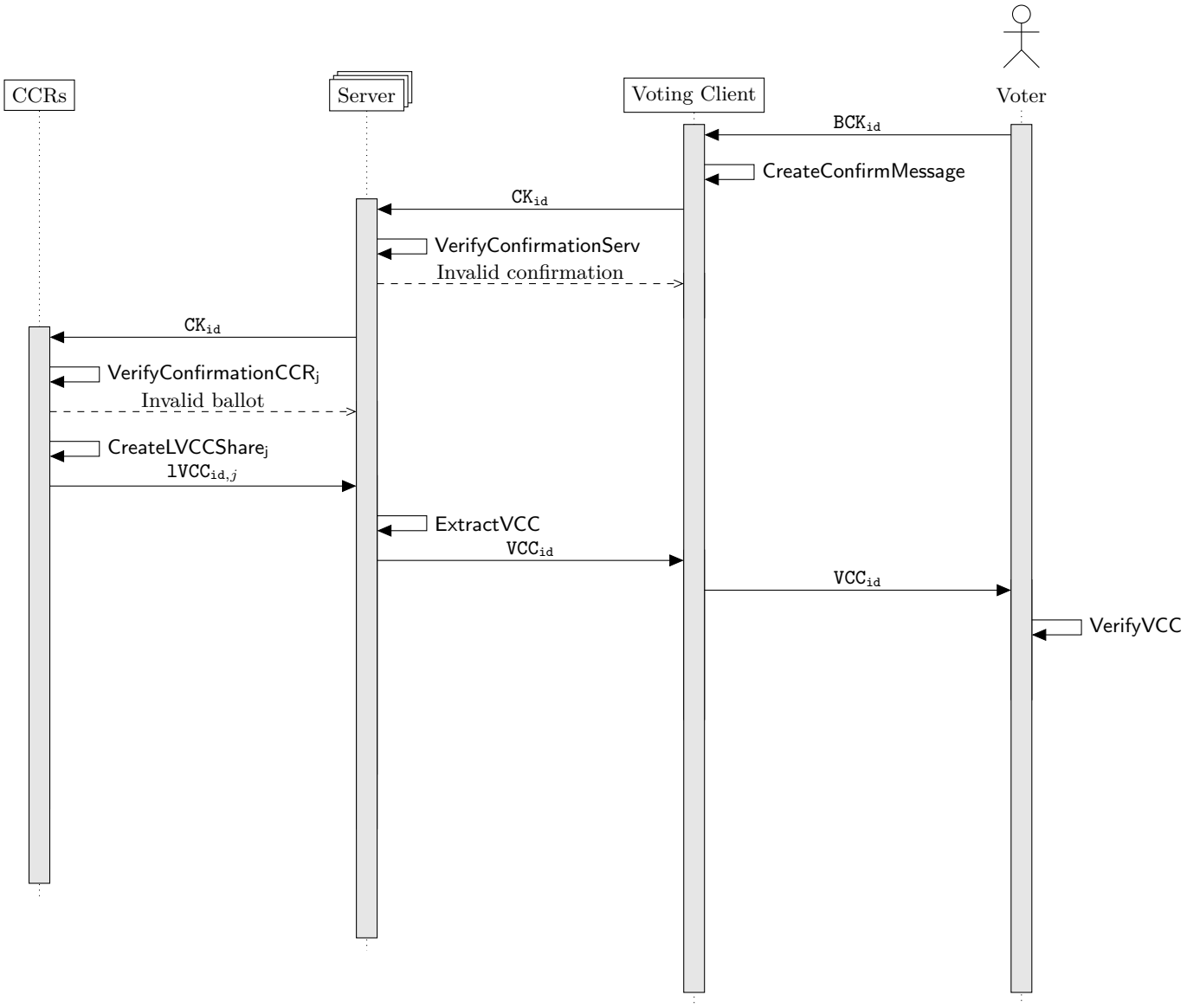

Figure 19: Protocol ConfirmVote executed between the Voting Server and the Choice Return Code control components CCRs. Upon termination, voters receive the Vote Cast Return Code and perform the verification.

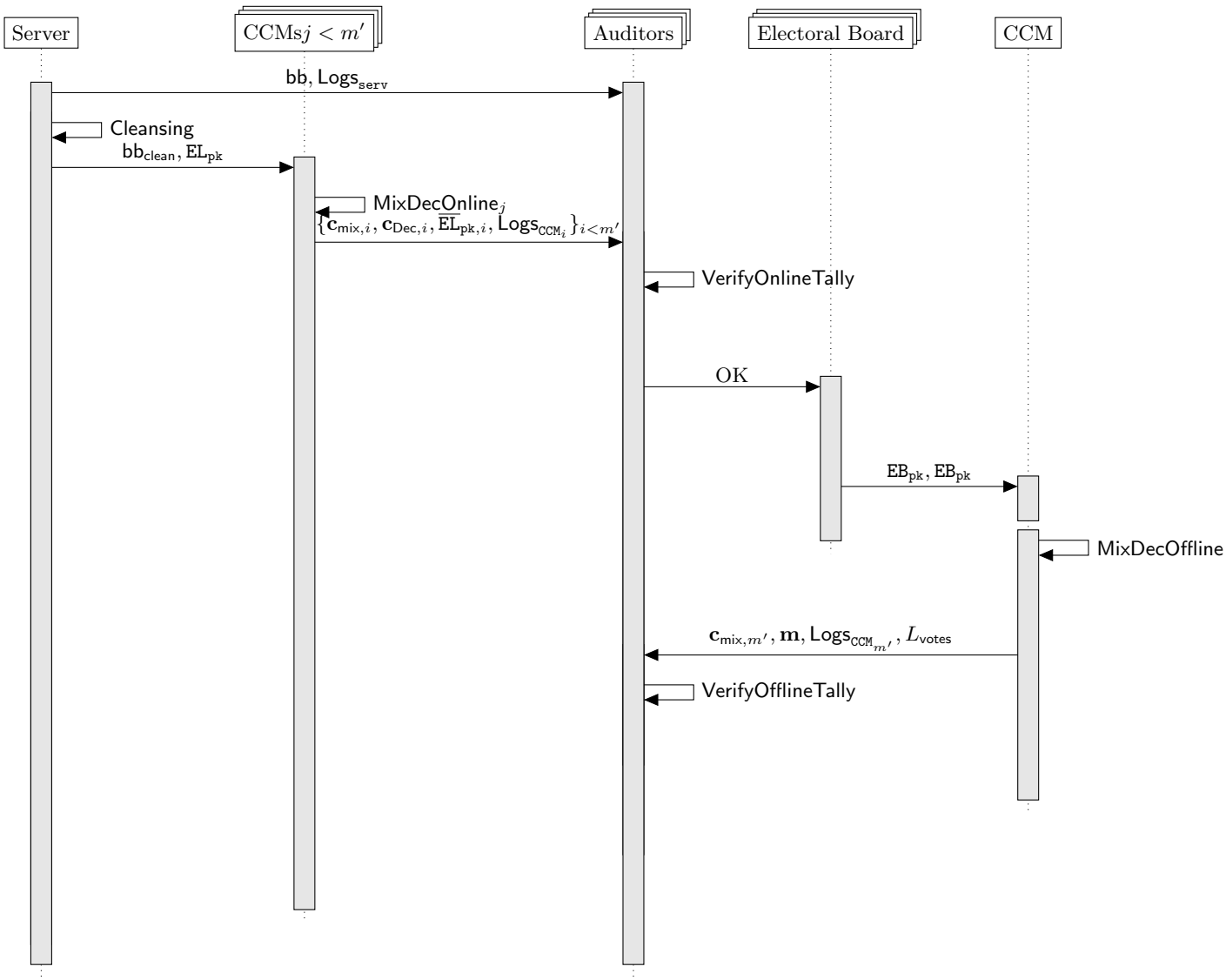

Figure 20: Protocol FirstMixes executed between the Voting Server and the Mixing control components CCMs. Protocol is divided in two phases: online mixing (all but last CCMs) and offline mixing (last CCM). Upon online mixing termination, auditors perform audit. If and only if the verification is successful, last CCM receives its key pair from Electoral Board members.

12.1 Configuration phase protocols

The execution flow of the algorithms in the configuration phase is depicted in Figure 17 (setup for voting phase) and Figure 16 (setup for tally phase).

12.1.1 Protocol SetupVoting

It is an interactive protocol run between the Print Office and the Choice Return Codes control components CCRs. It consist of the following algorithms.

12.1.1.1 GenEncryptionKeysPO($\mathtt{crs}, \mathsf{Logs}_{\mathtt{PO}}$)

This algorithm is executed by the Print Office. The algorithm takes as input the public parameters $\mathtt{crs}$ and logs $\mathsf{Logs}_{\mathtt{PO}}$. Then it generates the Secure Data Manager encryption key pair $(\mathtt{sk}_{\mathtt{SDM}}, \mathtt{pk}_{\mathtt{SDM}}) \leftarrow \mathsf{KeyGen}(\mathtt{gparams})$. This key pair is used to communicate with the control components. Additionally it logs the public key $\mathsf{Logs}_{\mathtt{PO}} \leftarrow \mathsf{Logs}_{\mathtt{PO}} \cup \{\mathtt{pk}_{\mathtt{SDM}}\}$.

12.1.1.2 GenKeysCCR$_j$($\mathtt{crs}, \mathsf{Logs}_{\mathtt{CCR}_j}$)

This algorithm is executed by the Choice Return Code control component CCR_j. It receives as input the public parameters $\mathtt{crs}$, and the logs $\mathsf{Logs}_{\mathtt{CCR}_j}$. Then it does the following:

- Generate the CCR_j Choice Return Codes encryption key pairs $(\mathtt{sk}_{\mathrm{CCR}_j,i}, \mathtt{pk}_{\mathrm{CCR}_j,i}) \leftarrow \mathsf{KeyGen}(\mathtt{gparams})$ for $i \in (1, \ldots, \psi)$. Set $\mathbf{sk}_{\mathrm{CCR}_j} = (\mathtt{sk}_{\mathrm{CCR}_j,1}, \ldots, \mathtt{sk}_{\mathrm{CCR}_j,\psi})$ and $\mathbf{pk}_{\mathrm{CCR}_j} = (\mathtt{pk}_{\mathrm{CCR}_j,1}, \ldots, \mathtt{pk}_{\mathrm{CCR}_j,\psi})$.
- Compute at random the CCR_j Choice Return Codes Generation private key $\mathtt{k'}_j \in \mathbb{Z}_q^*$.
- Update the logs $\mathsf{Logs}_{\mathtt{CCR}_j} \leftarrow \mathsf{Logs}_{\mathtt{CCR}_j} \cup \{\mathbf{pk}_{\mathrm{CCR}_j}\}$.

The algorithm outputs $((\mathbf{sk}_{\mathrm{CCR}_j}, \mathbf{pk}_{\mathrm{CCR}_j}), \mathtt{k'}_j)$ and $\mathsf{Logs}_{\mathtt{CCR}_j}$.

12.1.1.3 GenVerCardSetKeys($(\mathbf{pk}_{\mathrm{CCR}_1}, \ldots, \mathbf{pk}_{\mathrm{CCR}_m}), \mathsf{Logs}_{\mathtt{PO}}, \mathtt{crs}$)

This algorithm is executed by the Print Office to create the global public encryption key of the control components and the Codes Secret key.[9]

The algorithm takes as input the public keys $\mathbf{pk}_{\mathrm{CCR}_j} = (\mathtt{pk}_{\mathrm{CCR}_j,1}, \ldots, \mathtt{pk}_{\mathrm{CCR}_j,\psi})$ of each CCR_j. Then it does the following: :

- Compute the global Choice Return Codes Encryption public keys $\mathbf{pk}_{\mathrm{CCR}} = (\mathtt{pk}_{\mathrm{CCR},1}, \ldots, \mathtt{pk}_{\mathrm{CCR},\psi})$ as follows:

$$\mathtt{pk}_{\mathrm{CCR},i} = \prod_{j=1}^{m} \mathtt{pk}_{\mathrm{CCR}_j,i} \quad \forall i \in (1, \ldots, \psi).$$

[9] The Codes Secret key is a salt value used for the codes mapping table (see Algorithm 12.1.1.7).

- Generate a random Codes Secret key $\mathtt{C}_{\mathtt{sk}} \xleftarrow{\$} \{0,1\}^s$.
- Updates logs $\mathsf{Logs}_{\mathtt{PO}} \leftarrow \mathsf{Logs}_{\mathtt{PO}} \cup (\mathbf{pk}_{\mathrm{CCR}_1}, \ldots, \mathbf{pk}_{\mathrm{CCR}_m}, \mathtt{C}_{\mathtt{sk}})$.

The algorithm outputs $(\mathbf{pk}_{\mathrm{CCR}}, \mathtt{C}_{\mathtt{sk}}, \mathsf{Logs}_{\mathtt{PO}})$.

12.1.1.4 GenVerDat($\mathtt{pk}_{\mathtt{SDM}}, \mathsf{Logs}_{\mathtt{PO}}, \mathtt{crs}$)

It is an algorithm run by the Print Office. It generates verification data for each voter, and prepares ciphertexts to send to the Choice Return Code control components.

The algorithm receives as input the Secure data Manager public encryption key $\mathtt{pk}_{\mathtt{SDM}}$, logs $\mathsf{Logs}_{\mathtt{PO}}$ and the public parameters $\mathtt{crs}$. Then, it does the following for voters $\mathtt{id} \in \mathcal{ID}$:

1. Parse the voting options as $(v_1, \ldots, v_n) \leftarrow \mathbf{v}$.
2. Generate a random Voting Card ID $\mathtt{vcd}_{\mathtt{id}}$ for voter $\mathtt{id}$.[10] Set vector $\mathbf{vcd} = (\mathtt{vcd}_{\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$.
3. Generate a Verification Card key pair: $(\mathtt{k}_{\mathtt{id}}, \mathtt{K}_{\mathtt{id}} = g^{\mathtt{k}_{\mathtt{id}}}) \leftarrow \mathsf{KeyGen}(\mathtt{gparams})$ for voter $\mathtt{id}$. Set vectors $\mathbf{k} = (\mathtt{k}_{\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$, and $\mathbf{K} = (\mathtt{K}_{\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$.
4. Generate a random Ballot Casting Key $\mathtt{BCK}_{\mathtt{id}} \xleftarrow{\$} C_{bck}$ for voter $\mathtt{id}$. Set vector $\mathbf{bck} = (\mathtt{BCK}_{\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$.
5. Generate the following *encrypted verification material*:
   - Compute hashes of the Partial Choice Return Codes
   $$\mathtt{hpcc}_{\mathtt{id},i} = \left(\mathsf{H}(v_i^{\mathtt{k}_{\mathtt{id}}})\right)^2 \in \mathbb{Q}_p \quad \forall i \in (1, \ldots, n).$$
   - Encrypt under the Secure Data Manager public key $\mathtt{pk}_{\mathtt{SDM}}$ the above elements as:
   $$\mathbf{c}_{\mathtt{pcc},i,\mathtt{id}} \leftarrow \mathsf{Enc}(\mathtt{hpcc}_{\mathtt{id},i}, \mathtt{pk}_{\mathtt{SDM}}), \quad \forall i \in (1, \ldots, n).$$
   Set vector $\mathbf{c}_{\mathtt{pcc},\mathtt{id}} = (\mathbf{c}_{\mathtt{pcc},1,\mathtt{id}}, \ldots, \mathbf{c}_{\mathtt{pcc},n,\mathtt{id}})$, and let $\mathbf{C}_{\mathtt{pcc}}$, the $N \times n$-matrix with each row set to $\mathbf{c}_{\mathtt{pcc},\mathtt{id}}$.
   - Compute the Confirmation Key $\mathtt{CK}_{\mathtt{id}} = (\mathtt{BCK}_{\mathtt{id}})^{2 \cdot \mathtt{k}_{\mathtt{id}}}$.
   - Compute a hash of the Confirmation Key $\mathtt{hck}_{\mathtt{id}} = \left(\mathsf{H}(\mathtt{CK}_{\mathtt{id}})\right)^2 \in \mathbb{Q}_p$ and encrypt it under the Secure Data Manager public key $\mathtt{pk}_{\mathtt{SDM}}$:
   $$\mathbf{c}_{\mathtt{ck},\mathtt{id}} \leftarrow \mathsf{Enc}(\mathtt{hck}_{\mathtt{id}}, \mathtt{pk}_{\mathtt{SDM}}).$$
   Set vector $\mathbf{c}_{\mathtt{ck}} = (\mathbf{c}_{\mathtt{ck},\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$.
6. Update logs $\mathsf{Logs}_{\mathtt{PO}} \leftarrow \mathsf{Logs}_{\mathtt{PO}} \cup \{\mathbf{vcd}, \mathbf{K}, \mathbf{C}_{\mathtt{pcc}}, \mathbf{c}_{\mathtt{ck}}\}$.

The algorithm outputs $\left(\mathbf{vcd}, (\mathbf{k}, \mathbf{K}), \mathbf{bck}, \mathbf{C}_{\mathtt{pcc}}, \mathbf{c}_{\mathtt{ck}}\right)$

[10] It holds $\mathtt{vcd}_{\mathtt{id}} \neq \mathtt{vcd}_{\mathtt{id}'}$ for any two $\mathtt{id} \neq \mathtt{id}'$.

12.1.1.5 $\mathsf{GenEncLongCodeShares_j}(\mathtt{k'}_j, \mathbf{vcd}, \mathbf{C}_{\mathtt{pcc}}, \mathbf{c}_{\mathtt{ck}}, \mathsf{Logs}_{\mathsf{CCR}_j}, \mathtt{crs})$

This algorithm is executed by the Choice Return Code control component CCR_j. It creates shares of the long Choice Return Codes and a share of the long Vote Cast Return Code in an encrypted form.

It receives as input the CCR_j Choice Return Codes Generation private key $\mathtt{k'}_j$, Voting Card IDs $\mathbf{vcd}$ ciphertexts of the Partial Choice Return Codes $\mathbf{C}_{\mathtt{pcc}}$, ciphertexts of the Confirmation Keys $\mathbf{c}_{\mathtt{ck}}$ (of all voters), and logs $\mathsf{Logs}_{\mathsf{CCR}_j}$. Then it does the following:

1. For each voter $\mathtt{id} \in \mathcal{ID}$ compute the Voter Choice Return Code Generation private key $\mathtt{k}_{j,\mathtt{id}} = \mathsf{KDF}(\mathtt{vcd}_{\mathtt{id}}||\mathtt{k'}_j)$ and the corresponding public key $\mathtt{K}_{j,\mathtt{id}} = g^{\mathtt{k}_{j,\mathtt{id}}}$. Here $\mathtt{k}_{j,\mathtt{id}} \in \mathbb{Z}_q^*$. Let vectors $\mathbf{k}_j = (\mathtt{k}_{j,\mathtt{id}})_{\mathtt{id}\in\mathcal{ID}}$, and $\mathbf{K}_j = (\mathtt{K}_{j,\mathtt{id}})_{\mathtt{id}\in\mathcal{ID}}$.

2. For each voter $\mathtt{id} \in \mathcal{ID}$ compute the Voter Vote Cast Return Code Generation private key $\mathtt{kc}_{j,\mathtt{id}} = \mathsf{KDF}(\mathtt{vcd}_{\mathtt{id}}||\mathsf{ConfirmStr}||\mathtt{k'}_j)$ and the corresponding public key $\mathtt{Kc}_{j,\mathtt{id}} = g^{\mathtt{kc}_{j,\mathtt{id}}}$. Here $\mathtt{kc}_{j,\mathtt{id}} \in \mathbb{Z}_q^*$. Let vectors $\mathbf{kc}_j = (\mathtt{kc}_{j,\mathtt{id}})_{\mathtt{id}\in\mathcal{ID}}$, and $\mathbf{Kc}_j = (\mathtt{Kc}_{j,\mathtt{id}})_{\mathtt{id}\in\mathcal{ID}}$.

3. For each voter $\mathtt{id} \in \mathcal{ID}$ generate (encrypted) shares of the *long* Choice Return Codes:

   - Parse ciphertexts $\mathbf{c}_{\mathtt{pcc},\mathtt{id}} = (\mathbf{c}_{\mathtt{pcc},1,\mathtt{id}}, \ldots, \mathbf{c}_{\mathtt{pcc},n,\mathtt{id}})$ and exponentiate them to $\mathtt{k}_{j,\mathtt{id}}$. Thus, compute:

   $$\mathbf{c}_{\mathtt{expPCC},j,\mathtt{id},i} = \left(\mathbf{c}_{\mathtt{pcc},i,\mathtt{id}}\right)^{\mathtt{k}_{j,\mathtt{id}}} \quad \forall i \in (1,\ldots,n).$$

   Set $\mathbf{c}_{\mathtt{expPCC},j,\mathtt{id}} = (\mathbf{c}_{\mathtt{expPCC},j,\mathtt{id},1}, \ldots, , \mathbf{c}_{\mathtt{expPCC},j,\mathtt{id},n})$, and let $\mathbf{C}_{\mathtt{expPCC},j}$ be the $N \times n$-matrix with rows set to $\mathbf{c}_{\mathtt{expPCC},j,\mathtt{id}}$.

   - Compute a NIZK proof of correct exponentiation:

   $$\pi_{\mathtt{expPCC},j,\mathtt{id}} = \mathsf{ProveExp}\big(((p,q,g,\mathbf{c}_{\mathtt{pcc},\mathtt{id}}),(\mathtt{K}_{j,\mathtt{id}},\mathbf{c}_{\mathtt{expPCC},j,\mathtt{id}})),\mathtt{k}_{j,\mathtt{id}}\big).$$

   $\pi_{\mathtt{expPCC},j,\mathtt{id}}$ proves that $\mathbf{c}_{\mathtt{expPCC},j,\mathtt{id}}$ are computed by raising the group elements of $\mathbf{c}_{\mathtt{pcc},\mathtt{id}}$ to the Voter Choice Return Code Generation private key $\mathtt{k}_{j,\mathtt{id}}$ respectively. Let vector $\boldsymbol{\pi}_{\mathtt{expPCC},j} = (\pi_{\mathtt{expPCC},j,\mathtt{id}})_{\mathtt{id}\in\mathcal{ID}}$.

4. For each voter $\mathtt{id} \in \mathcal{ID}$ generate an (encrypted) share of the *long* Vote Cast Return Code:

   - Exponentiate $\mathbf{c}_{\mathtt{ck},\mathtt{id}}$ to $\mathtt{kc}_{j,\mathtt{id}}$. Thus, compute:

   $$\mathbf{c}_{\mathtt{expCK},j,\mathtt{id}} := \left(\mathbf{c}_{\mathtt{ck},\mathtt{id}}\right)^{\mathtt{kc}_{j,\mathtt{id}}}.$$

   Set vector of ciphertexts $\mathbf{c}_{\mathtt{expCK},j} = (\mathbf{c}_{\mathtt{expCK},j,\mathtt{id}})_{\mathtt{id}\in\mathcal{ID}}$.

   - Compute the following NIZK proof:

   $$\pi_{\mathtt{expCK},j,\mathtt{id}} := \mathsf{ProveExp}\big(((p,q,g,\mathbf{c}_{\mathtt{ck},\mathtt{id}}),(\mathtt{Kc}_{j,\mathtt{id}},\mathbf{c}_{\mathtt{expCK},j,\mathtt{id}})),\mathtt{kc}_{j,\mathtt{id}}\big).$$

   $\pi_{\mathtt{expCK},j,\mathtt{id}}$ proves that $\mathbf{c}_{\mathtt{expCK},j,\mathtt{id}}$ is computed by raising the elements of $\mathbf{c}_{\mathtt{ck},\mathtt{id}}$ to the Voter Vote Cast Return Code Generation private key $\mathtt{kc}_{j,\mathtt{id}}$. Let vector $\boldsymbol{\pi}_{\mathtt{expCK},j} = (\pi_{\mathtt{expCK},j,\mathtt{id}})_{\mathtt{id}\in\mathcal{ID}}$.

The algorithm outputs $((\mathbf{k}_j, \mathbf{K}_j), (\mathbf{kc}_j, \mathbf{Kc}_j), (\mathbf{C}_{\mathtt{expPCC},j}, \boldsymbol{\pi}_{\mathtt{expPCC},j}), (\mathbf{c}_{\mathtt{expCK},j}, \boldsymbol{\pi}_{\mathtt{expCK},j}))$.

12.1.1.6 CombineEncLongCodeShares($\mathbf{K}_j, \mathbf{Kc}_j, \mathbf{C}_{\mathtt{expPCC},j}, \boldsymbol{\pi}_{\mathtt{expPCC},j}, \mathbf{c}_{\mathtt{expCK},j}, \boldsymbol{\pi}_{\mathtt{expCK},j}, \mathsf{Logs}_{\mathtt{PO}}$)

This algorithm is executed by the Print Office. It combines the (encrypted) long codes shares generated by the Choice Return Code control components CCR_j.

It receives as input the matrix of encrypted long Choice Return Code shares $\mathbf{C}_{\mathtt{expPCC},j}$, the vector of encrypted long Vote Cast Return Code shares $\mathbf{c}_{\mathtt{expCK},j}$, and the vectors of Generation public keys $\mathbf{K}_j, \mathbf{Kc}_j$ (along with NIZKs of correct exponentiations) from each Choice Return Code control component CCR_j. Then it does the following:

1. Combine the encrypted long Choice Return Codes. Thus for each $j \in (1, \ldots, m)$, parse each row of $\mathbf{C}_{\mathtt{expPCC},j}$ as a vector of ciphertexts $\mathbf{c}_{\mathtt{expPCC},j,\mathtt{id}} = (\mathbf{c}_{\mathtt{expPCC},j,\mathtt{id},i})_{i=1}^{n}$ and multiply:

$$\mathbf{c}_{\mathtt{lcc},\mathtt{id},i} = \prod_{j=1}^{m} \mathbf{c}_{\mathtt{expPCC},j,\mathtt{id},i} \quad \forall i \in (1, \ldots, n).$$

   Let $\mathbf{C}_{\mathtt{lcc}}$ be the $N \times n$-matrix, where each row is set to $(\mathbf{c}_{\mathtt{lcc},\mathtt{id},i})_{i=1}^{n}$.

2. Combine the encrypted long Vote Cast Return Codes. Thus for each $j \in (1, \ldots, m)$, parse the vector of ciphertexts $\mathbf{c}_{\mathtt{expCK},j} = (\mathbf{c}_{\mathtt{expCK},j,\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$ and multiply:

$$\mathbf{c}_{\mathtt{lvc},\mathtt{id}} = \prod_{j=1}^{m} \mathbf{c}_{\mathtt{expCK},j,\mathtt{id}}. \quad \forall \mathtt{id} \in \mathcal{ID}.$$

   Let the vector of ciphertexts $\mathbf{c}_{\mathtt{lvc}} = (\mathbf{c}_{\mathtt{lvc},\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$.

3. Updates logs:

$$\mathsf{Logs}_{\mathtt{PO}} \leftarrow \mathsf{Logs}_{\mathtt{PO}} \cup \{\mathbf{K}_j, \mathbf{Kc}_j, \mathbf{C}_{\mathtt{expPCC},j}, \boldsymbol{\pi}_{\mathtt{expPCC},j}, \mathbf{c}_{\mathtt{expCK},j}, \boldsymbol{\pi}_{\mathtt{expCK},j}\}.$$

The algorithm outputs $(\mathbf{C}_{\mathtt{lcc}}, \mathbf{c}_{\mathtt{lvc}})$.

12.1.1.7 GenCMTable($\mathbf{vcd}, \mathbf{C}_{\mathtt{lcc}}, \mathbf{c}_{\mathtt{lvc}}, \mathtt{C}_{\mathtt{sk}}, \mathtt{sk}_{\mathtt{SDM}}, \mathtt{crs}$)

This algorithm is run by the Print Office to generate the Choice Return Codes and the short Vote Cast Return Code, to encrypt them with the Choice Return Code encryption symmetric key (derived from the long Choice Return Code) and the Vote Cast Return Code encryption symmetric key (derived from the long Vote Cast Return Code), and to store the encryptions in the Codes Mapping table.

The algorithm takes as input the Voting Cards IDs $\mathbf{vcd}$, the encrypted long codes $(\mathbf{C}_{\mathtt{lcc}}, \mathbf{c}_{\mathtt{lvc}})$, the Codes Secret key $\mathtt{C}_{\mathtt{sk}}$, the Secure Data Manager decryption key $\mathtt{sk}_{\mathtt{SDM}}$, and logs $\mathsf{Logs}_{\mathtt{PO}}$. Then, it does the following:

1. Parse the N-long vector $\mathbf{vcd} = (\mathtt{vcd}_{\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$.

2. Generate the long Choice Return Codes :

   - Parse each of the N rows of $\mathbf{C}_{\mathtt{lcc}}$ as $(\mathbf{c}_{\mathtt{lcc},\mathtt{id},i})_{i=1}^{n}$ and decrypt each ciphertext:

$$\mathtt{pC}_{i,\mathtt{id}} \leftarrow \mathsf{Dec}(\mathtt{sk}_{\mathtt{SDM}}, \mathbf{c}_{\mathtt{lcc},\mathtt{id},i}) \quad \forall i \in (1,\ldots,n).$$

   - Set the n long Choice Return Codes for voter $\mathtt{id}$ to:

$$\mathtt{lCC}_{i,\mathtt{id}} = \mathsf{H}(\mathtt{pC}_{i,\mathtt{id}} || \mathtt{vcd}_{\mathtt{id}}) \quad \forall i \in (1,\ldots,n).$$

3. Generate the long Vote Cast Return Codes:

   - Parse the N-long vector $\mathbf{c}_{\mathtt{lvc}} = (\mathbf{c}_{\mathtt{lvc},\mathtt{id}})_{\mathtt{id}\in\mathcal{ID}}$ and decrypt each ciphertext:

$$\mathtt{pVCC}_{\mathtt{id}} = \mathsf{Dec}(\mathtt{sk}_{\mathtt{SDM}}, \mathbf{c}_{\mathtt{lvc},\mathtt{id}}).$$

   - Set the long Vote Cast Return code to:

$$\mathtt{lVCC}_{\mathtt{id}} = \mathsf{H}(\mathtt{pVCC}_{\mathtt{id}} || \mathtt{vcd}_{\mathtt{id}}).$$

4. Generate random short Choice Return Codes $\mathtt{CC}_{i,\mathtt{id}} \leftarrow \mathcal{C}_{cc}$ for each voting option $v_i \in \mathbf{v}$ and voter $\mathtt{id} \in \mathcal{ID}$. Let vector $\mathbf{cc}_{\mathtt{id}} = (\mathtt{CC}_{i,\mathtt{id}})_{\mathtt{id}\in\mathcal{ID}}$, and $\mathbf{CC}$ the $N \times n$-matrix with each row set to $\mathbf{cc}_{\mathtt{id}}$.

5. Generate random short Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}} \leftarrow \mathcal{C}_{vcc}$ for voter $\mathtt{id} \in \mathcal{ID}$. Let vector $\mathbf{vcc} = (\mathtt{VCC}_{\mathtt{id}})_{\mathtt{id}\in\mathcal{ID}}$.

6. Generate the codes mapping table:

   - Compute hashes $\mathtt{hlCC}_{i,\mathtt{id}} = \mathsf{H}(\mathtt{lCC}_{i,\mathtt{id}} || \mathtt{C}_{\mathtt{sk}})$ and $\mathtt{hlVCC}_{\mathtt{id}} = \mathsf{H}(\mathtt{lVCC}_{\mathtt{id}} || \mathtt{C}_{\mathtt{sk}})$. Then derive the symmetric encryption keys $\mathtt{skcc}_{i,\mathtt{id}}$, $\mathtt{skvcc}_{\mathtt{id}}$ using a key derivation function[11]

$$\begin{aligned} \mathtt{skcc}_{i,\mathtt{id}} &= \mathsf{KDF}(\mathtt{hlCC}_{i,\mathtt{id}}) \quad \forall i \in (1,\ldots,n),\ \forall \mathtt{id} \in \mathcal{ID}, \\ \mathtt{skvcc}_{\mathtt{id}} &= \mathsf{KDF}(\mathtt{hlVCC}_{\mathtt{id}}) \quad \forall \mathtt{id} \in \mathcal{ID}. \end{aligned}$$

   - Encrypt the short Choice Return Code $\mathtt{CC}_{i,\mathtt{id}}$ and the short Vote Cast Return Code:

$$\begin{aligned} \mathtt{ctCC}_{\mathtt{id},i} &\leftarrow \mathsf{Enc}_{\mathtt{s}}(\mathtt{CC}_{i,\mathtt{id}}; \mathtt{skcc}_{i,\mathtt{id}}) \quad \forall i \in (1,\ldots,n),\ \forall \mathtt{id} \in \mathcal{ID}, \\ \mathtt{ctVCC}_{\mathtt{id}} &\leftarrow \mathsf{Enc}_{\mathtt{s}}(\mathtt{VCC}_{\mathtt{id}}; \mathtt{skvcc}_{\mathtt{id}}) \end{aligned}$$

   The mapping between long and short codes for voter with Voting Card ID $\mathtt{vcd}_{\mathtt{id}}$ is defined as

$$\mathsf{CMtable}_{\mathtt{id}} = \left\{ \{[\mathsf{H}(\mathtt{lCC}_{i,\mathtt{id}}), \mathtt{ctCC}_{\mathtt{id},i}]\}_{i=1}^{n}, [\mathsf{H}(\mathtt{lVCC}_{\mathtt{id}}), \mathtt{ctVCC}_{\mathtt{id}}] \right\}.$$

[11] KDF stands for the mask generation function defined in ISO-18033-2 and in PKCS#1v2.2.

- Set the code mapping table to $\mathsf{CMtable} := \{\mathsf{CMtable}_{\mathtt{id}}\}_{\mathtt{id} \in \mathcal{ID}}$. The entries of the table are shuffled to avoid trivial correlation.

- Updates logs:

$$\mathsf{Logs}_{\mathtt{PO}} \leftarrow \mathsf{Logs}_{\mathtt{PO}} \cup \{\mathsf{CMtable}\}.$$

The algorithm outputs $(\mathbf{CC}, \mathbf{vcc}, \mathsf{CMtable})$.

12.1.1.8 GenCredDat($\mathbf{k}$, crs)

This algorithm is executed by the Print Office. It creates credential material for the voters.

The algorithm takes as input the Verification Card private keys $\mathbf{k} = (\mathtt{k}_{\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$. Then for each voter $\mathtt{id} \in \mathcal{ID}$ it does the following:

1. Generate a random Start Voting Key $\mathtt{SVK}_{\mathtt{id}} \xleftarrow{\$} \mathcal{C}_{svk}$. Set vector $\mathbf{svk} = (\mathtt{SVK}_{\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$.

2. Generate a keystore symmetric encryption key as:

$$\mathtt{KSkey}_{\mathtt{id}} \leftarrow \mathsf{PBKDF}(\mathtt{SVK}_{\mathtt{id}}, \mathtt{KEYseed}).$$

   [12] Set vector $\mathbf{KSkeys} = (\mathtt{KSkey}_{\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$.

3. Symmetrically encrypt the Verification Card private key with the keystore symmetric encryption key: $\mathtt{VCks}_{\mathtt{id}} \leftarrow \mathsf{Enc}_{\mathtt{s}}(\mathtt{k}_{\mathtt{id}}; \mathtt{KSkey}_{\mathtt{id}})$. Set vector $\mathbf{VCks} = (\mathtt{VCks}_{\mathtt{id}})_{\mathtt{id} \in \mathcal{ID}}$

The algorithm outputs $(\mathbf{svk}, \mathbf{KSkeys}, \mathbf{VCks})$.

12.1.2 Protocol SetupTally

It is an interactive protocol run between the mixing control components CCMs and the Print Office. It consists of the following algorithms.

12.1.2.1 SetupTallyCCM$_j$(crs, Logs$_{\mathsf{CCM}_j}$)

This algorithm is executed by the Mixing control component CCM_j to generate its share of the Election key pair.

The algorithm takes as input the ElGamal parameters gparams and logs $\mathsf{Logs}_{\mathsf{CCM}_j}$. Then it does the following:

1. Compute the Election key pair share $(\mathtt{EL}_{\mathtt{pk},j}, \mathtt{EL}_{\mathtt{sk},j}) \leftarrow \mathsf{KeyGen}(\mathtt{gparams})$

2. Update logs $\mathsf{Logs}_{\mathsf{CCM}_j} \leftarrow \mathsf{Logs}_{\mathsf{CCM}_j} \cup \{\mathtt{EL}_{\mathtt{pk},j}\}$

The algorithm outputs the key pair $(\mathtt{EL}_{\mathtt{pk},j}, \mathtt{EL}_{\mathtt{sk},j})$.

[12] Where string KEYseed is a salt.

12.1.2.2 SetupTallySDM($\mathtt{crs}, \{\mathtt{EL}_{\mathtt{pk},j}\}_{j=1}^{m'-1}, \mathsf{Logs}_{\mathtt{PO}}$)

The algorithm takes as input the mixing public key $\{\mathtt{EL}_{\mathtt{pk},j}\}_{j=1}^{m'-1}$ of the online Mixing control components $\mathrm{CCM}_1, \ldots, \mathrm{CCM}_{m'-1}$ and logs $\mathsf{Logs}_{\mathtt{PO}}$. The it does the following:

1. Compute the Election key pair share of the offline Mixing control component $(\mathtt{EB}_{\mathtt{pk}}, \mathtt{EB}_{\mathtt{sk}}) \leftarrow \mathsf{KeyGen}(\mathtt{gparams})$.
2. Compute the Election public key $\mathtt{EL}_{\mathtt{pk}} = \mathtt{EB}_{\mathtt{pk}} \cdot \prod_{j=1}^{m'-1} \mathtt{EL}_{\mathtt{pk},j}$.
3. Update logs $\mathsf{Logs}_{\mathtt{PO}} \leftarrow \mathsf{Logs}_{\mathtt{PO}} \cup \{\mathtt{EL}_{\mathtt{pk}}, \{\mathtt{EL}_{\mathtt{pk},j}\}_{j=1}^{m'-1}, \mathtt{EB}_{\mathtt{pk}}\}$.

The algorithm outputs $\mathtt{EL}_{\mathtt{pk}}$ and $(\mathtt{EB}_{\mathtt{pk}}, \mathtt{EB}_{\mathtt{sk}})$.

12.1.3 VerifyConfigPhase($\mathsf{Logs}_{\mathtt{PO}}, \mathtt{crs}$)

This algorithm is run by an auditor. It verifies the proper execution of the configuration phase.

The algorithm takes as input the logs $\mathsf{Logs}_{\mathtt{PO}}$ of the Print Office. Then it does the following:

1. Extract from the $\mathsf{Logs}_{\mathtt{PO}}$ the following:
   - The output ciphertext matrix $\mathbf{C}_{\mathtt{pcc}}$ of Partial Choice Return Codes, and the output ciphertext vector $\mathbf{c}_{\mathtt{ck}}$ of Confirmation Keys of Algorithm 12.1.1.4.
   - The input public keys, ciphertexts and NIZK proofs
   $$(\mathbf{K}_j, \mathbf{Kc}_j, \mathbf{C}_{\mathtt{expPCC},j}, \boldsymbol{\pi}_{\mathtt{expPCC},j}, \mathbf{c}_{\mathtt{expCK},j}, \boldsymbol{\pi}_{\mathtt{expCK},j}),$$
   of Algorithm 12.1.1.6.
2. Verify the NIZK proofs of correct Partial Choice Return Codes exponentiation $\forall \mathtt{id} \in \mathcal{ID}, \forall j \in (1, m)$:
   $$b_{\mathtt{id},i} \leftarrow \mathsf{VerifyExp}(((p, q, g, \mathbf{c}_{\mathtt{pcc},\mathtt{id}}), (\mathtt{K}_{j,\mathtt{id}}, \mathbf{c}_{\mathtt{expPCC},j,\mathtt{id}})), \boldsymbol{\pi}_{\mathtt{expPCC},j})$$
   If some $b_{\mathtt{id},i} = 0$, abort and output $\perp$.
3. Verifiy the NIZK proofs of correct Confirmation Keys exponentiations $\forall \mathtt{id} \in \mathcal{ID}, \forall j \in (1, m)$:
   $$b_{\mathtt{id}} \leftarrow \mathsf{VerifyExp}(((p, q, g, \mathbf{c}_{\mathtt{ck},\mathtt{id}}), (\mathtt{Kc}_{j,\mathtt{id}}, \mathbf{c}_{\mathtt{expCK},j,\mathtt{id}})), \pi_{\mathtt{expCK},j,\mathtt{id}}).$$
   If $b_{\mathtt{id}} = 0$ abort and output $\perp$.

If, and only if, all verifications were successful, the algorithm outputs $\top$.

12.2 Voting phase protocols

The execution flow of the algorithms in the voting phase is depicted in Figure 18 (send vote) and Figure 19 (confirm vote).

12.2.1 Protocol SendVote

It is an interactive protocol run between the voter, the Voting Client , the server and the Choice Return Code control components CCRs. It consists of the following algorithms.

12.2.1.1 GetKey($\mathtt{SVK_{id}}, \mathtt{VCks_{id}}$)

This algorithm is executed by the Voting Client . It takes as input a Start Voting Key $\mathtt{SVK_{id}}$ and a Verification Card keystore $\mathtt{VCks_{id}}$ and it obtains a Verification Card private key $\mathtt{k_{id}}$ as follows:

- Generate the keystore encryption symmetric key:

$$\mathtt{KSkey_{id}} \leftarrow \mathsf{PBKDF}(\mathtt{SVK_{id}}, \mathtt{KEYseed}).$$

- Run the $\mathsf{Dec_s}$ algorithm with inputs $\mathtt{VCks_{id}}$ and $\mathtt{KSkey_{id}}$, and recover the Verification Card private key $\mathtt{k_{id}}$.

It outputs the Verification Card private key $\mathtt{k_{id}}$.

12.2.1.2 CreateVote $(\mathtt{vcd_{id}}, \mathbf{v}_{\mathtt{id}}, \mathtt{EL_{pk}}, \mathbf{pk}_{\mathrm{CCR}}, (\mathtt{K_{id}}, \mathtt{k_{id}}), \mathtt{crs})$

This algorithm is executed by the Voting Client to create the ballot with the voting options selected by the Voter.

It takes as input the Voting Card ID $\mathtt{vcd_{id}}$, the voting options $\mathbf{v}_{\mathtt{id}}$, the Election public key $\mathtt{EL_{pk}}$, the global Choice Return Code public keys $\mathbf{pk}_{\mathrm{CCR}} = (\mathtt{pk}_{\mathrm{CCR},1}, \ldots, \mathtt{pk}_{\mathrm{CCR},\psi})$, and the Verification Card key pair $(\mathtt{K_{id}}, \mathtt{k_{id}})$. Then it does the following:

1. Parse $\mathbf{v}_{\mathtt{id}} = (v_{j_1}, \ldots, v_{j_\psi})$ and compute: $\nu = \prod_{i=1}^{\psi} v_{j_i} \mod p$.
2. Encrypt the aggregation: $\mathtt{E1} = (c_{1,0}, c_{1,1}) \leftarrow \mathsf{Enc}(\nu, \mathtt{EL_{pk}}; r)$.
3. Generate a Schnorr proof $\pi_{\mathsf{sch}} \leftarrow \mathsf{ProveSch}((p, q, g, \mathtt{E1}), r, \mathtt{aux})$. The auxiliary information $\mathtt{aux}$ contains the Voting card ID $\mathtt{vcd_{id}}$.
4. Computes the Partial Choice Return Codes as

$$\mathbf{pCC}_{\mathtt{id}} = (\mathtt{pCC}_{1,\mathtt{id}}, \ldots, \mathtt{pCC}_{\psi,\mathtt{id}}) = (v_{j_1}^{\mathtt{k_{id}}}, \ldots, v_{j_\psi}^{\mathtt{k_{id}}}).$$

5. Compute a multi-recipient ElGamal encryption of these codes as

$$\mathtt{E2} = (c_{2,0}, c_{2,1}, \ldots, c_{2,\psi}) \leftarrow \mathsf{MultiEnc}\left(\mathbf{pk}_{\mathrm{CCR}}, \mathbf{pCC}_{\mathtt{id}}; r'\right).$$

6. Exponentiate $\mathtt{E1}$ and aggregate $\mathtt{E2}$:

$$\begin{aligned} \widetilde{\mathtt{E1}} &= (\widetilde{c}_{1,0}, \widetilde{c}_{1,1}) = (c_{1,0}^{\mathtt{k_{id}}}, c_{1,1}^{\mathtt{k_{id}}}), \\ \widetilde{\mathtt{E2}} &= (\widetilde{c}_{2,0}, \widetilde{c}_{2,1}) = \Big(c_{2,0}, \prod_{i=1}^{\psi} c_{2,i}\Big). \end{aligned}$$

Note that $\widetilde{\texttt{E1}}$ encrypts $\nu^{\texttt{k}_{\texttt{id}}}$ under public key $\texttt{EL}_{\texttt{pk}}$ with randomness $r \cdot \texttt{k}_{\texttt{id}}$, and $\widetilde{\texttt{E2}}$ encrypts $\prod_{i=1}^{\psi} \texttt{pCC}_{i,\texttt{id}} = \nu^{\texttt{k}_{\texttt{id}}}$ under public key $\prod_{i=1}^{\psi} \texttt{pk}_{\text{CCR},i}$ with randomness r'.

7. Generate two NIZK proofs to to link E1 and E2:

   - $\pi_{\text{Exp}} \leftarrow \text{ProveExp}\big(((p, q, g, \texttt{E1}), (\texttt{K}_{\texttt{id}}, \widetilde{\texttt{E1}})), \texttt{k}_{\texttt{id}}\big)$ proves that $\widetilde{\texttt{E1}}$ is computed by raising the elements of E1 to the Verification Card private key $\texttt{k}_{\texttt{id}}$.
   - $\pi_{\text{EqEnc}} \leftarrow \text{ProveEqEnc}\Big(\big(p, q, g, \texttt{EL}_{\texttt{pk}}, \prod_{i=1}^{\psi} \texttt{pk}_{\text{CCR},i}, \widetilde{\texttt{E1}}, \widetilde{\texttt{E2}}\big), (r \cdot \texttt{k}_{\texttt{id}}, r')\Big)$ proves that $\widetilde{\texttt{E1}}$ and $\widetilde{\texttt{E2}}$ encrypt plaintext $\nu^{\texttt{k}_{\texttt{id}}}$ under public keys $\texttt{EL}_{\texttt{pk}}$ and $\prod_{i=1}^{\psi} \texttt{pk}_{\text{CCR},i}$.

It outputs $\texttt{b}_{\texttt{id}} = (\texttt{vcd}_{\texttt{id}}, \texttt{E1}, \texttt{E2}, \widetilde{\texttt{E1}}, \texttt{K}_{\texttt{id}}, \boldsymbol{\pi}_{\text{ballot}})$, where $\boldsymbol{\pi}_{\text{ballot}} = (\pi_{\text{sch}}, \pi_{\text{Exp}}, \pi_{\text{EqEnc}})$.

12.2.1.3 VerifyBallotServ $\big(\texttt{b}_{\texttt{id}}, \texttt{EL}_{\texttt{pk}}, \mathbf{pk}_{\text{CCR}}, \texttt{K}_{\texttt{id}}, \text{Logs}_{\texttt{serv}}, \texttt{crs}\big)$

This algorithm is executed by the Server to verify the NIZKs of the ballot.

It receives as input a ballot $\texttt{b}_{\texttt{id}}$, the Election public key $\texttt{EL}_{\texttt{pk}}$, the global Choice Return Code control component public keys $\mathbf{pk}_{\text{CCR}} = \big(\texttt{pk}_{\text{CCR},1}, \ldots, \texttt{pk}_{\text{CCR},\psi}\big)$, Verification Card public key $\texttt{K}_{\texttt{id}}$ and the logs $\text{Logs}_{\texttt{serv}}$. Then it does the following:

1. Parse $\texttt{b}_{\texttt{id}}$ as $(\texttt{vcd}_{\texttt{id}}, \texttt{E1}, \texttt{E2}, \widetilde{\texttt{E1}}, \texttt{K}_{\texttt{id}}, \boldsymbol{\pi}_{\text{ballot}})$.
2. Extract from $\text{Logs}_{\texttt{serv}}$ the list of valid votes $\text{L}_{\texttt{validVotes}}$[13]. If $\texttt{vcd}_{\texttt{id}}$ is present in $\text{L}_{\texttt{validVotes}}$ abort and output $\perp$.
3. Otherwise parse E2 as $(c_{2,0}, c_{2,1}, \ldots, c_{2,\psi})$ and compute $\widetilde{\texttt{E2}} = \big(c_{2,0}, \prod_{i=1}^{\psi} c_{2,i}\big)$.
4. Validate NIZK proofs $\boldsymbol{\pi}_{\text{ballot}} = (\pi_{\text{sch}}, \pi_{\text{Exp}}, \pi_{\text{EqEnc}})$ by running:

   - $b_1 \leftarrow \text{VerifySch}((p, q, g, \texttt{E1}), \pi_{\text{sch}}, \texttt{aux})$, where aux includes the Voting Card ID $\texttt{vcd}_{\texttt{id}}$.
   - $b_2 \leftarrow \text{VerifyExp}(((p, q, g, \texttt{E1}), (\texttt{K}_{\texttt{id}}, \widetilde{\texttt{E1}})), \pi_{\text{Exp}})$.
   - $b_3 \leftarrow \text{VerifyEqEnc}((p, q, g, \texttt{EL}_{\texttt{pk}}, \prod_{i=1}^{\psi} \texttt{pk}_{\text{CCR},i}, \widetilde{\texttt{E1}}, \widetilde{\texttt{E2}}), \pi_{\text{EqEnc}})$.

   If some $b_i = 0$ abort and output $(0, \text{Logs}_{\texttt{serv}})$.
5. Extract the list $\text{L}_{\texttt{validVotes}}$ of valid ballots from $\text{Logs}_{\texttt{serv}}$, and find entry $(\texttt{b}_{\texttt{id}})$. If there is no entry, create entry $(\texttt{b}_{\texttt{id}})$. Also, update the logs with the list.

It outputs $(1, \text{Logs}_{\texttt{serv}})$. (If reached this point the ballot verifies successfully.)

[13] Voters cannot cast a vote twice.

12.2.1.4 $\mathsf{VerifyBallotCCR_j}\left(\mathtt{b_{id}}, \mathtt{EL_{pk}}, \mathbf{pk}_{\mathrm{CCR}}, \mathtt{K_{id}}, \mathsf{Logs}_{\mathrm{CCR}_j}, \mathtt{crs}\right)$

This algorithm is executed by the Choice Return Code control components to verify the NIZKs of the ballot.

It receives as input a ballot $\mathtt{b_{id}}$, the Election public key $\mathtt{EL_{pk}}$, the global Choice Return Code control component public keys $\mathbf{pk}_{\mathrm{CCR}} = \left(\mathtt{pk}_{\mathrm{CCR},1}, \ldots, \mathtt{pk}_{\mathrm{CCR},\psi}\right)$,Verification Card public key $\mathtt{K_{id}}$ and the logs $\mathsf{Logs}_{\mathrm{CCR}_j}$. Then it does the following:

1. Parse $\mathtt{b_{id}}$ as $(\mathtt{vcd_{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K_{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$.
2. Extract the list $\mathsf{L}_{\mathtt{queriedVotes},j}$ of requested ballots from the logs. If $\mathtt{vcd_{id}}$ is present in some ballot $\mathtt{b} \in \mathsf{L}_{\mathtt{queriedVotes},j}$ abort and output $(0, \mathsf{Logs}_{\mathrm{CCR}_j})$. Otherwise update the list $\mathsf{L}_{\mathtt{queriedVotes},j} \leftarrow \mathsf{L}_{\mathtt{queriedVotes},j} \cup (\mathtt{b_{id}})$.
3. Parse $\mathtt{E2}$ as $(c_{2,0}, c_{2,1}, \ldots, c_{2,\psi})$ and compute $\widetilde{\mathtt{E2}} = \left(c_{2,0}, \prod_{i=1}^{\psi} c_{2,i}\right)$.
4. Validate NIZK proofs $\boldsymbol{\pi}_{\mathsf{ballot}} = (\pi_{\mathsf{sch}}, \pi_{\mathsf{Exp}}, \pi_{\mathsf{EqEnc}})$ by running $\boldsymbol{\pi}_{\mathsf{ballot}}$:
   - $b_1 \leftarrow \mathsf{VerifySch}((p, q, g, \mathtt{E1}), \pi_{\mathsf{sch}}, \mathtt{aux})$.
   - $b_2 \leftarrow \mathsf{VerifyExp}(((p, q, g, \mathtt{E1}), (\mathtt{K_{id}}, \widetilde{\mathtt{E1}})), \pi_{\mathsf{Exp}})$.
   - $b_3 \leftarrow \mathsf{VerifyEqEnc}((p, q, g, \mathtt{EL_{pk}}, \prod_{i=1}^{\psi} \mathtt{pk}_{\mathrm{CCR},i}, \widetilde{\mathtt{E1}}, \widetilde{\mathtt{E2}}), \pi_{\mathsf{EqEnc}})$.

   If some $b_i = 0$ abort and output $(0, \mathsf{Logs}_{\mathtt{serv}})$.
5. Otherwise, extract the list $\mathsf{L}_{\mathtt{validVotes},j}$ of valid votes from the logs and update it: $\mathsf{L}_{\mathtt{validVotes},j} \leftarrow \mathsf{L}_{\mathtt{validVotes},j} \cup (\mathtt{b_{id}})$. Also, update the logs with the list.

It outputs $(1, \mathsf{Logs}_{\mathrm{CCR}_j})$. (If reached this point the ballot verifies successfully.)

12.2.1.5 $\mathsf{PartialDecryptPCC_j}(\mathtt{b_{id}}, \mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{sk}_{\mathrm{CCR}_j}, \mathsf{Logs}_{\mathrm{CCR}_j}, \mathtt{crs})$

This algorithm is executed in the Choice Return Codes control component CCR_j to partially decrypt the second ciphertext $\mathtt{E2}$ of the ballot.

It receives as input a ballot $\mathtt{b_{id}}$, the Choice Return Code control component key pairs $(\mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{sk}_{\mathrm{CCR}_j})$ and the logs $\mathsf{Logs}_{\mathrm{CCR}_j}$. Then it does the following:

1. Parse ballot as $\mathtt{b_{id}} = (\mathtt{vcd_{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K_{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$.
2. Extract lists $\mathsf{L}_{\mathtt{validVotes},j}$ and $\mathsf{L}_{\mathtt{decPCC},j}$ from logs. The latter list contains ballots already processed in this algorithm. If $\mathtt{b_{id}} \notin \mathsf{L}_{\mathtt{validVotes},j}$ or if $\mathtt{vcd_{id}}$ is present in some ballot $\mathtt{b} \in \mathsf{L}_{\mathtt{decPCC},j}$ abort and output $(0, \mathsf{Logs}_{\mathrm{CCR}_j})$.[14]

[14] Only ballots with valid NIZKs are processed: list $\mathsf{L}_{\mathtt{validVotes},j}$ is updated in Algorithm 12.2.1.4.

3. Parse $\texttt{E2}$ as $(c_{2,0}, c_{2,1}, \ldots, c_{2,\psi})$ and compute the DDH masks to decrypt $\texttt{E2}$. Thus, compute:

$$\mathbf{d}_{j,\texttt{id}} = (d_{j,1}, \ldots, d_{j,\psi}) = \left(c_{2,0}^{\texttt{sk}_{\text{CCR}_j,1}}, \ldots, c_{2,0}^{\texttt{sk}_{\text{CCR}_j,\psi}}\right).$$

4. Computes the following NIZK proof:

$$\pi_{\texttt{decPCC},j} \leftarrow \mathsf{ProveExp}\left(((p, q, g, c_{2,0}), (\prod_{i=1}^{\psi} \texttt{pk}_{\text{CCR}_j,i}, \prod_{i=1}^{\psi} d_{j,i})), \sum_{i=1}^{\psi} \texttt{sk}_{\text{CCR}_j,i}\right)$$

5. Update the list $\mathsf{L}_{\texttt{decPCC},j} \leftarrow \mathsf{L}_{\texttt{decPCC},j} \cup (\texttt{b}_{\texttt{id}}, \mathbf{d}_{j,\texttt{id}}, \pi_{\texttt{decPCC},j})$. Also update the logs with the list.

It outputs $(\mathbf{d}_{j,\texttt{id}}, \mathsf{Logs}_{\text{CCR}_j})$.

12.2.1.6 DecryptPCC $\left(\texttt{b}_{\texttt{id}}, \{\mathbf{d}_{j,\texttt{id}}\}_{j=1}^{m}, \mathsf{Logs}_{\texttt{serv}}, \texttt{crs}\right)$

This algorithm is executed by the Server to decrypt the second ciphertext $\texttt{E2}$ of the ballot.

It receives as input the ballot $\texttt{b}_{\texttt{id}}$, the DDH masks contributions $\mathbf{d}_{j,\texttt{id}} = (d_{j,1}, \ldots, d_{j,\psi})$ from each CCR_j (see algorithm $\mathsf{PartialDecryptPCC_j}$), and the logs $\mathsf{Logs}_{\texttt{serv}}$. Then it does the following:

1. Parse ballot as $\texttt{b}_{\texttt{id}} = (\texttt{vcd}_{\texttt{id}}, \texttt{E1}, \texttt{E2}, \widetilde{\texttt{E1}}, \texttt{K}_{\texttt{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$.

2. Parse $\texttt{E2}$ as $(c_{2,0}, c_{2,1}, \ldots, c_{2,\psi})$.

3. Merge the DDH mask contributions. Thus, compute:

$$d_i = \prod_{j=1}^{m} d_{j,i} \quad \forall i \in (1, \ldots, \psi),$$

   and let $\mathbf{d}_{\texttt{id}} = (d_1, \ldots, d_\psi)$.

4. Decrypt the multi-recipient ciphertext $\texttt{E2}$ to obtain the Partial Choice Return codes as follows:

$$\texttt{pCC}_{i,\texttt{id}} = \frac{c_{2,i}}{d_i}. \quad \forall i \in (1, \ldots, \psi).$$

   Denote $\mathbf{pCC}_{\texttt{id}} = (\texttt{pCC}_{1,\texttt{id}}, \ldots, \texttt{pCC}_{\psi,\texttt{id}})$.

5. Extract the list $\mathsf{L}_{\texttt{decPCC}}$ of votes for which decryption has been already done from $\mathsf{Logs}_{\text{CCR}_j}$ and do the following:

   - If there is an entry $(\texttt{b}_{\texttt{id}}, \star, \star, \star)$ abort an output $(\perp, \mathsf{Logs}_{\texttt{serv}})$.
   - Otherwise, update the list $\mathsf{L}_{\texttt{decPCC}} \leftarrow \mathsf{L}_{\texttt{decPCC}} \cup (\texttt{b}_{\texttt{id}}, \mathbf{pCC}_{\texttt{id}}, \mathbf{d}_{\texttt{id}}, \{\mathbf{d}_{j,\texttt{id}}\}_{j=1}^{m})$. Also, update the logs with the list.

It outputs $(\mathbf{pCC}_{\texttt{id}}, \mathsf{Logs}_{\texttt{serv}})$.

12.2.1.7 $\mathsf{CreateLCCShare_j}(\mathtt{b_{id}}, \mathbf{pCC}_{\mathtt{id}}, \mathtt{k}_{j,\mathtt{id}}, \mathtt{K}_{j,\mathtt{id}}, \mathsf{Logs}_{\mathsf{CCR}_j})$

This algorithm is executed by the Choice Return Code control component CCR_j to generate shares of the long Choice Return Codes.

It takes as input a ballot $\mathtt{b_{id}}$, Partial Choice Return codes $\mathbf{pCC}_{\mathtt{id}}$, the Voter Choice Return Code Generator key pair $(\mathtt{k}_{j,\mathtt{id}}, \mathtt{K}_{j,\mathtt{id}})$ and logs $\mathsf{Logs}_{\mathsf{CCR}_j}$. Then it does the following:

1. Parse ballot as $\mathtt{b_{id}} = (\mathtt{vcd_{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K_{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$.
2. Extract the list $\mathsf{L}_{\mathtt{sentVotes},j}$, list of ballots already requested for this algorithm, and list $\mathsf{L}_{\mathtt{decPCC},j}$ [15] from $\mathsf{Logs}_{\mathsf{CCR}_j}$.
3. If $\mathtt{b_{id}} \notin \mathsf{L}_{\mathtt{decPCC},j}$ or $\mathtt{vcd_{id}}$ is present in some ballot $\mathtt{b} \in \mathsf{L}_{\mathtt{sentVotes},j}$ abort and output $(\perp, \mathsf{Logs}_{\mathsf{CCR}_j})$.
4. Parse the partial Choice Return Codes as $\mathbf{pCC}_{\mathtt{id}} = (\mathtt{pCC}_{1,\mathtt{id}}, \ldots, \mathtt{pCC}_{\psi,\mathtt{id}})$.
5. Compute the long Choice Return Code shares as follows:
   - Hash and convert to group element:
   $$\mathtt{hpCC}_{i,\mathtt{id}} = (\mathsf{H}(\mathtt{pCC}_{i,\mathtt{id}}))^2 \quad \forall i \in (1, \ldots, \psi).$$
   - Exponentiate to $\mathtt{k}_{j,\mathtt{id}}$:
   $$\mathtt{lCC}_{j,i,\mathtt{id}} = (\mathtt{hpCC}_{i,\mathtt{id}})^{\mathtt{k}_{j,\mathtt{id}}} \quad \forall i \in (1, \ldots, \psi),$$
   where $\mathtt{k}_{j,\mathtt{id}}$ is the Voter Choice Return Code Generation private key of the CCR_j. Then, set $\mathbf{lCC}_{j,\mathtt{id}} = (\mathtt{lCC}_{j,1,\mathtt{id}}, \ldots, \mathtt{lCC}_{j,\psi,\mathtt{id}})$.
6. Computes the following NIZK.
   $$\pi_{\mathsf{exp},j} \leftarrow \mathsf{ProveExp}\left(((p, q, g, \mathtt{hpCC}_{1,\mathtt{id}}, \ldots, \mathtt{hpCC}_{\psi,\mathtt{id}}), (\mathtt{K}_{j,\mathtt{id}}, \mathbf{lCC}_{j,\mathtt{id}})), \mathtt{k}_{j,\mathtt{id}}\right) \quad \forall i \in (1, \ldots, \psi)$$
   to prove that $\mathbf{lCC}_{j,\mathtt{id}}$ are computed by raising $\mathtt{hpCC}_{1,\mathtt{id}}, \ldots, \mathtt{hpCC}_{\psi,\mathtt{id}}$ to the Voter Choice Return Code Generation private key $\mathtt{k}_{j,\mathtt{id}}$ corresponding to the Voter Choice Return Code Generation public key $\mathtt{K}_{j,\mathtt{id}}$.
7. Update $\mathsf{L}_{\mathtt{sentVotes},j} \leftarrow \mathsf{L}_{\mathtt{sentVotes},j} \cup (\mathtt{b_{id}}, \mathbf{pCC}_{\mathtt{id}}, \mathbf{lCC}_{j,\mathtt{id}}, \pi_{\mathsf{exp},j})$ and the logs with the list.

It outputs $(\mathbf{lCC}_{j,\mathtt{id}}, \mathsf{Logs}_{\mathsf{CCR}_j})$.

[15] $\mathsf{L}_{\mathtt{decPCC},j}$ is the list of ballots for which partial decryption have been already performed. It is updated in Algorithm 12.2.1.5

12.2.1.8 ExtractCRC($\mathtt{b_{id}}, \{\mathbf{lCC}_{j,\mathtt{id}}\}_{j=1}^{m}, \mathsf{CMtable}, \mathtt{C_{sk}}, \mathsf{bb}$)

This algorithm is executed by the server to extract the Choice Return Codes from the codes mapping table.

It takes as input the ballot $\mathtt{b_{id}}$, the shares $\mathbf{lCC}_{j,\mathtt{id}}$ of the long Choice Return Codes computed by the CCR_j, the codes mapping table $\mathsf{CMtable}$, the Codes secret key $\mathtt{C_{sk}}$, and the ballot box bb. Then it does the following:

1. Parse ballot as $\mathtt{b_{id}} = (\mathtt{vcd_{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K_{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$.
2. For each $j \in (1, \ldots, m)$, parse $\mathbf{lCC}_{j,\mathtt{id}} = (\mathtt{lCC}_{j,1,\mathtt{id}}, \ldots \mathtt{lCC}_{j,n,\mathtt{id}})$
3. Compute the long Choice Return Codes as:
   - Compute the *pre-long* Choice Return Codes $\mathtt{pC}_i^{\mathtt{id}}$ as:
   $$\mathtt{pC}_{i,\mathtt{id}} = \prod_{j=1}^{m} \mathtt{lCC}_{j,i,\mathtt{id}} \quad \forall i \in (1, \ldots, \psi).$$
   - The long Choice Return Codes are:
   $$\mathtt{lCC}_{i,\mathtt{id}} = \mathsf{H}(\mathtt{pC}_{i,\mathtt{id}} || \mathtt{vcd_{id}}) \quad \forall i \in (1, \ldots, \psi).$$
4. Derive the symmetric keys as follows:
   - Hash $\mathtt{hlCC}_{i,\mathtt{id}} = \mathsf{H}(\mathtt{lCC}_{i,\mathtt{id}} || \mathtt{C_{sk}}) \quad \forall i \in (1, \ldots, \psi)$.
   - The symmetric keys are $\mathtt{skcc}_{i,\mathtt{id}} \leftarrow \mathsf{KDF}(\mathtt{hlCC}_{i,\mathtt{id}}) \quad \forall i \in (1, \ldots, \psi)$.
5. Extract the Choice Return Codes from the codes mapping table:
   - If there exists an entry $[\mathsf{H}(\mathtt{lCC}_{i,\mathtt{id}}), \mathtt{ctCC}_{\mathtt{id},i})]$ in $\mathsf{CMtable}$, decrypt the Choice Return Code:
   $$\mathtt{CC}_{i,\mathtt{id}} \leftarrow \mathsf{Dec}^{s}(\mathtt{ctCC}_{\mathtt{id},i}; \mathtt{skcc}_{i,\mathtt{id}}).$$
   Then, set boolean flag $\mathtt{extCC} \leftarrow \top$, and let $\mathbf{cc}_{\mathtt{id}} = (\mathtt{CC}_{1,\mathtt{id}}, \ldots, \mathtt{CC}_{\psi,\mathtt{id}})$.
   - If entry $[\mathsf{H}(\mathtt{lCC}_{i,\mathtt{id}}), \mathtt{ctCC}_{\mathtt{id},i})]$ does not exist in $\mathsf{CMtable}$ for some $i \in (1, \ldots, \psi)$ unset boolean flag $\mathtt{extCC} \leftarrow \perp$.
6. Extract the list $\mathsf{L}_{\mathtt{sentVotes}}$ of ballots with extracted short Choice Return Codes from the ballot box bb and update it: $\mathsf{L}_{\mathtt{sentVotes}} \leftarrow \mathsf{L}_{\mathtt{sentVotes}} \cup (\mathtt{b_{id}}, \mathtt{extCC})$. In addition, update the ballot box with the list.

If $\mathtt{extCC} = \perp$ abort and output $(\perp, \mathsf{bb})$. Otherwise, output $(\mathbf{cc}_{\mathtt{id}}, \mathsf{bb})$.

12.2.1.9 VerifyCRC($\mathbf{v}_{\mathtt{id}}, \mathbf{cc}^*_{\mathtt{id}}, \mathbf{cc}_{\mathtt{id}}$)

This algorithm is executed by the Voter. It checks whether the Choice Return Codes $\mathbf{cc}^*_{\mathtt{id}} = (\mathtt{CC}^*_{1,\mathtt{id}}, \ldots, \mathtt{CC}^*_{\psi,\mathtt{id}})$ shown by her Voting Client are the same as the codes $\mathbf{cc}_{\mathtt{id}} = (\mathtt{CC}_{1,\mathtt{id}}, \ldots, \mathtt{CC}_{\psi,\mathtt{id}})$ printed in her Voting Card matching her choices $\mathbf{v}_{\mathtt{id}}$.

The voter outputs 1 if and only if $\mathtt{CC}^*_{i,\mathtt{id}} = \mathtt{CC}_{i,\mathtt{id}},\ \forall i \in (1, \ldots, \psi)$.

12.2.2 Protocol ConfirmVote

It is an interactive protocol run between the voter, the Voting Client , the server and the Choice Return Code control components CCRs. It consists of the following algorithms.

12.2.2.1 CreateConfirmMessage($\mathtt{BCK}_{\mathtt{id}}, \mathtt{k}_{\mathtt{id}}$)

This algorithm is executed by the Voting Client . It receives as input a Ballot Casting Key $\mathtt{BCK}_{\mathtt{id}}$, and a Verification Card private key $\mathtt{k}_{\mathtt{id}}$. It outputs the Confirmation Key $\mathtt{CK}_{\mathtt{id}} = (\mathtt{BCK}_{\mathtt{id}})^{2 \cdot \mathtt{k}_{\mathtt{id}}}$.

12.2.2.2 VerifyConfirmationServ($\mathtt{b}_{\mathtt{id}}, \mathsf{Logs}_{\mathtt{serv}}$)

This algorithm is executed by the Server. It receives as input a ballot $\mathtt{b}_{\mathtt{id}}$, and logs $\mathsf{Logs}_{\mathtt{serv}}$. Then it does the following checks:

1. Parse ballot as $\mathtt{b}_{\mathtt{id}} = (\mathtt{vcd}_{\mathtt{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K}_{\mathtt{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$.
2. Extract the list $\mathsf{L}_{\mathtt{sentVotes}}$ and the list $\mathsf{L}_{\mathtt{confirmedVotes}}$ from the logs [16]
3. If $\mathtt{vcd}_{\mathtt{id}} \notin \mathsf{L}_{\mathtt{sentVotes}}$ abort and output $\perp$.
4. If there exists an entry $(\mathtt{b}_{\mathtt{id}}, \mathtt{attempts}_{\mathtt{id}}) \in \mathsf{L}_{\mathtt{confirmedVotes}}$, and $\mathtt{attempts}_{\mathtt{id}}$ equals the maximum number of confirmation attempts[17] abort and output $\perp$.

It outputs $\top$. (If reached this point, short Choice Return Codes have been extracted for $\mathtt{b}_{\mathtt{id}}$ and the number of confirmation attempts is in range.)

12.2.2.3 VerifyConfirmationCCR$_j$($\mathtt{b}_{\mathtt{id}}, \mathsf{Logs}_{\mathtt{CCR}_j}$)

This algorithm is executed by the Choice Return Code control component CCR_j. It receives as input a ballot $\mathtt{b}_{\mathtt{id}}$, and logs $\mathsf{Logs}_{\mathtt{CCR}_j}$. Then it does the following checks:

1. Parse ballot as $\mathtt{b}_{\mathtt{id}} = (\mathtt{vcd}_{\mathtt{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K}_{\mathtt{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$.

[16] The Server updates list $\mathsf{L}_{\mathtt{sentVotes}}$ when it extracts the short Choice Codes from the codes mapping table, and list $\mathsf{L}_{\mathtt{confirmedVotes}}$ when it extracts short Vote Cast Return Codes. See Algorithm 12.2.1.8 and Algorithm 12.2.2.5.

[17] The maximum number of attempts to confirm a vote should be small, e.g. five attempts.

2. Extract the list $\mathsf{L}_{\mathtt{sentVotes},j}$ of ballots for which shares of the long Choice Codes have been already computed, and the list $\mathsf{L}_{\mathtt{confirmedVotes},j}$ of ballots for which a confirmation attempt has been requested.[18]

3. If there is no entry $\mathtt{b}_{\mathtt{id}} \notin \mathsf{L}_{\mathtt{sentVotes},j}$ abort and output $\perp$.

4. If there exists an entry $(\mathtt{b}_{\mathtt{id}}, \mathtt{attempts}_{\mathtt{id}}, \star, \star, \star) \in \mathsf{L}_{\mathtt{confirmedVotes},j}$, and $\mathtt{attempts}_{\mathtt{id}}$ equals the maximum number of confirmation attempts abort and output $\perp$.

It outputs $\top$. (If reached this point, long Choice Return Codes shares have been computed for $\mathtt{b}_{\mathtt{id}}$ and the number of confirmation attempts is in range.)

12.2.2.4 $\mathsf{CreateLVCCShare_j}(\mathtt{b}_{\mathtt{id}}, \mathtt{CK}_{\mathtt{id}}, (\mathtt{Kc}_{j,\mathtt{id}}, \mathtt{kc}_{j,\mathtt{id}}), \mathsf{Logs}_{\mathtt{CCR}_j})$

This algorithm is executed by the Choice Return Code control component CCR_j to create its share of the long Vote Cast Return Code.

It takes as input a ballot $\mathtt{b}_{\mathtt{id}}$, a Confirmation Key $\mathtt{CK}_{\mathtt{id}}$, the Voter Vote Cast Return Code Generation key pair $(\mathtt{Kc}_{j,\mathtt{id}}, \mathtt{kc}_{j,\mathtt{id}})$, and logs $\mathsf{Logs}_{\mathtt{CCR}_j}$. Then it does the following:

1. Parse ballot as $\mathtt{b}_{\mathtt{id}} = (\mathtt{vcd}_{\mathtt{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K}_{\mathtt{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$.

2. Hash the Confirmation Key and convert it to a group element: $\mathtt{hcm}_{\mathtt{id}} = \big(\mathsf{H}(\mathtt{CK}_{\mathtt{id}})\big)^2$.

3. Create the long Vote Cast Return Code share as:

$$\mathtt{lVCC}_{\mathtt{id},j} = {\mathtt{hcm}_{\mathtt{id}}}^{\mathtt{kc}_{j,\mathtt{id}}},$$

   where $\mathtt{kc}_{j,\mathtt{id}}$ is the Voter Vote Cast Return Code Generation private key.

4. Generate the following NIZK proof of correct exponentation

$$\pi_{\mathsf{expCK},j} := \mathsf{ProveExp}\left(((p, q, g, \mathtt{hcm}_{\mathtt{id}}), (\mathtt{Kc}_{j,\mathtt{id}}, \mathtt{lVCC}_{\mathtt{id},j})), \mathtt{kc}_{j,\mathtt{id}}\right).$$

5. Extract the list $\mathsf{L}_{\mathtt{confirmedVotes},j}$ of ballots processed in this algorithm and do the following:

   - Find entry $(\mathtt{b}_{\mathtt{id}}, \mathtt{attempts}_{\mathtt{id}}, \star, \star, \star)$. If the entry does not exist, create entry $(\mathtt{b}_{\mathtt{id}}, 0, \star, \star, \star)$
   - Update the entry:

$$(\mathtt{b}_{\mathtt{id}}, \mathtt{attempts}_{\mathtt{id}}, \star, \star, \star) \leftarrow (\mathtt{b}_{\mathtt{id}}, \mathtt{attempts}_{\mathtt{id}} + 1, \mathtt{CK}_{\mathtt{id}}, \mathtt{lVCC}_{\mathtt{id},j}, \pi_{\mathsf{expCK},j}).$$

     Also, update the logs with the list.

It outputs $(\mathtt{lVCC}_{\mathtt{id},j}, \mathsf{Logs}_{\mathtt{CCR}_j})$.

[18] The control component update list $\mathsf{L}_{\mathtt{sentVotes},j}$ and list $\mathsf{L}_{\mathtt{confirmedVotes},j}$ during generation of the long code shares. See Algorithm 12.2.1.7 and Algorithm 12.2.2.4.

12.2.2.5 ExtractVCC($\mathtt{b_{id}}, \{\mathtt{lVCC_{id,j}}\}_{j=1}^{m}, \mathsf{CMtable}, \mathtt{C_{sk}}, \mathsf{bb}$)

This algorithm is executed by the Server to retrieve the Vote Cast Return Code from the codes mapping table.

It takes as input a ballot $\mathtt{b_{id}}$, the long Vote Cast Return Code share $\mathtt{lVCC_{id,j}}$ of each CCR_j, the codes mapping table $\mathsf{CMtable}$, the Codes secret key $\mathtt{C_{sk}}$, and the ballot box bb. Then it does the following:

1. Parse ballot as $\mathtt{b_{id}} = (\mathtt{vcd_{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K_{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$.
2. Compute the long Vote Cast Return Code as follows:
   - Compute the *pre-long* Vote Cast Return Code $\mathtt{pVCC_{id}} = \prod_{j=1}^{m} \mathtt{lVCC_{id,j}}$.
   - Compute the long Vote Cast Return Code: $\mathtt{lVCC_{id}} = \mathsf{H}(\mathtt{pVCC_{id}} || \mathtt{vcd_{id}})$.
3. Extract the Vote Cast Return Code from the codes mapping table:
   - Compute $\mathtt{hlVCC_{id}} = \mathsf{H}(\mathtt{lVCC_{id}} || \mathtt{C_{sk}})$ and derive the symmetric encryption key $\mathtt{skvcc_{id}} = \mathsf{KDF}(\mathtt{hlVCC_{id}})$.
   - If there exists an entry $[\mathsf{H}(\mathtt{lVCC_{id}}), \mathtt{ctVCC_{id}})]$ in $\mathsf{CMtable}$, decrypt the Vote Cast Return Code:
     $$\mathtt{VCC_{id}} \leftarrow \mathsf{Dec}^{s}(\mathtt{ctVCC_{id}}; \mathtt{skvcc_{id}}).$$
     Then, set the boolean flag $\mathtt{extVCC} \leftarrow \top$.
   - If there is no entry $[\mathsf{H}(\mathtt{lVCC_{id}}), \mathtt{ctVCC_{id}})]$ in $\mathsf{CMtable}$ unset boolean flag $\mathtt{extVCC} \leftarrow \bot$.
4. Extract the list $\mathsf{L}_{\mathtt{confirmedVotes}}$ from bb and do the following:
   - Find entry $(\mathtt{b_{id}}, \mathtt{attempts_{id}}, \star)$. If the entry does not exist, create entry $(\mathtt{b_{id}}, 0, \star)$.
   - Update the entry: $(\mathtt{b_{id}}, \mathtt{attempts_{id}}, \star) \leftarrow (\mathtt{b_{id}}, \mathtt{attempts_{id}}+1, \mathtt{extVCC})$. Also update the ballot box with the list.

If $\mathtt{extVCC} = \bot$ abort and output $(\bot, \mathsf{bb})$. Otherwise, output $(\mathtt{VCC_{id}}, \mathsf{bb})$.

12.2.2.6 VerifyVCC($\mathtt{VCC^{*}_{id}}, \mathtt{VCC_{id}}$)

This algorithm is executed by the Voter. It checks whether the Vote Cast Return Code $\mathtt{VCC^{*}_{id}}$ shown by her Voting Client is the same as the code $\mathtt{VCC_{id}}$ printed in her Voting Card.

The voter outputs 1 if and only if $\mathtt{VCC^{*}_{id}} = \mathtt{VCC_{id}}$.

12.2.3 VerifyVotingPhase($\mathsf{bb}, \mathsf{Logs}_{\mathtt{PO}}, \mathsf{Logs}_{\mathtt{serv}}, \{\mathsf{Logs}_{\mathrm{CCR}_j}\}_{j=1}^{m}$)

This algorithm is run by an auditor. It verifies transcript consistency and NIZKs of the voting phase.

It takes as input the ballot box bb, the codes mapping table $\mathsf{CMtable}$, the Codes Secret key $\mathtt{C_{sk}}$, and the logs of the Print Office, the Server, and the Choice Return Codes control components. Then it proceeds as follows.

Parse ballot box and logs:

1. Extract the lists $\mathsf{L}_{\mathtt{sentVotes}}$ and $\mathsf{L}_{\mathtt{confirmedVotes}}$ from the ballot box bb.
2. For each $j \in (1, \ldots, m)$ extract from $\mathsf{Logs}_{\mathtt{CCR}_j}$ the lists $\mathsf{L}_{\mathtt{validVotes},j}$, $\mathsf{L}_{\mathtt{decPCC},j}$, $\mathsf{L}_{\mathtt{sentVotes},j}$ and $\mathsf{L}_{\mathtt{confirmedVotes},j}$.
3. Extract from $\mathsf{Logs}_{\mathtt{serv}}$ the list $\mathsf{L}_{\mathtt{decPCC}}$.
4. Extract $\mathsf{PublicKeys}_j = \left(\mathbf{pk}_{\mathrm{CCR}_j}, \{\mathtt{vcd}_{\mathtt{id}}, \mathtt{Kc}_{j,\mathtt{id}}, \mathtt{K}_{j,\mathtt{id}}, \}_{\mathtt{id} \in \mathcal{ID}}\right)$ from $\mathsf{Logs}_{\mathtt{PO}}$.
5. Extract $(\mathsf{CMtable}, \mathtt{c}_{\mathtt{sk}})$ from $\mathsf{Logs}_{\mathtt{PO}}$.

Consistent ballot box: Check that ballots in lists $\mathsf{L}_{\mathtt{sentVotes}}$ and $\{\mathsf{L}_{\mathtt{sentVotes},j}\}_{j=1}^{m}$ are the same. Likewise, check that ballots in lists $\mathsf{L}_{\mathtt{confirmedVotes}}$ and $\{\mathsf{L}_{\mathtt{confirmedVotes},j}\}_{j=1}^{m}$ are the same. If an inconsistency is found output $\perp$.

Validity of confirmed votes: Check that all ballots $\mathtt{b}_{\mathtt{id}}$ in list $\mathsf{L}_{\mathtt{confirmedVotes}}$ are also present in lists $\mathsf{L}_{\mathtt{validVotes},j}$ for $j \in (1, \ldots, m)$. If an inconsistency is found for some j, output $\perp$.

Extractable short Choice Return Codes: For each $\mathtt{b}_{\mathtt{id}} \in \mathsf{L}_{\mathtt{confirmedVotes}}$ do the following

1. Verify correct decryption of Partial Choice Return Codes:
   - Find entry $(\mathtt{b}_{\mathtt{id}}, \mathbf{pCC}_{\mathtt{id}}, \mathbf{d}_{\mathtt{id}}, \{\mathbf{d}_{j,\mathtt{id}}\}_{j=1}^{m})$ in $\mathsf{L}_{\mathtt{decPCC}}$, and entry $(\mathtt{b}_{\mathtt{id}}, \mathbf{d}^{\star}_{j,\mathtt{id}}, \pi_{\mathtt{decPCC},j})$ in $\mathsf{L}_{\mathtt{decPCC},j}$. If some entry is not found output $\perp$.
   - If $\mathbf{d}^{\star}_{j,\mathtt{id}} \neq \mathbf{d}_{j,\mathtt{id}}$ for some $j \in (1, \ldots, m)$, output $\perp$.
   - Parse $\mathtt{E2} \in \mathtt{b}_{\mathtt{id}}$ as $\mathtt{E2} = (c_{2,0}, c_{2,1}, \ldots, c_{2,\psi})$, and $\mathbf{d}_{j,\mathtt{id}} = (d_{j,1}, \ldots, d_{j,\psi})$. Then, execute:
   $$b_j \leftarrow \mathsf{VerifyExp}(((p, q, g, c_{2,0}), (\prod_{i=1}^{\psi} \mathtt{pk}_{\mathrm{CCR}_j,i}, \prod_{i=1}^{\psi} d_{j,i})), \pi_{\mathtt{decPCC},j}) \ \forall j \in (1, \ldots, m).$$
   If some $b_j = 0$ output $\perp$.
   - Parse $\mathbf{d}_{\mathtt{id}} = (d_1, \ldots, d_\psi)$ and $\mathbf{pCC}_{\mathtt{id}} = (\mathtt{pCC}_{1,\mathtt{id}}, \ldots, \mathtt{pCC}_{\psi,\mathtt{id}})$. If $d_i \neq \prod_{j=1}^{\psi} d_{j,i}$ or $\mathtt{pCC}_{i,\mathtt{id}} \neq \frac{c_{2,i}}{d_i}$ for some $i \in (1, \ldots, \psi)$ output $\perp$.
2. Verify correct generation of long Choice Return Codes:
   - Find entry $(\mathtt{b}_{\mathtt{id}}, \mathbf{pCC}^{\star}_{\mathtt{id}}, \mathbf{lCC}_{j,\mathtt{id}}, \pi_{\mathtt{exp},j}) \in \mathsf{L}_{\mathtt{sentVotes},j}$. If $\mathbf{pCC}_{\mathtt{id}} \neq \mathbf{pCC}^{\star}_{\mathtt{id}}$ for some $j \in (1, \ldots, m)$ output $\perp$.
   - Then for $\forall j \in (1, \ldots, m)$ execute:
   $$b_j \leftarrow \mathsf{VerifyExp}(((p, q, g, \mathsf{H}(\mathtt{pCC}_{1,\mathtt{id}}))^2, \ldots, \mathsf{H}(\mathtt{pCC}_{\psi,\mathtt{id}}))^2), (\mathtt{K}_{j,\mathtt{id}}, \mathbf{lCC}_{j,\mathtt{id}}), \pi^{(j)}_{\mathtt{exp},i})$$
   If some $b_j = 0$ output $\perp$.
3. Using input $(\mathtt{b}_{\mathtt{id}}, \{\mathbf{lCC}_{j,\mathtt{id}}\}_{j=1}^{m}, \mathsf{CMtable}, \mathtt{c}_{\mathtt{sk}})$ if $\mathtt{b}_{\mathtt{id}}$ is marked as extractable in $\mathsf{L}_{\mathtt{sentVotes}}$, check there is an entry in the codes mapping table; otherwise if ballot is non-extractable, check there is no entry. (See Algorithm 12.2.1.8.) If an incosistency is found, output $\perp$.

Extractable short Vote Cast Return Code: For each $\mathtt{b_{id}} \in \mathsf{L}_{\mathtt{confirmedVotes}}$ do the following

1. Verify correct generation of long Vote Cast Return Code:
   - Find entry $(\mathtt{b_{id}}, \mathtt{attempts_{id}}, \mathtt{CK_{id}}, \mathtt{lVCC}_{\mathtt{id},j}, \pi_{\mathtt{expCK},j}) \in \mathsf{L}_{\mathtt{confirmedVotes},j}$. If $\mathtt{attempts_{id}}$ or $\mathtt{CK_{id}}$ are different for any $j \neq j'$ output $\perp$.
   - For each $j \in (1, \ldots, m)$ execute:

     $$b_j \leftarrow \mathsf{VerifyExp}(((p, q, g, \mathsf{H}(\mathtt{CK_{id}}))^2), (\mathtt{Kc}_{j,\mathtt{id}}, \mathtt{lVCC}_{\mathtt{id},j}), \pi_{\mathtt{expCK},j}).$$

     If some $b_j = 0$ output $\perp$.
2. Using input $(\mathtt{b_{id}}, \{\mathtt{lVCC}_{\mathtt{id},j}\}_{j=1}^{m}, \mathsf{CMtable}, \mathtt{C_{sk}})$: if ballot $\mathtt{b_{id}}$ is marked as extractable in $\mathsf{L}_{\mathtt{confirmedVotes}}$ check there exist an entry in $\mathsf{CMtable}$; for non-extractable ballots check there is no entry. (See Algorithm 12.2.2.5.) If some inconsistency is found output $\perp$.

If and only if all verifications are successful it outputs $\top$.

12.3 Tally phase protocol

The execution flow of the protocols in the tally phase is depicted in Figure 20.

12.3.1 Protocol MixOnline

It is an interactive protocol run between the Server and all but the last Mixing control components CCMs. It consist in the following algorithms.

12.3.1.1 Cleansing(bb)

It is an algorithm run by the server. It takes as input the ballot box bb and it does the following:

1. Extract from the ballot box bb the list of confirmed ballots $\mathsf{L}_{\mathtt{confirmedVotes}}$ and list of sent ballots $\mathsf{L}_{\mathtt{sentVotes}}$.
2. Create an empty cleansed ballot box $\mathsf{bb}_{\mathsf{clean}}$.
3. For each entry of $\mathsf{L}_{\mathtt{confirmedVotes}}$ do:
   - Parse the entry as $(\mathtt{b_{id}}, \mathtt{attempts_{id}}, \mathtt{extVCC})$
   - Parse $\mathtt{b_{id}}$ as $(\mathtt{vcd_{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K_{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$.
   - Update $\mathsf{bb}_{\mathsf{clean}} \leftarrow \mathsf{bb}_{\mathsf{clean}} \cup \{\mathtt{E1}\}$.

It outputs $\mathsf{bb}_{\mathsf{clean}}, \mathsf{L}_{\mathtt{confirmedVotes}}, \mathsf{L}_{\mathtt{sentVotes}}/\mathsf{L}_{\mathtt{confirmedVotes}}$.

12.3.1.2 $\mathsf{MixDecOnline}_j\left(\mathbf{c}_{\mathsf{mix},j-1}, \mathbf{c}_{\mathsf{Dec},j-1}, \overline{\mathtt{EL}}_{\mathtt{pk},j-1}, \mathtt{EL}_{\mathtt{pk},j}, \mathtt{EL}_{\mathtt{sk},j}, \mathsf{Logs}_{\mathsf{CCM}_j}, \mathtt{crs}\right)$

This algorithm is executed by the Mixing control component CCM_j for $j \in (1, \ldots, m'-1)$ to shuffle ciphertexts and partially decrypt them.

It takes as input a list of shuffled ciphertexts $\mathbf{c}_{\mathsf{mix},j-1}$, a list of partially decrypted ciphertexts $\mathbf{c}_{\mathsf{Dec},j-1}$, the remaining Election public key $\overline{\mathtt{EL}}_{\mathtt{pk},j-1}$, control component Election key pair share $(\mathtt{EL}_{\mathtt{pk},j}, \mathtt{EL}_{\mathtt{sk},j})$ and logs $\mathsf{Logs}_{\mathsf{CCM}_j}$.

If $j = 1$, the first ciphertext list $\mathbf{c}_{\mathsf{mix},j-1}$ is empty, the second ciphertext list $\mathbf{c}_{\mathsf{Dec},j-1}$ comes from the cleansed ballot box $\mathsf{bb}_{\mathsf{clean}}$, and the remaining election public key $\overline{\mathtt{EL}}_{\mathtt{pk},j-1}$ is the election public key $\mathtt{EL}_{\mathtt{pk}}$.

Then, it does the following:

1. Shuffle the input ciphertexts: $(\mathbf{c}_{\mathsf{mix},j}, \pi_{\mathsf{mix},j}) \leftarrow \mathsf{Mix}(\overline{\mathtt{EL}}_{\mathtt{pk},j-1}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_{\mathsf{Dec},j-1})$.
2. Partially decrypt the shuffled ciphertexts:
   - Compute a list of partially decrypted ciphertexts $\mathbf{c}_{\mathsf{Dec},j} = \left(\overline{\mathbf{c}}_1, \ldots, \overline{\mathbf{c}}_N\right)$ as

     $$\overline{\mathbf{c}}_i = (\overline{c}_{i,1}, \overline{c}_{i,2}) = \left(\hat{c}_{i,1}, \hat{c}_{i,2} \cdot (\hat{c}_{i,1})^{-\mathtt{EL}_{\mathtt{sk},j}}\right) \quad \text{for } \forall(\hat{c}_{i,1}, \hat{c}_{i,2}) \in \mathbf{c}_{\mathsf{mix},j}, \text{ where } i \in (1, \ldots, N),$$

     and a list of NIZK proofs $\boldsymbol{\pi}_{\mathsf{dec},j} = \left(\pi_{\mathsf{dec},j,1}, \ldots, \pi_{\mathsf{dec},j,N}\right)$ for correct partial decryptions as

     $$\pi_{\mathsf{dec},j,i} = \mathsf{ProveDec}\left(\left(p, q, g, \mathtt{EL}_{\mathtt{pk},j}, \hat{\mathbf{c}}_i, \overline{c}_{i,2}\right), \mathtt{EL}_{\mathtt{sk},j}\right) \quad \forall i \in (1, \ldots, N).$$

   - Compute the remaining election public key for the next control component. Thus, set: $\overline{\mathtt{EL}}_{\mathtt{pk},j} = \overline{\mathtt{EL}}_{\mathtt{pk},j-1}/\mathtt{EL}_{\mathtt{pk},j}$. Note that $\mathbf{c}_{\mathsf{Dec},j}$ is encrypted under public key $\overline{\mathtt{EL}}_{\mathtt{pk},j}$.
3. Update logs:

   $$\mathsf{Logs}_{\mathsf{CCM}_j} \leftarrow \mathsf{Logs}_{\mathsf{CCM}_j} \cup \left((\mathbf{c}_{\mathsf{mix},j-1}, \mathbf{c}_{\mathsf{Dec},j-1}), (\mathbf{c}_{\mathsf{mix},j}, \mathbf{c}_{\mathsf{Dec},j}), (\overline{\mathtt{EL}}_{\mathtt{pk},j-1}, \mathtt{EL}_{\mathtt{pk},j}), (\pi_{\mathsf{mix},j}, \boldsymbol{\pi}_{\mathsf{dec},j})\right).$$

It outputs $(\mathbf{c}_{\mathsf{mix},j}, \mathbf{c}_{\mathsf{Dec},j}, \overline{\mathtt{EL}}_{\mathtt{pk},j}, \mathsf{Logs}_{\mathsf{CCM}_j})$.

12.3.2 Protocol MixOffline

It is a protocol executed in the last (offline) Mixing control component $\mathrm{CCM}_{m'}$. It consist in two algorithms. For simplicity, decoding it also executed in this node, although it can be done in a different one.

12.3.2.1 $\mathsf{MixDecOffline}\left(\mathbf{c}_{\mathsf{mix},m'-1}, \mathbf{c}_{\mathsf{Dec},m'-1}, \overline{\mathtt{EL}}_{\mathtt{pk},m'-1}, \mathtt{EB}_{\mathtt{pk}}, \mathtt{EB}_{\mathtt{sk}}, \mathsf{Logs}_{\mathsf{CCM}_{m'}}, \mathtt{crs}\right)$

This algorithm is executed by the last Mixing control component $\mathrm{CCM}_{m'}$, and it is very similar to the online mixes, the difference is that it outputs the decrypted votes.

It takes as input a list of shuffled ciphertexts $\mathbf{c}_{\mathsf{mix},m'-1}$, a list of partially decrypted ciphertexts $\mathbf{c}_{\mathsf{Dec},m'-1}$, the remaining Election public key $\overline{\mathtt{EL}}_{\mathtt{pk},m'-1}$, Electoral Board's key pair $(\mathtt{EB}_{\mathtt{pk}}, \mathtt{EB}_{\mathtt{sk}})$, and logs $\mathsf{Logs}_{\mathsf{CCM}_{m'}}$. Then it does the following:

1. Shuffle the input ciphertexts: $(\mathbf{c}_{\mathsf{mix},m'}, \pi_{\mathsf{mix},m'}) = \mathsf{Mix}(\mathtt{EB}_{\mathtt{pk}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_{\mathsf{Dec},m'-1})$.

2. Decrypt the shuffled ciphertexts:

$$m_i \leftarrow \mathsf{Dec}(\hat{\mathbf{c}}_i, \mathtt{EB}_{\mathtt{sk}}) = \hat{c}_{i,2} \cdot (\hat{c}_{i,1})^{-\mathtt{EB}_{\mathtt{sk}}} \quad \forall i \in (1, \ldots, N),$$

   - Compute a list of plaintexts $\mathbf{m} = (m_1, \ldots, m_N)$ as

$$m_i = \mathsf{Dec}(\hat{\mathbf{c}}_i, \mathtt{EB}_{\mathtt{sk}}) = \hat{c}_{i,2} \cdot (\hat{c}_{i,1})^{-\mathtt{EB}_{\mathtt{sk}}} \quad \text{for } \forall(\hat{c}_{i,1}, \hat{c}_{i,2}) \in \mathbf{c}_{\mathsf{mix},j}, \text{ where } i \in (1, \ldots, N),$$

     and a list of NIZK proofs $\boldsymbol{\pi}_{\mathsf{dec},m'} = (\pi_{\mathsf{dec},m',1}, \ldots, \pi_{\mathsf{dec},m',N})$ for correct decryption as

$$\pi_{\mathsf{dec},m',i} = \mathsf{ProveDec}\left((p, q, g, \mathtt{EB}_{\mathtt{pk}}, \hat{\mathbf{c}}_i, m_i), \mathtt{EB}_{\mathtt{sk}}\right) \quad \forall i \in (1, \ldots, N),$$

     where g is the ElGamal group generator contained in $\mathsf{crs}_{\mathtt{conf}}$.

3. Update logs:

$$\mathsf{Logs}_{\mathsf{CCM}_{m'}} = \mathsf{Logs}_{\mathsf{CCM}_{m'}} \cup \{(\mathbf{c}_{\mathsf{mix},j-1}, \mathbf{c}_{\mathsf{Dec},m'-1}), (\mathbf{c}_{\mathsf{mix},m'}, \mathbf{m}), (\overline{\mathsf{EL}}_{\mathtt{pk},m'-1}, \mathtt{EB}_{\mathtt{pk}}), (\pi_{\mathsf{mix},m'}, \boldsymbol{\pi}_{\mathsf{dec},m'})\}.$$

It outputs $(\mathbf{c}_{\mathsf{mix},m'}, \mathbf{m}, \mathsf{Logs}_{\mathsf{CCM}_{m'}})$.

12.3.2.2 DecodePlaintexts$(\mathbf{m}, \mathbf{v})$

This algorithm is executed by the last Mixing control component $\mathrm{CCM}_{m'}$ to decode the plaintext as votes.

It takes as input the decrypted plaintexts $\mathbf{m} = (m_1, \ldots, m_N)$ and the list of voting options options $\mathbf{v} = (v_1, \ldots, v_n)$. The it does the following:

1. For each m_i factorize it as $m_i = \prod_{s=1}^{\psi} v_{i_s}$. Let $\mathbf{v}_i = (v_{i_1}, \ldots, v_{i_\psi})$.

2. Set $L_{\mathsf{votes}} = \{\mathbf{v}_i\}_{i=1}^{N}$.

It outputs the list of votes L_{votes}.

12.3.3 Audit tally algorithms

12.3.3.1 VerifyOnlineTally$(\mathsf{bb}, \mathsf{Logs}_{\mathtt{PO}}, \mathsf{Logs}_{\mathtt{serv}}, \{\mathsf{Logs}_{\mathsf{CCM}_j}\}_{j=1}^{m'-1}, \mathbf{c}_{\mathsf{mix},m'-1}, \mathbf{c}_{\mathsf{Dec},m'-1},)$

It takes as input the ballot box bb and logs $\mathsf{Logs}_{\mathtt{PO}}$, $\mathsf{Logs}_{\mathtt{serv}}$, $\{\mathsf{Logs}_{\mathsf{CCM}_j}\}_{j=1}^{m'-1}$ and output $\mathbf{c}_{\mathsf{mix},m'-1}, \mathbf{c}_{\mathsf{Dec},m'-1}$. Then it does the following:

Audit flow consistency: Ensure inputs $\mathsf{bb}, \mathsf{Logs}_{\mathtt{PO}}, \mathsf{Logs}_{\mathtt{serv}}$ correspond to the inputs of Algorithm 12.2.3 (voting phase audit).

Parses logs:

- Extract $\big((\mathbf{c}_{\mathsf{mix},j-1}, \mathbf{c}_{\mathsf{Dec},j-1}), (\mathbf{c}_{\mathsf{mix},j}, \mathbf{c}_{\mathsf{Dec},j}), (\overline{\mathtt{EL}}_{\mathtt{pk},j-1}, \mathtt{EL}_{\mathtt{pk},j}), (\pi_{\mathsf{mix},j}, \boldsymbol{\pi}_{\mathsf{dec},j})\big)$ from $\{\mathsf{Logs}_{\mathtt{CCM}_j}\}_{j=1}^{m'-1}$.
- Extracts $\{\mathtt{EL}^*_{\mathtt{pk},j}\}_{j=1}^{m'-1}$, $\mathtt{EB}^*_{\mathtt{pk}}$ and $\mathtt{EL}^*_{\mathtt{pk}}$ from $\mathsf{Logs}_{\mathtt{PO}}$.

Checks information consistency:

- Checks that $\overline{\mathtt{EL}}_{\mathtt{pk},1} = \mathtt{EL}^*_{\mathtt{pk}}$.
- Checks that $\overline{\mathtt{EL}}_{\mathtt{pk},m'-1} = \mathtt{EB}^*_{\mathtt{pk}}$.
- For each $j = (1, \ldots, m'-1)$:
  - Checks that $\mathtt{EL}^*_{\mathtt{pk},j} = \mathtt{EL}_{\mathtt{pk},j}$.
  - Checks that all logged outputs of CCM_{j-1} are logged as inputs by CCM_j.
  - Checks that the logged outputs of the last online mixing component $\mathrm{CCM}_{m'-1}$ are identical to $\mathbf{c}_{\mathsf{mix},m'-1}, \mathbf{c}_{\mathsf{Dec},m'-1}$.
  - Check that the mixing key used by CCM_j is computed as $\overline{\mathtt{EL}}_{\mathtt{pk},j} = \overline{\mathtt{EL}}_{\mathtt{pk},j-1}/\mathtt{EL}_{\mathtt{pk},j}$, where $\overline{\mathtt{EL}}_{\mathtt{pk},j-1}$ is the mixing key used by CCM_{j-1}. Note, that the first component is using public election key that is identical to the one logged by the Print Office $\mathtt{EL}^*_{\mathtt{pk}}$.

Checks Cleansing: Execute $\mathsf{Cleansing}(\mathsf{bb}, \mathsf{Logs}_{\mathtt{serv}})$ (see Algorithm 12.3.1.1) and check the output $\mathsf{bb}_{\mathsf{clean}}$ is identical to $\mathbf{c}_{\mathsf{Dec},0}$.

Verifies NIZKs:

- For each $j = (1, \ldots, m'-1)$:
  - Verifies mixing proof by running $\mathsf{MixVerify}(\overline{\mathtt{EL}}_{\mathtt{pk},j-1}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_{\mathsf{Dec},j-1}, \mathbf{c}_{\mathsf{mix},j}, \pi_{\mathsf{mix},j})$. Note, that for CCM_1 $\overline{\mathtt{EL}}_{\mathtt{pk},0} = \mathtt{EL}_{\mathtt{pk}}$.
  - For each $i = (1, \ldots, N)$ runs the decryption proof verification:

$$\mathsf{VerifyDec}\left((p, q, g, \mathtt{EL}_{\mathtt{pk},j}, \hat{\mathbf{c}}_i, \overline{c}_{i,2}), \pi_{\mathsf{dec},m',i}, \mathtt{aux}\right),$$

    where $\hat{\mathbf{c}}_i, \overline{\mathbf{c}}_i = (\overline{c}_{i,1}, \overline{c}_{i,2})$ are i-th elements of the lists $\mathbf{c}_{\mathsf{mix},j}, \mathbf{c}_{\mathsf{Dec},j}$ respectively.

If and only if all the validations are successful, the process outputs $\top$. Otherwise, it outputs $\bot$.

12.3.3.2 $\mathsf{VerifyOfflineTally}(\mathbf{c}_{\mathsf{mix},m'-1}, \mathbf{c}_{\mathsf{Dec},m'-1}, \mathsf{Logs}_{\mathtt{CCM}_{m'}}, \mathbf{c}_{\mathsf{mix},m'}, \mathbf{m}, L_{\mathsf{votes}}, \mathsf{Logs}_{\mathtt{PO}})$

It takes as input an output of the the last online mixing component $\mathrm{CCM}_{m'-1}$ i.e mixed ciphertexts $\mathbf{c}_{\mathsf{mix},m'-1}$ and partially decrypted ciphertexts $\mathbf{c}_{\mathsf{Dec},m'-1}$, logs $\mathsf{Logs}_{\mathtt{CCM}_{m'}}$ and output $\mathbf{c}_{\mathsf{mix},m'}, \mathbf{m}, L_{\mathsf{votes}}$ of the offline $\mathrm{CCM}_{m'}$ mixing component and logs of the Print Office $\mathsf{Logs}_{\mathtt{PO}}$ and verifies the first tally phase as follows:

Audit flow consistency: Checks that $\mathbf{c}_{\mathsf{mix},m'-1}, \mathbf{c}_{\mathsf{Dec},m'-1}$ are identical to the output of the last online mixing component that was verified in $\mathsf{VerifyOnlineTally}$.

Parses logs:

- Parses $\mathsf{Logs}_{\mathtt{CCM}_{m'}}$ as $\{\mathbf{c}_{\mathsf{mix},j-1}, \mathbf{c}_{\mathsf{Dec},m'-1}, \mathbf{c}_{\mathsf{mix},m'}, \mathbf{m}, \overline{\mathtt{EL}}_{\mathtt{pk},m'-1}, \mathtt{EB}_{\mathtt{pk}}, \pi_{\mathsf{mix},m'}, \boldsymbol{\pi}_{\mathsf{dec},m'}\}$.
- Extracts $\mathtt{EB}^*_{\mathtt{pk}}$ and $\mathtt{EL}^*_{\mathtt{pk}}$ from $\mathsf{Logs}_{\mathtt{PO}}$.

Checks information consistency:

- Checks that $\mathtt{EB}_{\mathtt{pk}} = \mathtt{EB}^*_{\mathtt{pk}}$ and $\overline{\mathtt{EL}}_{\mathtt{pk},m'-1} = \mathtt{EB}_{\mathtt{pk}}$.
- Verifies that logged by $\mathrm{CCM}_{m'}$ inputs $\mathbf{c}_{\mathsf{mix},m'-1}, \mathbf{c}_{\mathsf{Dec},m'-1}$ are identical to the verified outputs of the last online mixing nodes.
- Verifies that logged by $\mathrm{CCM}_{m'}$ outputs $\mathbf{c}_{\mathsf{mix},m'}, \mathbf{m}, L_{\mathsf{votes}}$ are identical to the output of $\mathrm{CCM}_{m'}$.

Verifies NIZKs:

- Verifies mixing proof by running $\mathsf{MixVerify}(\mathtt{EL}_{\mathtt{pk},m'}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_{\mathsf{Dec},m'-1}, \mathbf{c}_{\mathsf{mix},m'}, \pi_{\mathsf{mix},m'})$.
- For each $i = (1, \ldots, N)$ runs the decryption proof verification:

$$\mathsf{VerifyDec}\left((p, q, g, \mathtt{EB}_{\mathtt{pk}}, \hat{\mathbf{c}}_i, m_i), \pi_{\mathsf{dec},m',i}, \mathsf{aux}\right),$$

  where $\hat{\mathbf{c}}_i, m_i$ are i-th elements of the lists $\mathbf{c}_{\mathsf{mix},m'}, \mathbf{m}$ respectively.

Verifies decoding: Runs $\mathsf{DecodePlaintexts}(\mathbf{m}, \mathbf{v})$ and checks that it matches L_{votes}.

If and only if all the validations are successful, the process outputs $\top$. Otherwise, it outputs $\bot$.

12.3.3.3 Auditors.VerifyElection($\mathsf{bb}, \mathsf{Logs}_{\mathtt{serv}}, \mathsf{Logs}_{\mathtt{PO}}, \{\mathsf{Logs}_{\mathtt{CCR}_j}\}_{j=1}^{m}, \{\mathsf{Logs}_{\mathtt{CCM}_j}\}_{j=1}^{m'}$)

It takes as input the Server's ballot box bb, server's logs $\mathsf{Logs}_{\mathtt{serv}}$,Print Office 's logs $\mathsf{Logs}_{\mathtt{PO}}$, logs of all control components denoted as $\{\mathsf{Logs}_{\mathtt{CCR}_j}\}_{j=1}^{m}, \{\mathsf{Logs}_{\mathtt{CCM}_j}\}_{j=1}^{m'}$, and does the following checks:

- Runs $\mathsf{VerifyConfigPhase}(\mathsf{Logs}_{\mathtt{PO}}, \{\mathsf{Logs}_{\mathtt{CCR}_j}\}_{j=1}^{m})$.
- Runs $\mathsf{VerifyVotingPhase}(\mathsf{bb}, \mathsf{Logs}_{\mathtt{serv}}, \{\mathsf{Logs}_{\mathtt{CCR}_j}\}_{j=1}^{m}, \mathsf{Logs}_{\mathtt{PO}})$.
- Runs $\mathsf{VerifyOnlineTally}(\mathsf{bb}, \mathsf{Logs}_{\mathtt{serv}}, \{\mathsf{Logs}_{\mathtt{CCM}_j}\}_{j=1}^{m'-1}, \mathsf{Logs}_{\mathtt{PO}})$.
- Retrieves $\mathbf{c}_{\mathsf{mix},m'-1}, \mathbf{c}_{\mathsf{Dec},m'-1}$ from the logs of the last online mixing node $\mathsf{Logs}_{\mathtt{CCM}_{m'-1}}$.
- Retrieves $\mathbf{c}_{\mathsf{mix},m'}, \mathbf{m}, L_{\mathsf{votes}}$ from the logs of the offline mixing component $\mathsf{Logs}_{\mathtt{CCM}_{m'}}$.
- Runs $\mathsf{VerifyOfflineTally}(\mathbf{c}_{\mathsf{mix},m'-1}, \mathbf{c}_{\mathsf{Dec},m'-1}, \mathsf{Logs}_{\mathtt{CCM}_{m'}}, \mathbf{c}_{\mathsf{mix},m'}, \mathbf{m}, L_{\mathsf{votes}}, \mathsf{Logs}_{\mathtt{PO}})$.

If and only if all the validations are successful, the process outputs $\top$. Otherwise, it outputs $\bot$.

Part IV

Security analysis

13 Security framework

We conduct a thorough analysis in the provable secure framework [9]. The desired properties that sVote must fulfill are defined with games. A game is a set of well-defined instructions between two entities, a challenger (i.e. the game itself) and an adversary. The game finalizes with a decision signaling whether the adversary wins or not. The game-playing framework provides an environment to analyze, in a meaningful way, cryptographic schemes: the scheme under scrutiny (in our case, sVote) exhibits certain property if the corresponding game decides affirmatively with probability close to 0. Bounding the success probability of the adversary is precisely what the security reduction does: one starts with the original game and gradually change it to reach an (ideal) game where it is easy to argue that the adversary cannot win. Every change (or hop) is reduced down to a different property that has been already established; in our case, reduced either to the properties of the building blocks that sVote is built on, or to an already established property of sVote. Ultimately, everything is reduced down to the building blocks.

Random oracle model. In the analysis, hash functions are modeled as a (programmable) random oracle. Although not explicitly stated in the game definitions, all the games control this oracle.

13.1 Properties analyzed

The analysis of sVote presented here assesses the three requirements established by the Swiss Chancellery [16] to use an electronic voting system by a vast majority of the electorate. In a nutshell, these requirements are individual verifiability (cast-as-intended and recorded-as-cast), universal verifiability (counted-as-recorded) and voting secrecy (confidentiality of voters' intent and non-disclosure of early provisional results).

In Section 14 we prove two preliminary lemmas that will be useful for individual verifiability. Section 15 addresses individual verifiability and Section 16 universal verifiability. We finalize with privacy in Section 17. The proofs of all the claims are presented in Section 18.

14 Preliminary results

14.1 Correct setup

We show that the configuration phase of sVote is correct and leaks no sensitive information. More concretely, protocol SetupVoting generates a correct Codes mapping table, independently of the actions of corrupted Choice Return Code control components. Also, the transcript of the protocol reveals nothing about the private keys $\mathbf{k}_h, \mathbf{kc}_h$ of a non-compromised (honest) Choice Return Code control component CCR_h.

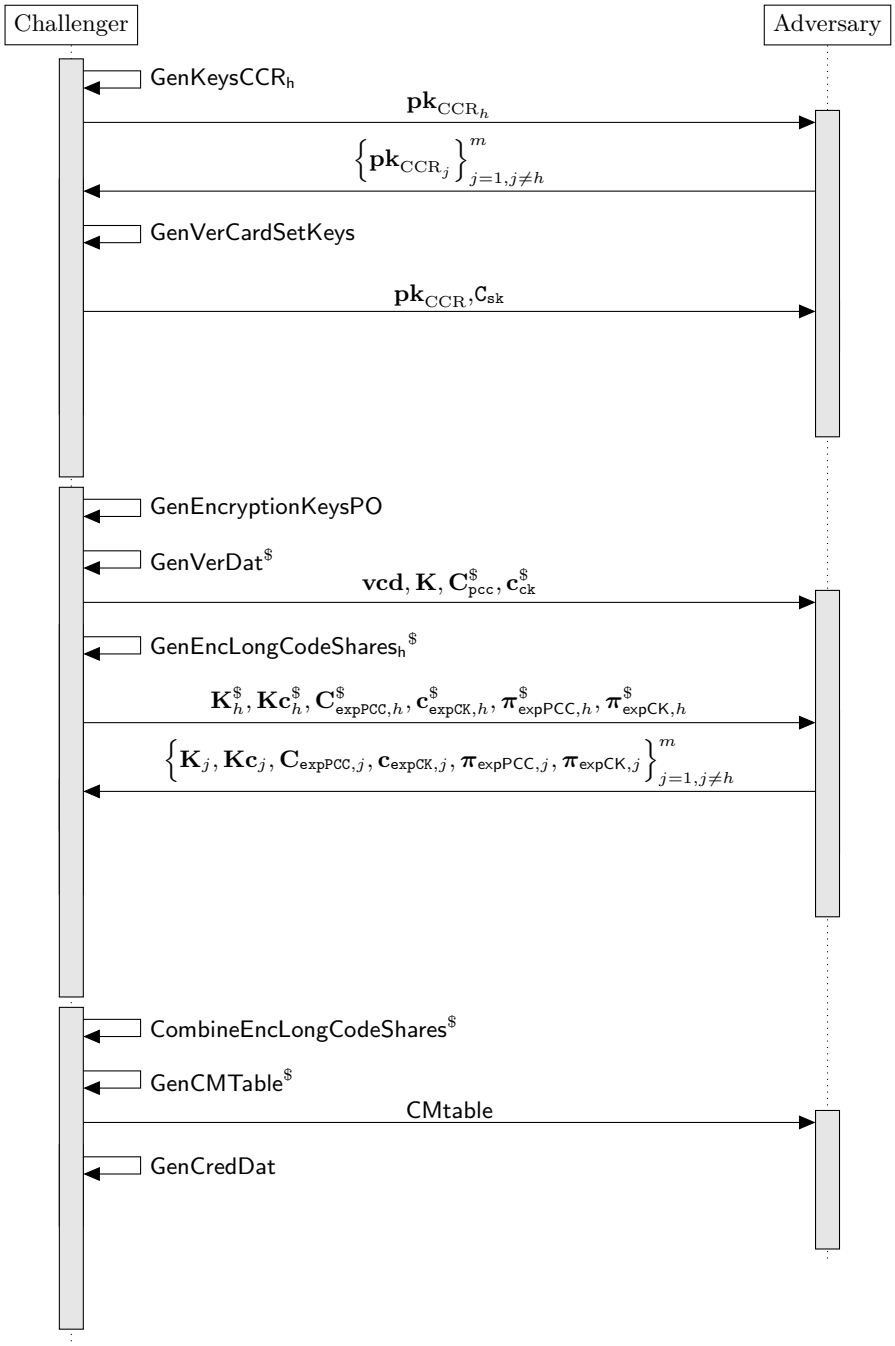

Figure 21: Protocol IdealSetupVoting used in the simulated game sMTC of Figure 23. It is executed between the Challenger (the game), controlling the Print Office and the honest CCR_h, and the adversary $\mathcal{A}$. Algorithm GenEncLongCodeShares$_h{}^{\$}$ outputs random ciphertexts $\mathbf{C}^{\$}_{\mathtt{expPCC},h}, \mathbf{c}^{\$}_{\mathtt{expCK},h}$ and random public keys $\mathbf{K}^{\$}_h, \mathbf{Kc}^{\$}_h$. Algorithm CombineEncLongCodeShares$^{\$}$ extracts CCR_j's private keys $\mathbf{k}_j$ and $\mathbf{kc}_j$ for all $j \neq h$ using the extractor of the exponentiation proof system Exp. Finally, the algorithm GenCMTable$^{\$}$ uses the extracted keys and the honest CCR_h's private keys $\mathbf{k}_h, \mathbf{kc}_h$ to construct the (correct) Codes mapping table CMtable.

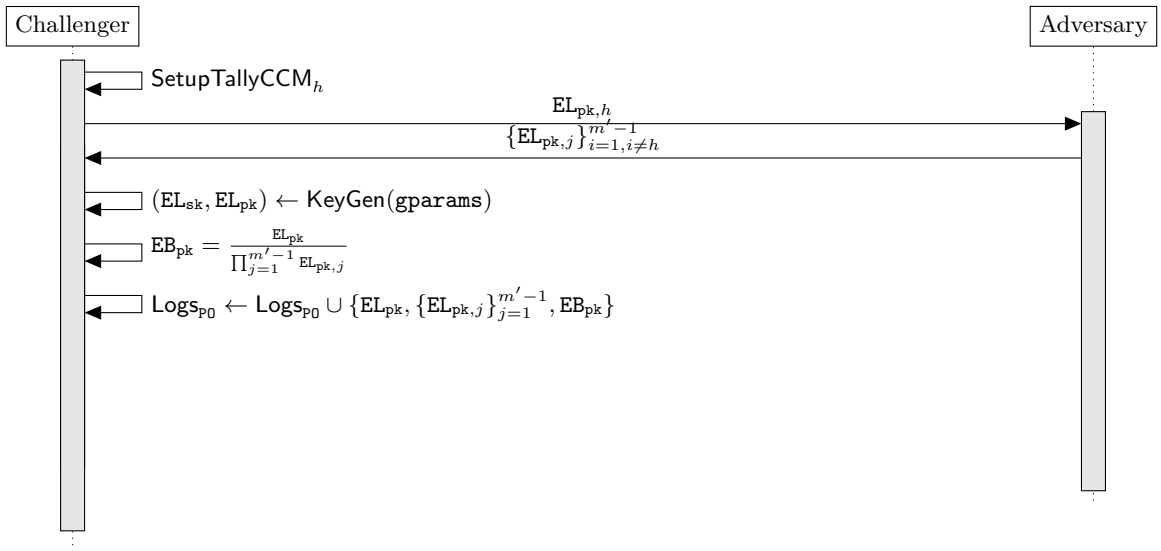

Figure 22: Protocol IdealSetupTally used in the simulated games. It is executed between the Challenger (the game), controlling the Print Office and the honest CCM$_h$, and the adversary $\mathcal{A}$. In this protocol, the Challenger gets control over the election private key $\mathtt{EL}_{\mathtt{sk}}$ by manipulating an honest CCM$_h$ private key share. Note that during the tally phase, the Challenger will have to run an extractor $\mathcal{E}_{\mathcal{A}}$ to extract witnesses from π_{dec} NIZKs in order to adjust $\mathtt{EB}_{\mathtt{sk}}$.

Modeling an ideal setup. Adversaries against the *mapping table correctness* property attempt to force the Print Office to generate the Codes mapping table for which Choice Return Codes or Vote Cast Return Codes are not extractable for a correct protocol execution, even though the auditors execute their verification algorithm VerifyConfigPhase successfully. In Figure 23 we define two games, rMTC and sMTC. The former corresponds to the real setup and the latter to an idealized setup, outlined in Figure 21. The real setup is correct and private if no adversary $\mathcal{A}$ can distinguish it from the ideal setup. The advantage of $\mathcal{A}$ is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,mtc}} = |Pr\left[1 \leftarrow \mathsf{rMTC}_{\mathcal{A}}(\mathtt{crs})\right] - Pr\left[1 \leftarrow \mathsf{sMTC}_{\mathcal{A}}(\mathtt{crs})\right]| \approx 0,$$

$\mathsf{rMTC}^{\mathsf{sVote}}(\mathtt{crs})$:

1. Execute $\mathsf{SetupVoting}_{\mathcal{A}}$
2. $b \leftarrow \mathcal{A}$ // $\mathcal{A}$ outputs a decision
3. If $0 \leftarrow$ VerifyConfigPhase output $\perp$ (abort)
4. Else output b

$\mathsf{sMTC}\mathcal{A}^{\mathsf{sVote}}(\mathtt{crs})$:

1. Execute IdealSetupVoting
2. $b \leftarrow \mathcal{A}$ // Adversary outputs a decision
3. If $0 \leftarrow$ VerifyConfigPhase output $\perp$ (abort)
4. Else output b

Figure 23: **(left)** real rMTC game **(right)** simulated sMTC game. Protocol IdealSetupVoting is outlined in Figure 21.

Lemma 1 *In the random oracle model*

$$Adv_{\mathcal{A}}^{\mathsf{sVote,mtc}} \leq Adv_{\mathcal{B}_1}^{\mathsf{Exp,NIZK}} + Adv_{\mathcal{B}_2}^{\mathsf{kdf}} + 2 \cdot Adv_{\mathcal{B}_3}^{F,\mathsf{PRF}} + Adv_{\mathcal{B}_4}^{\mathsf{ElGamal,INDCPA}} + 2 \cdot Adv_{\mathcal{B}_5}^{\mathbb{Q}_p,\mathsf{SGSP}} + \epsilon_{ext},$$

where $\mathcal{B}_1$ is an adversary against the zero-knowledge property of the exponentiation proof system Exp*, $\mathcal{B}_2$ an adversary against the key-derivation function, $\mathcal{B}_3$ an adversary against the weak pseudorandom property of the exponentiation function in the group of quadratic residues $\mathbb{Q}_p$, $\mathcal{B}_4$ an adversary against the semantic security of ElGamal encryption scheme, $\mathcal{B}_5$ and adversary against the* SGSP *problem [25], and ϵ_{ext} is the extraction error of the exponentiation proof system.*

Its proof is in Section 18.1.

14.2 Vote compliance

We show that sVote enjoys a special notion of vote correctness[19] that we call *vote compliance*. In a nutshell, if a ballot submitted by a Voting Client is marked as sent and extractable in the ballot box, and the auditors check that there exists an entry in the Codes mapping table using the ballot and the transcript of the Choice Return Code control components, then the ballot must contain ψ voting options with high probability.

We start defining what is understood by a valid ballot, a compliant ballot and a compliant *registered* ballot. The latter notion is used in the security reductions; although the definitions may seem artificial their purpose is to work with them.

Definition 16 (Valid ballot) *Let $\mathbb{G}_q$ an (ElGamal) cyclic group with generator g, $\mathtt{EL}_{\mathtt{pk}} \in \mathbb{G}_q$ the Election public key, $\mathbf{pk}_{\mathrm{CCR}} = (\mathtt{pk}_{\mathrm{CCR},1}, \ldots, \mathtt{pk}_{\mathrm{CCR},\psi})$ the global public keys of the control components, and let $(\mathtt{k}_{\mathtt{id}}, \mathtt{K}_{\mathtt{id}}) \in \mathbb{Z}_q^* \times \mathbb{G}_q$ a Verification Card key pair. A submitted ballot*

$$\begin{aligned} \mathtt{b}_{\mathtt{id}} &= (\mathtt{vcd}_{\mathtt{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K}_{\mathtt{id}}, \boldsymbol{\pi}_{\mathsf{ballot}}) \\ \mathtt{E2} &= (c_{2,0}, c_{2,1}, \ldots, c_{2,\psi}) \\ \boldsymbol{\pi}_{\mathsf{ballot}} &= (\pi_{\mathsf{sch}}, \pi_{\mathsf{Exp}}, \pi_{\mathsf{EqEnc}}) \end{aligned}$$

is valid *if the NIZKs $\boldsymbol{\pi}_{\mathsf{ballot}}$ verify successfully. That is, if:*

$1 \leftarrow \mathsf{VerifySch}((p, q, g, \mathtt{E1}), \pi_{\mathsf{sch}}, \mathtt{aux})$, *where* $\mathtt{aux}$ *includes* $\mathtt{vcd}_{\mathtt{id}}$.

$1 \leftarrow \mathsf{VerifyExp}(((p, q, g, \mathtt{E1}), (\mathtt{K}_{\mathtt{id}}, \widetilde{\mathtt{E1}})), \pi_{\mathsf{Exp}})$.

$1 \leftarrow \mathsf{VerifyEqEnc}((p, q, g, \mathtt{EL}_{\mathtt{pk}}, \prod_{i=1}^{\psi} \mathtt{pk}_{\mathrm{CCR},i}, \widetilde{\mathtt{E1}}, \widetilde{\mathtt{E2}}), \pi_{\mathsf{EqEnc}})$, *where* $\widetilde{\mathtt{E2}} = \left(c_{2,0}, \prod_{i=1}^{\psi} c_{2,i}\right)$.

[19] Vote correctness asks for ballots containing *valid combinations* of the voting options as defined by the election rules. Vote correctness is out of the scope of this analysis.

Definition 17 (Compliant ballot) *Let* $\mathbb{G}_q, \mathtt{EL_{pk}}, \mathbf{pk}_{\mathrm{CCR}}, \mathtt{k_{id}}$ *as in Definition 16. A submitted ballot* $\mathtt{b_{id}} = (\mathtt{vcd_{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K_{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$ *with* $\mathtt{E1} = \mathsf{Enc}(\nu, \mathtt{EL_{pk}})$, $\mathtt{E2} = \mathsf{MultiEnc}((w_1, \ldots, w_\psi), \mathbf{pk}_{\mathrm{CCR}})$, *and* $\widetilde{\mathtt{E1}} = \mathsf{Enc}(w, \mathtt{EL_{pk}})$ *is* compliant *if it holds:*

(i) $w = \nu^{\mathtt{k_{id}}}$ *in* $\mathbb{G}_q$.

(ii) $w = \prod_{i=1}^{\psi} w_i$ *in* $\mathbb{G}_q$.

Definition 18 (Compliant registered ballot) *Let* $\mathbb{G}_q, \mathtt{EL_{pk}}, \mathbf{pk}_{\mathrm{CCR}}, \mathtt{k_{id}}$, *a submitted ballot* $\mathtt{b_{id}}$, *and group elements* ν, w, $w_1, \ldots, w_\psi$ *as in Definition 17. The ballot is* compliant *and* registered *if it holds:*

(i) There exists an entry $(\mathtt{b_{id}}, \mathbf{pCC}_{\mathtt{id}}, \mathbf{lCC}_{j,\mathtt{id}}, \pi_{\mathsf{exp},j})$, *where* $\mathbf{pCC}_{\mathtt{id}} = (\mathtt{pCC}_{1,\mathtt{id}}, \ldots, \mathtt{pCC}_{\psi,\mathtt{id}})$ *in the list of sent votes* $\mathsf{L}_{\mathtt{sentVotes},j}$ *of the Choice Return Code control component* CCR_j *for all* $j \leq m$.

(ii) $\nu = \prod_{i=1}^{\psi} w_i$, *and* $\mathtt{pCC}_{i,\mathtt{id}} = w_i^{\mathtt{k_{id}}}$ *where* w_i *is a voting option.*

Modeling vote compliance. Adversaries against vote compliance attempt to force registration of non-compliant ballots in an honest Choice Return Code control component, even though the auditors and the honest control component execute their verification algorithms successfully. We analyze sVote up to termination of protocol SendVote. This is enough since it is at this point when a ballot is marked as *sent* by the Server and the honest control component. In Figure 24 we derive a game capturing adversaries against vote compliance in sVote. The game is stated with an ideal setup, recall that in Section 14.1 we have shown that the real setup SetupVoting from Section 12.1.1, and the ideal one IdealSetupVoting from Figure 23, are computationally indistinguishable. The advantage of $\mathcal{A}$ is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,vc}} = Pr\left[1 \leftarrow \mathsf{vc}_{\mathcal{A}}(\mathtt{crs})\right].$$

In the game, event $\mathsf{bad_{vc}}$ is defined as:

$$\mathsf{bad_{vc}} = \begin{cases} \text{Ballot } \mathtt{b_{id}} \text{ is not a compliant registered ballot (see Definition 18)} \\ \mathtt{b_{id}} \in \mathsf{L_{sentVotes}} \subset \mathsf{bb} \text{ is marked as extractable for Choice Return Codes} \\ 1 \leftarrow \mathsf{VerifyVotingPhase}(\mathsf{bb}, \mathsf{CMtable}, \mathtt{c_{sk}}, \mathsf{Logs_{P0}}, \mathsf{Logs_{serv}}, \{\mathsf{Logs_{CCR}}_j\}_{j=1}^{m}) \end{cases}$$

Lemma 2 *In the random oracle model*

$$Adv_{\mathcal{A}}^{\mathsf{sVote,vc}} \leq 2 \cdot Adv_{\mathcal{B}_1}^{\mathsf{Exp,sSOUND}} + Adv_{\mathcal{B}_2}^{\mathsf{EqEnc,sSOUND}},$$

where $\mathcal{B}_1$, $\mathcal{B}_2$ *are adversaries against the simulation soundness property of the exponentiation and plaintext equality proof systems* Exp, EqEnc, *respectively.*

Its proof is in Section 18.2.

$\underline{\mathsf{vc}_{\mathcal{A}}(\mathtt{crs})}$

1. $\mathtt{id}, h \leftarrow \mathcal{A}$ // $\mathcal{A}$ specifies honest CCR_h and a target voter $\mathtt{id}$
2. $\mathsf{dat} \leftarrow \mathsf{IdealSetupVoting}$
3. Extract $(\mathsf{CMtable}, \mathtt{C}_{\mathtt{sk}}, \mathbf{vcd}, \mathbf{CC}, \mathbf{k}, \mathbf{K}, \mathbf{svk}, \mathbf{pk}_{\mathrm{CCR}}, (\mathbf{k}_j, \mathbf{K}_j)_{j \leq m}) \leftarrow \mathsf{dat}$
4. $(\mathtt{EL}_{\mathtt{pk}}, \mathtt{EL}_{\mathtt{sk}}) \leftarrow \mathsf{IdealSetupTally}$
5. Extract $\mathsf{Logs}_{\mathsf{PO}}$ from Print Office.
6. $b \leftarrow \mathsf{VerifyConfigPhase}(\mathsf{Logs}_{\mathsf{PO}}, \mathtt{crs})$. If $b = \perp$ output 0. // Perform configuration phase audit
7. $\mathsf{bb} \leftarrow \emptyset$ // Initialize an empty ballot box
8. Give the following data to $\mathcal{A}$:

$$\mathsf{advDat} = (\mathbf{v}_{\mathtt{id}}, \mathbf{vcd}, \mathbf{svk}, (\mathbf{k}, \mathbf{K}), \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, \mathsf{bb}, \mathsf{CMtable}, \mathtt{C}_{\mathtt{sk}}, ((\mathbf{k}_j, \mathbf{K}_j))_{j \neq h}, \mathbf{K}_h)$$

9. $((\mathsf{b}_{\mathtt{id}}, \mathsf{bb}), (\mathbf{pCC}_{\mathtt{id}}, (\mathbf{d}_{j,\mathtt{id}}, \pi_{\mathsf{decPCC},j}), (\mathbf{lCC}_{j,\mathtt{id}}, \pi_{\mathsf{exp},j})_{j \neq h})) \leftarrow \mathsf{SendVote}_{\mathcal{A}}$ // $\mathcal{A}$ submits ballot, updated ballot box, partial Choice Return Codes together with decryption shares, and long Choice Return Code shares for each CCR_j with $j \neq h$
10. If $\mathsf{bad}_{\mathsf{vc}}$ output 1

Figure 24: Game modeling attacks against compliance of registered ballots. Protocols IdealSetupVoting, SetupTally, and SendVote are executed in interaction with the adversary $\mathcal{A}$, who keeps state across calls. Algorithms corresponding to honest CCR_h are controlled by the game. Public parameters $\mathtt{crs}$ are implicit.

15 Individual verifiability

In this section we analyze the individual verifiability property of sVote. The security against the attack vectors modeled here is "up to guessing the return codes". Thus, our analysis shows that adversaries can not do better than guessing either the (short) Choice Return codes, the Vote Cast code, or the Ballot Casting key - all adversarial bounds account for the spaces of these short codes.

15.1 Sent as intended

Adversaries against *sent as intended* attempt to force registration, in an honest control component CCR_h, of a ballot with at least one voting option (out of ψ) different to the voting option selected by an honest voter. The voter, honest control components, and auditors do not detect the modification although they execute their verification algorithms successfully.

As for the case of vote compliance, we analyze sVote up to termination of protocol SendVote, since it is at this point when a ballot is marked as *sent* by the Server and the honest control component, and the voter executes her verification algorithm. In Figure 25 we present a game capturing adversaries against

the sent as intended property of sVote. It is similar to game vc, however this time the adversary must output a ballot registered as sent by an honest control component not containing the voter's intent and the set of valid Choice Return Codes corresponding to the selections of the voter. The advantage of $\mathcal{A}$ is defined as:

$$\mathsf{Adv}^{\mathsf{sVote,sai}}_{\mathcal{A}} = Pr\left[1 \leftarrow \mathsf{sai}_{\mathcal{A}}(\mathtt{crs})\right].$$

In the game, event $\mathsf{bad}_{\mathsf{sai}}$ is defined as:

$$\mathsf{bad}_{\mathsf{sai}} = \begin{cases} \mathsf{Dec}(\mathtt{E1}, \mathtt{EL}_{\mathtt{sk}}) \neq \prod_{i=1}^{\psi} v_{j_i} \ \text{// ballot } \mathtt{b}_{\mathtt{id}} \text{ does not contain voter's selections} \\ \mathtt{b}_{\mathtt{id}} \in \mathsf{L}_{\mathtt{sentVotes}} \subset \mathsf{bb} \text{ is marked as extractable for Choice Return Codes} \\ 1 \leftarrow \mathsf{VerifyVotingPhase}(\mathsf{bb}, \mathsf{CMtable}, \mathtt{C}_{\mathtt{sk}}, \mathsf{Logs}_{\mathtt{PO}}, \mathsf{Logs}_{\mathtt{serv}}, \{\mathsf{Logs}_{\mathtt{CCR}_j}\}_{j=1}^{m}) \\ 1 \leftarrow \mathsf{VerifyCRC}(\mathbf{v}_{\mathtt{id}}, \mathbf{cc}^{*}_{\mathtt{id}}, \mathbf{cc}_{\mathtt{id}}) \end{cases}$$

$\underline{\mathsf{sai}_{\mathcal{A}}(\mathtt{crs})}$

1. $\mathtt{id}, h \leftarrow \mathcal{A}$ // $\mathcal{A}$ specifies honest CCR_h and a target voter $\mathtt{id}$
2. $\mathsf{dat} \leftarrow \mathsf{SetupVoting}(\mathtt{crs})$
3. Extract $(\mathsf{CMtable}, \mathtt{C}_{\mathtt{sk}}, \mathbf{vcd}, \mathbf{CC}, \mathbf{k}, \mathbf{K}, \mathbf{svk}, \mathbf{pk}_{\mathrm{CCR}}, (\mathbf{k}_j, \mathbf{K}_j)_{j \leq m}) \leftarrow \mathsf{dat}$
4. $(\mathtt{EL}_{\mathtt{pk}}, \mathtt{EL}_{\mathtt{sk}}) \leftarrow \mathsf{SetupTally}(\mathtt{crs})$
5. $\mathsf{bb} \leftarrow \emptyset$ // Initialize an empty ballot box
6. Choose at random $\mathbf{v}_{\mathtt{id}} = (v_{j_1}, \ldots, v_{j_\psi})$. // Voter's selections
7. Set $\mathbf{CC}^{*} \leftarrow \mathbf{CC} \backslash \mathbf{cc}_{\mathtt{id}}$ // Do not give Choice Return Codes for Voter $\mathtt{id}$ to $\mathcal{A}$
8. Give the following data to $\mathcal{A}$:

$$\mathsf{advDat} = \quad (\mathbf{v}_{\mathtt{id}}, \mathbf{vcd}, \mathbf{svk}, (\mathbf{k}, \mathbf{K}), \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, \mathsf{bb}, \mathsf{CMtable}, \mathtt{C}_{\mathtt{sk}}, ((\mathbf{k}_j, \mathbf{K}_j))_{j \neq h}, \mathbf{K}_h, \mathbf{CC}^{*})$$

9. $(\mathbf{cc}^{*}_{\mathtt{id}}, (\mathtt{b}_{\mathtt{id}}, \mathsf{bb}), (\mathbf{pCC}_{\mathtt{id}}, (\mathbf{d}_{j,\mathtt{id}}, \pi_{\mathsf{decPCC},j}), (\mathbf{lCC}_{j,\mathtt{id}}, \pi_{\mathsf{exp},j})_{j \neq h})) \leftarrow \mathsf{SendVote}$ // $\mathcal{A}$ submits ballot, updated ballot box, partial Choice Return Codes together with decryption shares, long Choice Return Code shares for each CCR_j with $j \neq h$, and short Choice Return Code shares for voter $\mathtt{id}$
10. If $\mathsf{bad}_{\mathsf{sai}}$ output 1

Figure 25: Game modeling attacks against sent as intended (sai) property of sVote. Protocols SetupVoting, SetupTally, and SendVote are executed in interaction with the adversary $\mathcal{A}$, who keeps state across calls. Algorithms corresponding to honest CCR_h are controlled by the game. Public parameters $\mathtt{crs}$ are implicit.

Theorem 1 *In the random oracle model*

$$Adv^{\mathsf{sVote,sai}}_{\mathcal{A}} \leq Adv^{\mathsf{sVote,mtc}}_{\mathcal{B}_1} + Adv^{\mathsf{sVote,vc}}_{\mathcal{B}_2} + Adv^{\mathsf{Exp,NIZK}}_{\mathcal{B}_3} + Adv^{\mathsf{SEnc,INDCPA}}_{\mathcal{B}_4} + \frac{1}{|\mathcal{C}_{cc}|},$$

where $\mathcal{B}_1$, $\mathcal{B}_2$ are adversaries against the Codes mapping table correctness and vote compliance properties of sVote *(respectively), $\mathcal{B}_3$ an adversary against the zero-knowledge property of the exponentiation proof*

system, $\mathcal{B}_4$ an adversary against the semantic security of the symmetric encryption scheme, and $\mathcal{C}_{cc}$ is the space of short Choice Return Code values.

Its proof is in Section 18.3.

15.2 Recorded as confirmed

We define the property *recorded as confirmed* of sVote as the inability to either reject confirmed votes or inject non-confirmed votes in the ballot box.

15.2.1 Vote rejection

In this type of attacks, an honest voter sends and confirms her vote. The adversary $\mathcal{A}$ attempts to avoid registration of the confirmed vote in an honest control component CCR_h, resulting in a vote deleted from the ballot box. The voter and auditors do not detect the modification although they follow their respective verification algorithms. In Figure 26 we define game rac-rej, capturing adversaries against vote rejection. The advantage of $\mathcal{A}$ is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,rac\text{-}rej}} = Pr[1 \leftarrow \mathsf{rac\text{-}rej}(\mathtt{crs})].$$

In the game, event $\mathsf{bad}_{\mathsf{rej}}$ is defined as:

$$\mathsf{bad}_{\mathsf{rej}} = \begin{cases} \text{Ballot } \mathtt{b}_{\mathtt{id}} \notin \mathsf{L}_{\mathtt{confirmedVotes}} \subset \mathsf{bb} \text{ or it is marked as non-extractable for VCC} \\ 1 \leftarrow \mathsf{VerifyVCC}(\mathtt{VCC}^{*}_{\mathtt{id}}, \mathtt{VCC}_{\mathtt{id}}) \\ 1 \leftarrow \mathsf{VerifyVotingPhase}(\mathsf{bb}, \mathsf{CMtable}, \mathtt{C}_{\mathtt{sk}}, \mathsf{Logs}_{\mathtt{PO}}, \mathsf{Logs}_{\mathtt{serv}}, \{\mathsf{Logs}_{\mathtt{CCR}_j}\}_{j=1}^{m}) \end{cases}$$

Theorem 2 *In the random oracle model,*

$$Adv_{\mathcal{A}}^{\mathsf{sVote,rac\text{-}rej}} \leq Adv_{\mathcal{B}_1}^{\mathsf{sVote,mtc}} + Adv_{\mathcal{B}_2}^{\mathsf{Exp,NIZK}} + Adv_{\mathcal{B}_3}^{\mathsf{Exp,sSOUND}} + Adv_{\mathcal{B}_4}^{\mathsf{SEnc,INDCPA}} + \frac{1}{|\mathcal{C}_{vcc}|},$$

where $\mathcal{B}_1$ is an adversary against the Codes mapping table correctness property of sVote*, $\mathcal{B}_2, \mathcal{B}_3$ are adversaries against the zero-knowledge and simulation soundness properties of the exponentiation proof system* Exp*, respectively, $\mathcal{B}_4$ an adversary against the semantic security of the symmetric encryption scheme, and $\mathcal{C}_{vcc}$ is the space of short Vote Cast Return Code values.*

Its proof is in Section 18.4.

15.2.2 Vote injection

In this type of attacks, an honest voter sends but does not confirm her vote. The adversary attempts to force registration of the non-confirmed vote resulting in inserting a vote in the ballot box. The auditors

$\underline{\mathsf{rac\text{-}rej}_{\mathcal{A}}(\mathtt{crs})}$

1. $\mathtt{id}, h \leftarrow \mathcal{A}$ // $\mathcal{A}$ specifies honest CCR_h and a target voter $\mathtt{id}$
2. $\mathsf{dat} \leftarrow \mathsf{SetupVoting}$
3. Extract $(\mathsf{CMtable}, \mathtt{C_{sk}}, \mathbf{vcd}, \mathbf{vcc}, \mathbf{k}, \mathbf{K}, \mathbf{svk}, \mathbf{bck}, \mathbf{pk}_{\mathrm{CCR}}, (\mathbf{kc}_j, \mathbf{Kc}_j)_{j \leq m}) \leftarrow \mathsf{dat}$
4. $(\mathtt{EL_{pk}}, \mathtt{EL_{sk}}) \leftarrow \mathsf{SetupTally}(\mathtt{crs})$
5. $\mathsf{bb} \leftarrow \emptyset$ // Initialize an empty ballot box
6. Choose at random $\mathbf{v}_{\mathtt{id}} = (v_{j_1}, \ldots, v_{j_\psi})$. // Voter's selections
7. Set $\mathbf{bck}^* \leftarrow \mathbf{bck} \backslash \mathtt{BCK_{id}}$, and $\mathbf{vcc}^* \leftarrow \mathbf{vcc} \backslash \mathtt{VCC_{id}}$ // Do not give BCK (yet) and VCC for Voter $\mathtt{id}$ to $\mathcal{A}$.
8. Give the following data to $\mathcal{A}$:

$$\mathsf{advDat} = (\mathbf{v}_{\mathtt{id}}, \mathbf{vcd}, \mathbf{svk}, (\mathbf{k}, \mathbf{K}), \mathtt{EL_{pk}}, \mathbf{pk}_{\mathrm{CCR}}, \mathsf{bb}, \mathsf{CMtable}, \mathtt{C_{sk}}, (\mathbf{kc}_j, \mathbf{Kc}_j)_{j \neq h}), \mathbf{Kc}_h, \mathbf{bck}^*, \mathbf{vcc}^*)$$

9. $(\mathbf{cc}^*_{\mathtt{id}}, \mathtt{b_{id}}) \leftarrow \mathsf{SendVote}$
10. If voter accepts $\mathbf{cc}^*_{\mathtt{id}}$ give $\mathtt{BCK_{id}}$ to $\mathcal{A}$.
11. $(\mathtt{CK_{id}}, \mathtt{VCC}^*_{\mathtt{id}}, (\mathtt{lVCC}_{\mathtt{id},j}, \pi_{\mathsf{expCK},j})_{j \neq h}) \leftarrow \mathsf{ConfirmVote}$ // $\mathcal{A}$ submits Confirmation Key, Vote Cast Return Code and long Vote Cast Return Code shares for each CCR_j with $j \neq h$.
13. If $\mathsf{bad_{rej}}$ output 1

Figure 26: Game modeling attacks against recorded as confirmed (rejection) property of sVote. Protocols SetupVoting, SetupTally, SendVote and ConfirmVote are executed in interaction with the adversary $\mathcal{A}$, who keeps state across calls. Algorithms corresponding to honest CCR_h are controlled by the game. Public parameters $\mathtt{crs}$ are implicit.

do not detect the modification although they follow their verification algorithms. In Figure 27 we define game rac-inj capturing adversaries against vote injection. The advantage of $\mathcal{A}$ is defined as:

$$\mathsf{Adv}^{\mathsf{sVote,rac\text{-}inj}}_{\mathcal{A}} = Pr[1 \leftarrow \mathsf{rac\text{-}inj}(\mathtt{crs})],$$

In the game, event $\mathsf{bad_{inj}}$ is defined as:

$$\mathsf{bad_{inj}} = \begin{cases} \text{Ballot } \mathtt{b_{id}} \in \mathsf{L_{confirmedVotes}} \subset \mathsf{bb} \text{ is marked as extractable for VCC} \\ 1 \leftarrow \mathsf{VerifyVotingPhase}(\mathsf{bb}, \mathsf{CMtable}, \mathtt{C_{sk}}, \mathsf{Logs_{PO}}, \mathsf{Logs_{serv}}, \{\mathsf{Logs_{CCR}}_j\}_{j=1}^{m}) \end{cases}$$

Theorem 3 *In the random oracle model*

$$Adv^{\mathsf{sVote,rac\text{-}inj}}_{\mathcal{A}} \leq Adv^{\mathsf{sVote,mtc}}_{\mathcal{B}_1} + Adv^{\mathsf{Exp,NIZK}}_{\mathcal{B}_2} + \frac{1}{|\mathcal{C}_{bck}|} + \frac{1}{q},$$

where $\mathcal{B}_1$ is an adversary against the Codes mapping table property of sVote*, $\mathcal{B}_2$ an adversary against simulation sound property of the Exponentiation proof system* Exp*, and $\mathcal{C}_{bck}$ is the space of Ballot Casting Keys values.*

Its proof is in Section 18.5.

$\underline{\mathsf{rac\text{-}inj}_{\mathcal{A}}(\mathtt{crs})}$

1. $\mathtt{id}, h \leftarrow \mathcal{A}$ // $\mathcal{A}$ specifies honest CCR_h and a target voter $\mathtt{id}$
2. $\mathsf{dat} \leftarrow \mathsf{SetupVoting}_{\mathcal{A}}$
3. Extract $(\mathsf{CMtable}, \mathtt{C}_{\mathtt{sk}}, \mathbf{vcd}, \mathbf{k}, \mathbf{K}, \mathbf{svk}, \mathbf{bck}, \mathbf{pk}_{\mathrm{CCR}}, (\mathbf{kc}_j, \mathbf{Kc}_j)_{j \leq m}) \leftarrow \mathsf{dat}$
4. $(\mathtt{EL}_{\mathtt{pk}}, \mathtt{EL}_{\mathtt{sk}}) \leftarrow \mathsf{SetupTally}(\mathtt{crs})$
5. $\mathsf{bb} \leftarrow \emptyset$ // Initialize an empty ballot box
6. Choose at random $\mathbf{v}_{\mathtt{id}} = (v_{j_1}, \ldots, v_{j_\psi})$. // Voter's selections
7. Set $\mathbf{bck}^* \leftarrow \mathbf{bck} \backslash \mathtt{BCK}_{\mathtt{id}}$ // Do not give BCK (ever) for Voter $\mathtt{id}$ to $\mathcal{A}$.
8. Give the following data to $\mathcal{A}$:

$$\mathsf{advDat} = (\mathbf{v}_{\mathtt{id}}, \mathbf{vcd}, \mathbf{svk}, (\mathbf{k}, \mathbf{K}), \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, \mathsf{bb}, \mathsf{CMtable}, \mathtt{C}_{\mathtt{sk}}, (\mathbf{kc}_j, \mathbf{Kc}_j)_{j \neq h}), \mathbf{Kc}_h, \mathbf{bck}^*)$$

9. $(\mathbf{cc}^*_{\mathtt{id}}, \mathtt{b}_{\mathtt{id}}) \leftarrow \mathsf{SendVote}$ // $\mathcal{A}$ submits ballot and Choice Return Codes and Voter stops interaction
10. $(\mathtt{CK}_{\mathtt{id}}, (\mathtt{lVCC}_{\mathtt{id},j}, \pi_{\mathtt{expCK},j})_{j \neq h}) \leftarrow \mathsf{ConfirmVote}$ // $\mathcal{A}$ submits Confirmation Key, and long Vote Cast Return Code shares for each CCR_j with $j \neq h$ without knowledge of $\mathtt{BCK}_{\mathtt{id}}$.
11. If $\mathsf{bad}_{\mathsf{inj}}$ output 1

Figure 27: Game modeling attacks against recorded as confirmed (injection) property of sVote. Protocols SetupVoting, SetupTally, SendVote and ConfirmVote are executed in interaction with the adversary $\mathcal{A}$, who keeps state across calls. Algorithms corresponding to honest CCR_h are controlled by the game. Public parameters $\mathtt{crs}$ are implicit.

16 Universal verifiability

We define the property *correct tally* of sVote as the inability of an adversary to alter the election result. Contrary to properties related to individual verifiability and privacy, an adversary against correct tally is given full control of the Mixing control components.

16.1 On ballot boxes ready for tally

A ballot box bb is *ready for tally* if it has been audited and, additionally, the setup phase has also been audited *before* bb is sent to tally. Thus:

$$\mathsf{bb} \text{ is ready for tally} \Rightarrow \begin{cases} 1 \leftarrow \mathsf{VerifyVotingPhase}(\mathsf{bb}, \mathsf{Logs}_{\mathtt{PO}}, \mathsf{Logs}_{\mathtt{serv}}, \{\mathsf{Logs}_{\mathrm{CCR}_j}\}_{j=1}^{m}) \\ 1 \leftarrow \mathsf{VerifyConfigPhase}(\mathsf{Logs}_{\mathtt{PO}}) \end{cases}$$

In other words, our formulation assumes the ballot box resulting from the voting phase in a consistent state with the transcript of the Server and the Choice Return Code control components. Only such a ballot box is given to an adversary against correct tally as input.

16.1.1 Extracting votes from ballots

VOTE EXTRACTOR. The votes embedded in a given confirmed ballot $\mathtt{b}_{\mathtt{id}}$ can be extracted from its corresponding vector of partial Choice Codes $\mathbf{pCC}_{\mathtt{id}}$ with the knowledge of the voter's Verification Card private key $\mathtt{k}_{\mathtt{id}}$. The extraction mechanism is simple, just compute the $\mathtt{k}_{\mathtt{id}}$-th roots of $\mathbf{pCC}_{\mathtt{id}}$.

Algorithm extractVoteWithAuxInfo($\mathbf{pCC}_{\mathtt{id}}, \mathtt{k}_{\mathtt{id}}, \mathtt{crs}$)

It takes as input the partial Choice Codes $\mathbf{pCC}_{\mathtt{id}} = (\mathtt{pCC}_{1,\mathtt{id}}, \ldots, \mathtt{pCC}_{\psi,\mathtt{id}})$ and the Verification Card private key $\mathtt{k}_{\mathtt{id}}$. Then it does the following:

1. Compute inverse $k'_{\mathtt{id}} = \mathtt{k}_{\mathtt{id}}^{-1} \mod q$.
2. For each $i \in (1, \ldots, \psi)$ set $v_{\mathtt{id},i} = \mathtt{pCC}_{i,\mathtt{id}}^{k'_{\mathtt{id}}} \in \mathbb{Q}_p$.

It outputs $\mathbf{v}_{\mathtt{id}} = (v_{\mathtt{id},1}, \ldots, v_{\mathtt{id},\psi})$. Note that step 1 is well-defined since q is prime and therefore there exist and inverse $k'_{\mathtt{id}} \in \mathbb{Z}_q^*$.

CORRECTNESS OF THE VOTE EXTRACTOR. A ballot box ready for tally contains confirmed ballots that are *compliant registered ballots* as per Definition 18 with very high probability. Intuitively, this follows from the vote compliance property of sVote (Lemma 2). Moreover, the above is true independently of whether or not the voter behaved honestly at the moment of casting her votes.

Lemma 3 (correct vote extraction) *Let* bb *be a ballot box that is ready for tally, let* $\mathtt{b}_{\mathtt{id}} = (\mathtt{vcd}_{\mathtt{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K}_{\mathtt{id}}, \boldsymbol{\pi}_{\mathsf{ballot}}) \in \mathsf{bb}$ *be a ballot with* $\mathbf{pCC}_{\mathtt{id}}$ *its corresponding vector of partial Choice Codes contained in the logs* $\mathsf{Logs}_{\mathtt{serv}}$ *of the Server, and let* $\mathtt{k}_{\mathtt{id}}$ *the Verification Card Private key of voter* id. *Then if*

(i) $\mathtt{E1} \in \mathsf{bb}_{\mathsf{clean}}$ *such that* $\mathsf{bb}_{\mathsf{clean}} \leftarrow \mathsf{Cleansing}(\mathsf{bb})$

(ii) $\nu \leftarrow \mathsf{Dec}(\mathtt{E1}, \mathtt{EL}_{\mathtt{sk}})$

(iii) $\mathbf{v}_{\mathtt{id}} \leftarrow \mathsf{extractVoteWithAuxInfo}(\mathbf{pCC}_{\mathtt{id}}, \mathtt{k}_{\mathtt{id}})$

it holds

$$Pr[\nu \neq \prod_{i=1}^{\psi} v_{\mathtt{id},i}] \leq Adv_{\mathcal{B}_1}^{\mathsf{sVote,mtc}} + 3 \cdot Adv_{\mathcal{B}_2}^{\mathsf{Exp,sSOUND}} + Adv_{\mathcal{B}_3}^{\mathsf{EqEnc,sSOUND}}.$$

where $\mathcal{B}_1$ *is an adversary against the mapping table correctness of* sVote, *(see Figure 23), and* $\mathcal{B}_2$, $\mathcal{B}_3$ *are adversaries against the simulation soundness property of the exponentiation and plaintext equality proof systems* Exp, EqEnc, *respectively.*

The proof of this lemma is straightforward. If the ballot $\mathtt{b}_{\mathtt{id}}$ is a *compliant registered ballot* as in Definition 18, then it clearly holds $\nu = \prod_{i=1}^{\psi} v_{\mathtt{id},i}$ with probability 1.

On the other hand, since the ballot box is ready for tally, the setup and voting phases have been audited. We can move to an ideal setup scenario, with an error probability upper bounded by $\mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{sVote,mtc}}$. Then, using Lemma 2, we upper bound the probability of a non-compliant registered ballot $\mathtt{b_{id}}$ ending up in the cleansed ballot box $\mathsf{bb_{clean}}$ with $3 \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{Exp,sSOUND}} + \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{EqEnc,sSOUND}}$.

16.2 Modeling correct tally

We define correct tally with an experiment comparing the election result corresponding to a real execution of the tally phase with the election result corresponding to a simulated (ideal) tally. The ideal tally ignores the Mixing control components and instead extract votes from the partial Choice Code vectors corresponding to confirmed ballots stored in the (already audited) ballot box.

Also, the game knows the Verification Card Private key $\mathtt{k_{id}}$ of all voters. Modeling with this piece of extra knowledge is done for simplicity and justified by two observations: (i) the Verification Card Private key $\mathtt{k_{id}}$ can be extracted from the ballot NIZK Proofs $\boldsymbol{\pi}_{\mathsf{ballot}}$, (ii) an alternative game formulation of correct tally accounting for the three phases [20]) would know these keys anyways since the game would control the Print Office during setup. In Figure 28 we define game $\mathsf{correctTally}$ capturing adversaries against the correctness of tally. The advantage of an adversary $\mathcal{A}$ is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,correctTally}} = Pr[1 \leftarrow \mathsf{correctTally}_{\mathcal{A}}^{\mathsf{sVote}}(\mathsf{bb}, \mathsf{Logs}_{\mathtt{PO}}, \mathsf{Logs}_{\mathtt{serv}}, \mathbf{k}, \mathtt{crs})].$$

In the game, event $\mathsf{bad}_{\mathsf{correctTally}}$ is defined as:

$$\mathsf{bad}_{\mathsf{correctTally}} = \begin{cases} L^*_{\mathsf{votes}} \neq L_{\mathsf{votes}} \\ L_{\mathsf{votes}} \neq \perp \end{cases}$$

where L^*_{votes} is the output result of the adversary $\mathcal{A}$ and L_{votes} is the result of the ideal tally executed by the game.

Theorem 4 *Let N be the total number of voters. In the random oracle model*

$$Adv_{\mathcal{A}}^{\mathsf{sVote,correctTally}} \leq Adv_{\mathcal{B}_1}^{\mathsf{sVote,mtc}} + N \cdot (3 \cdot Adv_{\mathcal{B}_2}^{\mathsf{Exp,sSOUND}} + Adv_{\mathcal{B}_3}^{\mathsf{EqEnc,sSOUND}}) + Adv_{\mathcal{B}_1}^{\mathsf{mix,sound}} + Adv_{\mathcal{B}_2}^{\mathsf{dec,sSOUND}},$$

where $\mathcal{B}_1$ is an adversary against the mapping table correctness of sVote*, $\mathcal{B}_2$, $\mathcal{B}_3$ are adversaries against the simulation soundness property of the exponentiation and plaintext equality proof systems* Exp, EqEnc*, respectively, $\mathcal{B}_4$ is an adversary against the soundness of the Shuffle proof system, and $\mathcal{B}_5$ an adversary against the soundness of the decryption proof system.*

Its proof is in Section 18.6.

[20] Accounting for the three phases when we only want to analyze the tally phase is redundant given the analysis of the previous sections, and unnecessarily complicates the game and the reduction with no clear benefit.

$\underline{\mathsf{correctTally}}^{\mathcal{A}^{\mathsf{sVote}}}(\mathsf{bb}, \mathsf{Logs}_{\mathtt{PO}}, \mathsf{Logs}_{\mathtt{serv}}, \mathbf{k}, \mathtt{crs})$:

1. $\left(L^*_{\mathsf{votes}}, (\mathsf{Logs}_{\mathtt{CCM}_j})_{j=1}^{m'}\right) \leftarrow \mathcal{A}(\mathsf{bb})$ // $\mathcal{A}$ executes protocols MixOnline and MixOffline
2. Audit tally phase:
   2.1 $b_1 \leftarrow \mathsf{VerifyOnlineTally}(\mathsf{bb}, \{\mathsf{Logs}_{\mathtt{CCM}_j}\}_{j=1}^{m'-1}, \mathsf{Logs}_{\mathtt{serv}}, \mathsf{Logs}_{\mathtt{PO}})$
   2.2. $b_2 \leftarrow \mathsf{VerifyOfflineTally}(L^*_{\mathsf{votes}}, \mathbf{m}, \mathsf{Logs}_{\mathtt{CCM}_{m'}}, \mathsf{Logs}_{\mathtt{PO}})$
3. If $b_1 = 0$ or $b_2 = 0$ set $L_{\mathsf{votes}} \leftarrow \perp$
4. Else, execute the ideal tally:
   4.1. $\mathsf{bb}_{\mathsf{clean}} \leftarrow \mathsf{Cleansing}(\mathsf{bb})$
   4.2. Get list $\mathsf{L}_{\mathtt{decPCC}}$ from $\mathsf{Logs}_{\mathtt{serv}}$
   4.3. For each $\mathtt{E1} \in \mathsf{bb}_{\mathsf{clean}}$ do:
      4.3.1 Find entry $(\mathtt{b_{id}}, \mathbf{pCC}_{\mathtt{id}}, \star, \star) \in \mathsf{L}_{\mathtt{decPCC}}$ with $\mathtt{E1} \subset \mathtt{b_{id}}$ // Get partial Choice Codes
      4.3.2 Get $\mathtt{k_{id}}$ from $\mathbf{k}$ // Get Verification Card private key for voter $\mathtt{id}$
      4.3.3 $\mathbf{v}_{\mathtt{id}} = (v_{\mathtt{id},1}, \ldots, v_{\mathtt{id},\psi}) \leftarrow \mathsf{extractVoteWithAuxInfo}(\mathbf{pCC}_{\mathtt{id}}, \mathtt{k_{id}})$ // Extract votes
   4.4. Check if $v_{\mathtt{id},i}$ is in $\mathbf{v}$, if not set $L_{\mathsf{votes}} \leftarrow \perp$ // Check extracted vote is a valid voting option
   4.5. Else, set $L_{\mathsf{votes}} \leftarrow \{\mathbf{v}_{\mathtt{id}}\}_{\mathtt{id} \in \mathcal{ID}}$
5. If $\mathsf{bad}_{\mathsf{correctTally}}$ output 1

Figure 28: Game modeling attacks against correct tally. The adversary $\mathcal{A}$ is in full control of the Mixing control components CCMs. In step 2.1, algorithm VerifyOnlineTally checks that the inputs $\mathsf{bb}, \mathsf{Logs}_{\mathtt{PO}}, \mathsf{Logs}_{\mathtt{serv}}$ are the same of algorithm VerifyVotingPhase (see Section 12.2.3). This ensures that the ballot box passed as input has been audited, thus it is ready for tally. The game executes an ideal tally in step 4 via vote extraction from the partial Choice Codes using the Verification Card private keys of the voters. Public parameters $\mathtt{crs}$ are implicit.

17 Privacy

We define the property *ballot privacy* of sVote as the inability to learn information about the cast votes, beyond what is unavoidably leaked by the election results.

For the privacy case, Voting Client is considered to be honest. Note that this assumption is fair for any client-side encryption based e-voting system since the voting options are introduced in clear. Thus, the honest parts of the system would be Voting Client , Print Office , one of Electoral Board members, one of CCRs and one of CCMs.

For simplicity we consider only two CCMs, whereas in [43] four mixing control components are specified. We claim that our reduction does not affect proof structure due to the mandatory audit performed before the last CCM is executed and the fact that the last key is distributed among members of Electoral Board.

Consider a case of N CCMs where only one of them is honest. In such scenario, the mandatory verification of online mixing tally (VerifyOnlineTally) would be performed after CCM_{N-1} execution. If and only the verification holds, Electoral Board members [21] would submit their private key shares and

[21] Electoral Board members can be viewed as a set of 'human control components', thus at least one of the members

CCM_N will be able to reconstruct its private key. All possible corruption scenarios can be mapped to 4 cases as presented in Table 5.

	Honest CCM is among first N-1	VerifyOnlineTally outputs $\top$	Honest EB members submit their keyshares	CCM_N knows its key	Case description
Case 1	Yes	No	No	No	CCM_N is corrupted, but can't reconstruct its key.
Case 2		Yes	Yes	Yes	CCM_N is corrupted and can reconstruct its key.
Case 3	No	No	No	No	CCM_N is honest, but can't reconstruct its key.
Case 4		Yes	Yes	Yes	CCM_N is honest and knows its key.

Table 5: Possible corruption scenarios in case of N CCMs and Electoral Board

It is easy to see, that all corruption cases can be efficiently modeled with just two CCMs due to the fact that last CCM doesn't know its key until VerifyOnlineTally verification is successful. Therefore, in privacy definition and proofs are modeled are only two CCMs and the Challenger holds the last CCM's private key $\mathtt{EL}_{\mathtt{sk},\mathtt{EB}}$. The Challenger reveals this key to $\mathcal{A}$ if and only if $\mathcal{A}$ chooses to corrupt the last mixing node and VerifyOnlineTally outputs $\top$.

We base our definition of the ballot privacy on the definition defined in [10] for the Helios scheme and adopt it to sVote as shown in 19.

On the high level, the idea of the game-based ballot privacy definition is to capture the situation, where an adversary that provides voter with two possible voting options sets $\mathbf{v}^0_{\mathtt{id}}, \mathbf{v}^1_{\mathtt{id}}$ cannot distinguish which one was chosen by an honest voter. To avoid trivial wins, the adversary is restricted to select only sets that would produce the same result i.e. $\{\mathbf{v}^0_{\mathtt{id}}\}_{\mathtt{id}\in\Omega}$ should be a permutation of $\{\mathbf{v}^1_{\mathtt{id}}\}_{\mathtt{id}\in\Omega}$, where Ω are all honest voters ids that confirmed their votes successfully (both Choice Return Codes and Vote Cast Code).

The game has 3 phases:

Setup During the setup phase, the challenger and adversary jointly participate in the election configuration and set up process. At the end of this phase, the adversary additionally receives private information of all corrupt voters.

Voting During the voting phase, the Adversary can schedule vote casting, vote confirmation (honest voter checks Choice Return Codes and, if codes are valid, introduces $\mathtt{BCK}_{\mathtt{id}}$) and vote verification (honest voter checks Vote Cast Code) of all honest voters as well as request an honest CCR to perform

is honest and refuses to submit the decryption key share if validation fails. The use of humans as control components is permitted by section 4.4.10 of [16], "it is also permitted to implement a group of control components so that they take the form of people".

necessary actions for Choice Return Codes or Vote Cast Code retrieval. The challenger, that is playing on behalf of honest parties, flips a coin β and uses $\{\mathbf{v}^{\beta}_{\mathtt{id}}\}$ for all honest voters, also it keeps track of all cast, verified, confirmed ballots and failed verifications.

Tally The Adversary can select which CCM it wishes to corrupt[22]. The challenge's behavior depends on the coin β. In case, when $\beta = 0$, the challenger honestly executes an honest CCM and outputs the result. However, if $\beta = 1$, the challenger would change honest voters' ballots cast for voting options $\{\mathbf{v}^{1}_{\mathtt{id}}\}$ to those generated for $\{\mathbf{v}^{0}_{\mathtt{id}}\}$ and use simulators to fake shuffle and decryption proofs.

Definition 19 (Privacy) *Consider* sVote *scheme described in section 12 for a set $\mathcal{ID}$ of voter identities. We say that the scheme has ballot privacy if there exist algorithms $\mathcal{S}_{\mathsf{dec}}$ and $\mathcal{S}_{\mathsf{Mix}}$ such that:*

$$Adv^{\mathsf{sVote,bpriv}}_{\mathcal{A}} = \left| Pr\left[1 \leftarrow [\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V}}(\mathtt{crs})\right] - Pr\left[1 \leftarrow [\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V}}(\mathtt{crs})\right] - \frac{1}{|\mathcal{C}_{svk}|}\right| \approx 0,$$

where $\mathcal{C}_{svk}$ is a code space of Start Voting Keys $\mathtt{SVK}_{\mathtt{id}}$.

Theorem 5 *In the random oracle model*

$$\begin{aligned} Adv^{\mathsf{sVote,bpriv}}_{\mathcal{A}} \leq\ & Adv^{\mathsf{dec,NIZK}}_{\mathcal{B}.1} + Adv^{\mathsf{mix,NIZK}}_{\mathcal{B}.2} + Adv^{\mathsf{Exp,NIZK}}_{\mathcal{B}.3} + Adv^{\mathsf{sVote,mtc}}_{\mathcal{B}.4} \\ & + Adv^{\mathsf{sch,NIZK}}_{\mathcal{B}.5} + + Adv^{\mathsf{EqEnc,NIZK}}_{\mathcal{B}.6} + N_h \cdot Adv^{\mathsf{SEnc,INDCPA}}_{\mathcal{B}.7} \\ & + Adv^{F,\mathsf{PRF}}_{\mathcal{B}.8} + N_h \cdot Adv^{\mathbb{Q}_p,\mathsf{ESGSP}}_{\mathcal{B}.9} + Adv^{\mathsf{ElGamal,INDCPA}}_{\mathcal{B}.10}, \end{aligned}$$

where $\mathcal{B}.1$ is an adversary against the zero-knowledge property of the Decryption proof system dec*, $\mathcal{B}.2$ is an adversary against the zero-knowledge property of the shuffle argument* mix*, $\mathcal{B}.3$ is an adversary against the zero-knowledge property of the* Exp *proof system, $\mathcal{B}.4$ is an adversary against mapping table correctness property of* sVote*, $\mathcal{B}.5$ is an adversary against the zero-knowledge property of the* sch *proof system, $\mathcal{B}.6$ is an adversary against the zero-knowledge property of the* EqEnc *proof system, $\mathcal{B}.7$ is an adversary against IND-CPA property of symmetric encryption scheme, $\mathcal{B}.8$ is an adversary against the weak pseudorandom property of the exponentiation function $F(k, x) = x^k$, $\mathcal{B}.9$ is an adversary against the* ESGSP *problem [25], $\mathcal{B}.10$ is an adversary against the IND-CPA property of the ElGamal.*

Its proof is in Section 18.7.

[22]Note that, if CCM_2 is corrupt, it will receive its private share of election key if and only if CCM_1 doesn't abort.

$\underline{\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V}}(\mathtt{crs})}$:

//setup phase

1. $\mathcal{ID}, \mathcal{ID}_h, h, c \leftarrow \mathcal{A}(\mathtt{crs})$ // $\mathcal{A}$ specifies voter identities, honest voters, an honest CCR_h and whether it corrupts online ($c = 1$) or offline ($c = 0$) CCM
2. $\mathsf{dat} \leftarrow \mathsf{SetupVoting}_{\mathcal{A}}(\mathtt{crs})$
3. $(\mathtt{EL}_{\mathtt{pk}}, \mathtt{EL}_{\mathtt{pk},1}, \mathtt{EB}_{\mathtt{pk}}, \mathtt{EB}_{\mathtt{sk}}) \leftarrow \mathsf{SetupTally}_{\mathcal{A}}(\mathtt{crs})$ // $\mathtt{EB}_{\mathtt{sk}}$ is not known to $\mathcal{A}$ regardless of which CCM is corrupted. If and only if $c = 1$, $\mathtt{EL}_{\mathtt{sk},1}$ is generated by $\mathcal{A}$.
4. $b \leftarrow \mathsf{VerifyConfigPhase}(\mathsf{Logs}_{\mathtt{PD}}, \mathtt{crs})$. If $b = \perp$, abort the game.
5. $\mathsf{bb}, \mathsf{bb}_0, \mathsf{bb}_1 \leftarrow \emptyset$ // Initialize 3 empty ballot boxes
6. At the end of this phase $\mathcal{A}$ obtains:
   - server's data: $\mathsf{bb}, \mathtt{C}_{\mathtt{sk}}, \mathsf{CMtable}$
   - public data: $\mathbf{vcd}, \mathbf{K}, \mathbf{VCks}, \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, \mathtt{EB}_{\mathtt{pk}}, \mathtt{EL}_{\mathtt{pk},1}$
   - corrupt voters' data: $(\mathbf{cc}_{\mathtt{id}}, \mathtt{SVK}_{\mathtt{id}}, \mathtt{VCC}_{\mathtt{id}}, \mathtt{BCK}_{\mathtt{id}})_{\mathtt{id} \in \mathcal{ID} \setminus \mathcal{ID}_h}$

//voting phase

$\mathcal{A}$ is allowed to query oracles $\mathcal{O}\mathsf{CastBallot}, \mathcal{O}\mathsf{HonestCCRdec}, \mathcal{O}\mathsf{ConfirmBallot}, \mathcal{O}\mathsf{HonestCCRconfirm}, \mathcal{O}\mathsf{VerifyVCC}$ defined in Figure 30.

//tally phase

$\mathcal{A}$ is allowed to query oracle $\mathcal{O}\mathsf{HonestMix}$ defined in Figure 31 only once.

Figure 29: Experiments $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V}}(\mathtt{crs})$ defined for $\beta \in \{0, 1\}$. The adversary $\mathcal{A}$ has access to set of oracles $\mathcal{O}\mathsf{CastBallot}$, $\mathcal{O}\mathsf{HonestCCRdec}$, $\mathcal{O}\mathsf{ConfirmBallot}$,$\mathcal{O}\mathsf{HonestCCRconfirm}$,$\mathcal{O}\mathsf{VerifyVCC}$, $\mathcal{O}\mathsf{HonestMix}$. $\mathcal{A}$ can query oracle $\mathcal{O}\mathsf{HonestMix}$ only once. For $\beta = 1$ the experiment also depends on $\mathcal{S}_{\mathsf{dec}}$ and $\mathcal{S}_{\mathsf{Mix}}$, To simplify notation, oracle's explicit dependence on β, $\mathcal{S}_{\mathsf{Mix}}$ and $\mathcal{S}_{\mathsf{dec}}$ is not shown.

$\mathcal{O}\mathsf{CastBallot}(\mathtt{id}, \mathbf{v}^0_{\mathtt{id}}, \mathbf{v}^1_{\mathtt{id}})$: //cast ballot on behalf of honest voter

1. If $\mathtt{id} \notin \mathcal{ID}_h$ or $\mathbf{v}^0_{\mathtt{id}}, \mathbf{v}^1_{\mathtt{id}}$ have more than ψ voting options, abort.
2. $\mathtt{k}_{\mathtt{id}} \leftarrow \mathsf{GetKey}(\mathtt{SVK}_{\mathtt{id}}, \mathtt{VCks}_{\mathtt{id}})$ // obtain verification card private key
3. $\mathtt{b}^0_{\mathtt{id}} \leftarrow \mathsf{CreateVote}(\mathtt{VC}_{\mathtt{id}}, \mathbf{v}^0_{\mathtt{id}}, \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, (\mathtt{K}_{\mathtt{id}}, \mathtt{k}_{\mathtt{id}}), \mathtt{crs})$ // ballot for option $v^0_{\mathtt{id}}$
   $\mathtt{b}^1_{\mathtt{id}} \leftarrow \mathsf{CreateVote}(\mathtt{VC}_{\mathtt{id}}, \mathbf{v}^1_{\mathtt{id}}, \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, (\mathtt{K}_{\mathtt{id}}, \mathtt{k}_{\mathtt{id}}), \mathtt{crs})$ // ballot for option $v^1_{\mathtt{id}}$
4. $\nu^i_{\mathtt{id}} = \prod_{k=1}^{\psi} v^i_k$, where $\mathbf{v}^i_{\mathtt{id}} = (v_1, \ldots, v_\psi)$ for $i \in \{0, 1\}$
5. $\mathsf{bb}_0 \leftarrow \mathsf{bb}_0 \cup (\mathtt{id}, \nu^0_{\mathtt{id}}, \mathtt{b}^0_{\mathtt{id}})$ and $\mathsf{bb}_1 \leftarrow \mathsf{bb}_1 \cup (\mathtt{id}, \nu^1_{\mathtt{id}}, \mathtt{b}^1_{\mathtt{id}})$
6. Output $\mathtt{b}^\beta_{\mathtt{id}}$

$\mathcal{O}\mathsf{HonestCCRdec}(\mathtt{b}_{\mathtt{id}})$: //honest CCR partially decrypt encrypted $\mathbf{pCC}_{\mathtt{id}}$

1. $(b, \mathsf{Logs}_{\mathrm{CCR}_h}) \leftarrow \mathsf{VerifyBallotCCR}_{\mathtt{h}}(\mathtt{b}_{\mathtt{id}}, \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, \mathtt{K}_{\mathtt{id}}, \mathsf{Logs}_{\mathrm{CCR}_h}, \mathtt{crs})$
2. If $b = 0$ abort, otherwise proceed.
3. $(\mathbf{d}_{j,\mathtt{id}}, \mathsf{Logs}_{\mathrm{CCR}_j}) \leftarrow \mathsf{PartialDecryptPCC}_{\mathtt{h}}(\mathtt{b}_{\mathtt{id}}, \mathtt{pk}_{\mathrm{CCR},h}, \mathtt{sk}^{(h)}_{\mathrm{CCR}}, \mathsf{Logs}_{\mathrm{CCR}_h}, \mathtt{crs})$
4. Output $\mathbf{d}_{j,\mathtt{id}}$

$\mathcal{O}\mathsf{HonestCCRexp}(\mathtt{b}_{\mathtt{id}}, \mathbf{pCC}_{\mathtt{id}})$: //honest CCR exponentiate $\mathbf{pCC}_{\mathtt{id}}$

1. $(\mathbf{lCC}_{h,\mathtt{id}}, \mathsf{Logs}_{\mathrm{CCR}_h}) \leftarrow \mathsf{CreateLCCShare}(\mathtt{b}_{\mathtt{id}}, \mathbf{pCC}_{\mathtt{id}}, \mathtt{k}_{h,\mathtt{id}}, \mathtt{K}_{h,\mathtt{id}}, \mathsf{Logs}_{\mathrm{CCR}_h}))$
2. Output $\mathbf{lCC}_{h,\mathtt{id}}$ // the result is either $\perp$ or $\mathbf{lCC}_{h,\mathtt{id}}$

$\mathcal{O}\mathsf{ConfirmBallot}(\mathtt{id}, \mathbf{cc}^*_{\mathtt{id}})$: //confirm ballot on behalf of an honest voter

1. If bb_β has an entry $(\mathtt{id}, \mathsf{confirmed})$ or $(\mathtt{id}, \mathsf{wrongCRCs})$, abort.
2. $b \leftarrow \mathsf{VerifyCRC}(v_\beta, \mathbf{cc}^*_{\mathtt{id}}, \mathbf{cc}^\beta_{\mathtt{id}})$
3. If $b = 0$:
   $\mathsf{bb}_0 \leftarrow \mathsf{bb}_0 \cup (\mathtt{id}, \mathsf{wrongCRCs})$
   $\mathsf{bb}_1 \leftarrow \mathsf{bb}_1 \cup (\mathtt{id}, \mathsf{wrongCRCs})$
   abort.
4. $\mathtt{CM}^{\mathtt{id}} \leftarrow \mathsf{CreateConfirmMessage}(\mathtt{BCK}_{\mathtt{id}}, \mathtt{k}_{\mathtt{id}})$
5. $\mathsf{bb}_0 \leftarrow \mathsf{bb}_0 \cup (\mathtt{id}, \mathsf{confirmed})$ and $\mathsf{bb}_1 \leftarrow \mathsf{bb}_1 \cup (\mathtt{id}, \mathsf{confirmed})$
6. Output $\mathtt{CM}^{\mathtt{id}}$.

$\mathcal{O}\mathsf{HonestCCRconfirm}(\mathtt{b}_{\mathtt{id}}, \mathtt{CM}^{\mathtt{id}})$: //honest CCR confirms ballot

1. $b \leftarrow \mathsf{VerifyConfirmationCCR}_{\mathtt{h}}(\mathtt{b}_{\mathtt{id}}, \mathsf{Logs}_{\mathrm{CCR}_h})$
2. If $b = 0$, abort.
3. $(\mathtt{lVCC}_{\mathtt{id},h}, \mathsf{Logs}_{\mathrm{CCR}_h}) \leftarrow \mathsf{CreateLVCCShare}_{\mathtt{h}}(\mathtt{b}_{\mathtt{id}}, \mathtt{CM}^{\mathtt{id}}, (\mathtt{Kc}_{h,\mathtt{id}}, \mathtt{kc}_{h,\mathtt{id}}), \mathsf{Logs}_{\mathrm{CCR}_h})$.
4. Output $\mathtt{lVCC}_{\mathtt{id},h}$.

$\mathcal{O}\mathsf{VerifyVCC}(\mathtt{id}, \mathtt{VCC}^*_{\mathtt{id}})$: //verify VCC on behalf of an honest voter

1. If bb_β has an entry $(\mathtt{id}, \mathsf{verified})$ or $(\mathtt{id}, \mathsf{wrongVCC})$, abort.
2. $b \leftarrow \mathsf{VerifyVCC}(\mathtt{VCC}^*_{\mathtt{id}}, \mathtt{VCC}_{\mathtt{id}})$.
3. if $b = 0$
   $\mathsf{bb}_0 \leftarrow \mathsf{bb}_0 \cup (\mathtt{id}, \mathsf{wrongVCC})$
   $\mathsf{bb}_1 \leftarrow \mathsf{bb}_1 \cup (\mathtt{id}, \mathsf{wrongVCC})$
   abort.
4. $\mathsf{bb}_0 \leftarrow \mathsf{bb}_0 \cup (\mathtt{id}, \mathsf{verified})$ and $\mathsf{bb}_1 \leftarrow \mathsf{bb}_1 \cup (\mathtt{id}, \mathsf{verified})$
5. Output 1.

Figure 30: Privacy: Oracles given to the adversary $\mathcal{A}$ in experiments $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V}}(\mathtt{crs})$ (see Figure 29) modeling the voting phase. They are implicitly parameterized with bit $\beta \in \{0, 1\}$.

$\mathcal{O}\mathsf{HonestMix}(\mathsf{bb}, \mathsf{Logs}_{\mathtt{serv}}, \{\mathsf{Logs}_{\mathtt{CCR}_{arg1}}\}_{j=1,j\neq h}^{m}, \mathsf{bb}_{\mathsf{clean}}, \mathbf{c}_{\mathsf{mix},\mathcal{A}}, \mathbf{c}_{\mathsf{Dec},\mathcal{A}}, \mathsf{Logs}_{\mathtt{CCM}_{\mathcal{A}}})$: // tally

1. If for $\mathtt{id} \in \mathcal{ID}_h$ bb_β has $(\mathtt{id}, \mathsf{confirmed})$ entry, but not $(\mathtt{id}, \mathsf{verified})$, raise a complaint.
2. $b \leftarrow \mathsf{VerifyVotingPhase}(\mathsf{bb}, \mathsf{Logs}_{\mathtt{PO}}, \mathsf{bb}, \mathsf{Logs}_{\mathtt{serv}}, \{\mathsf{Logs}_{\mathtt{CCR}_{arg1}}\}_{j=1}^{m})$
3. If $b = \perp$, abort
4. //Check that adversary cannot trivially break privacy based on tally results

   $\mathbf{V}_h^1 \leftarrow \{\nu_{\mathtt{id}}^1 : (\mathtt{id}, \mathsf{verified}) \in \mathsf{bb}_1\}$// extract verified options of honest voters from bb_1

   $\mathbf{V}_h^0 \leftarrow \{\nu_{\mathtt{id}}^0 : (\mathtt{id}, \mathsf{verified}) \in \mathsf{bb}_0\}$// extract verified options of honest voters from bb_0
5. If $\mathbf{V}_h^0$ is not a permutation of $\mathbf{V}_h^1$, abort.
6. if $c = 0$: //online mixing $(\mathtt{EL}_{\mathtt{pk},h}, \mathtt{EL}_{\mathtt{sk},h}) = (\mathtt{EL}_{\mathtt{pk},1}, \mathtt{EL}_{\mathtt{sk},1})$

   if $\beta = 0$:

   $(\mathbf{c}_{\mathsf{mix},h}, \mathbf{c}_{\mathsf{Dec},h}, \overline{\mathtt{EL}}_{\mathtt{pk},h}, \mathsf{Logs}_{\mathtt{CCM}_h}) \leftarrow \mathsf{MixDecOnline}_h(\mathsf{bb}_{\mathsf{clean}}, \mathtt{EL}_{\mathtt{pk}}, \mathtt{EL}_{\mathtt{pk},h}, \mathtt{EL}_{\mathtt{sk},h}, \mathsf{Logs}_{\mathtt{CCR}_h}, \mathtt{crs})$

   else if $\beta = 1$: // simulate mixing and decryption proofs

   $\mathtt{b}_h^1 \leftarrow \{\mathtt{b}_{\mathtt{id}} : (\mathtt{id}, \mathsf{verified}) \in \mathsf{bb}_1\}$// extract verified ballots of honest voters from bb_1

   $\mathtt{b}_h^0 \leftarrow \{\mathtt{b}_{\mathtt{id}} : (\mathtt{id}, \mathsf{verified}) \in \mathsf{bb}_0\}$// extract verified ballots of honest voters from bb_0

   $\mathsf{bb}_h = \mathsf{bb}_{\mathsf{clean}}/\mathtt{b}_h^1 \cup \mathtt{b}_h^0$ //change honest voter's ballots

   Permute entries in bb_h to get $\mathsf{bb}_h^{\mathsf{shuffled}}$

   $(\mathbf{c}_{\mathsf{mix},h}, \pi_{\mathsf{mix},h}^*) = \mathcal{S}_{\mathsf{Mix}}(\mathtt{EL}_{\mathtt{pk}}, \mathtt{ck}_{\mathsf{mix}}, \mathsf{bb}_{\mathsf{clean}}, \mathsf{bb}_h^{\mathsf{shuffled}})$. //simulate shuffle proof

   Compute $\mathbf{c}_{\mathsf{Dec},h} = (\overline{\mathbf{c}}_1, \ldots, \overline{\mathbf{c}}_\ell)$ as $\overline{\mathbf{c}}_i = \left(c_{i,1}, c_{i,2} \cdot (c_{i,1})^{-\mathtt{EL}_{\mathtt{sk},h}}\right)$, for each $\mathbf{c}_i = (c_{i,1}, c_{i,2}) \in \mathbf{c}_{\mathsf{mix},h}$

   and a list of NIZKs $\boldsymbol{\pi}_{\mathsf{dec},h}^* = \left(\pi_{\mathsf{dec},h,1}^*, \ldots, \pi_{\mathsf{dec},h,\ell}^*\right)$, where $\pi_{\mathsf{dec},h,i}^* = \mathcal{S}_{\mathsf{dec}}\left(g, \mathtt{EL}_{\mathtt{pk},h}, \mathbf{c}_i, \overline{\mathbf{c}}_i\right)$.

   $\mathsf{Logs}_{\mathtt{CCM}_h} \leftarrow \mathsf{Logs}_{\mathtt{CCM}_h} \cup \left((\mathsf{bb}_{\mathsf{clean}}), (\mathbf{c}_{\mathsf{mix},h}, \mathbf{c}_{\mathsf{Dec},h}), (\mathtt{EL}_{\mathtt{pk}}/\mathtt{EL}_{\mathtt{pk},h}, \mathtt{EL}_{\mathtt{pk}}), (\pi_{\mathsf{mix},h}^*, \boldsymbol{\pi}_{\mathsf{dec},h}^*)\right)$.

   Output: $(\mathtt{EL}_{\mathtt{sk},\mathtt{EB}}, \mathbf{c}_{\mathsf{mix},h}, \mathbf{c}_{\mathsf{Dec},h}, \mathsf{Logs}_{\mathtt{CCM}_h})$. //offline CCM private key and mixing results
6. else if $c = 1$ // offline mixing

   $b \leftarrow \mathsf{VerifyOnlineTally}(\mathsf{bb}, \mathsf{Logs}_{\mathtt{PO}}, \mathsf{Logs}_{\mathtt{serv}}, \mathsf{Logs}_{\mathtt{CCM}_h}, \mathsf{Logs}_{\mathtt{CCM}_{\mathcal{A}}} \mathbf{c}_{\mathsf{mix},\mathcal{A}}, \mathbf{c}_{\mathsf{Dec},\mathcal{A}})$

   If $b = \perp$, abort

   if $\beta = 0$:

   $(\mathbf{c}_{\mathsf{mix},h}, \mathbf{m}, \mathsf{Logs}_{\mathtt{CCM}_h}) \leftarrow \mathsf{MixDecOffline}(\mathbf{c}_{\mathsf{mix},\mathcal{A}}, \mathbf{c}_{\mathsf{Dec},\mathcal{A}}, \mathtt{EB}_{\mathtt{pk}}, \mathtt{EB}_{\mathtt{sk}}, \mathsf{Logs}_{\mathtt{CCR}_h}, \mathtt{crs})$

   Output: $(\mathbf{c}_{\mathsf{mix},h}, \mathbf{m}, \mathsf{Logs}_{\mathtt{CCR}_h})$

   else if $\beta = 1$: // simulate mixing and decryption proofs

   $\mathbf{m}^h = \mathbf{m}/\mathbf{V}_h^1 \cup \mathbf{V}_h^0$ // substitute $\mathbf{V}_h^1$ with $\mathbf{V}_h^0$ in $\mathbf{m}$

   Permute entries in $\mathbf{c}_{\mathsf{Dec},\mathcal{A}}$ to get $\mathbf{c}_{\mathsf{Dec},\mathcal{A}}^{\mathsf{shuffled}}$

   $(\mathbf{c}_{\mathsf{mix},h}, \pi_{\mathsf{mix},h}^*) = \mathcal{S}_{\mathsf{Mix}}(\mathtt{EL}_{\mathtt{pk}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_{\mathsf{Dec},\mathcal{A}}, \mathbf{c}_{\mathsf{Dec},\mathcal{A}}^{\mathsf{shuffled}})$. //simulate shuffle proof

   Compute $\boldsymbol{\pi}_{\mathsf{dec},h}^* = \left(\pi_{\mathsf{dec},h,1}^*, \ldots, \pi_{\mathsf{dec},h,\ell}^*\right)$ as $\pi_{\mathsf{dec},h,i}^* = \mathcal{S}_{\mathsf{dec}}\left(g, \mathtt{EB}_{\mathtt{pk}}, \mathbf{c}_i, m_i^h\right)$, for each $m_i^h \in \mathbf{m}^h$, $\mathbf{c}_i \in \mathbf{c}_{\mathsf{mix},h}$.

   $\mathsf{Logs}_{\mathtt{CCM}_h} = \mathsf{Logs}_{\mathtt{CCM}_h} \cup \{(\mathbf{c}_{\mathsf{mix},\mathcal{A}}, \mathbf{c}_{\mathsf{Dec},\mathcal{A}}), (\mathbf{c}_{\mathsf{mix},h}, \mathbf{m}^h), (\overline{\mathtt{EL}}_{\mathtt{pk},\mathcal{A}}, \mathtt{EB}_{\mathtt{pk}}), (\pi_{\mathsf{mix},h}^*, \boldsymbol{\pi}_{\mathsf{dec},h}^*)\}$.

   Output: $(\mathbf{c}_{\mathsf{mix},h}, \mathbf{m}^h, \mathsf{Logs}_{\mathtt{CCM}_h})$.

Figure 31: Privacy: Oracle given to $\mathcal{A}$ in experiments $\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{bpriv},\beta}(\mathtt{crs})$ (see Figure 29) during the tally phase. It can be queried only once. The oracle is implicitly parameterized with bit $\beta \in \{0,1\}$ and simulators $\mathcal{S}_{\mathsf{dec}}$ and $\mathcal{S}_{\mathsf{Mix}}$.

18 Security reductions

18.1 Proof of Lemma 1 (codes mapping table correctness)

We use six hops to transform the real game rMTC (executing the real protocol $\mathsf{SetupVoting}$ of Figure 17) into the simulated game sMTC (executing the simulated protocol $\mathsf{IdealSetupVoting}$ of Figure 21). Both games are defined in Figure 23. After the transformation is completed, clearly $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,mtc.6}} = 0$ since the real and simulated games are identical.

GAME rMTC.1. In this game, algorithm $\mathsf{CombineEncLongCodeShares}$ (see Section 12.1.1.6) ignores input from CCR_j, $\forall j \neq h$ and instead uses the extractor $\mathcal{E}_{\mathcal{A}}$ of the Exponentiation proof system to obtain witnesses $\{\mathtt{k}_{j,\mathtt{id}}, \mathtt{kc}_{j,\mathtt{id}}\}_{j=1,j\neq h}^{m}$ from the NIZKs $\{\boldsymbol{\pi}_{\mathsf{expPCC},j}, \boldsymbol{\pi}_{\mathsf{expCK},j}\}_{j=1,j\neq h}^{m}$.

Additionally, to answer consistently between phases, the honest CCR_h keeps an internal Codes mapping table for Voter $\mathtt{id}$

$$\mathsf{CMtable}_{\mathtt{id}}^{\mathrm{CCR}_h} = \Big\{\mathtt{id}, \big\{[\mathtt{pCC}_{i,\mathtt{id}}, x_i]\big\}_{i=1}^{n}, [\mathtt{CK}_{\mathtt{id}}, y]\Big\},$$

where $x_i = \big((\mathsf{H}(v_i^{\mathtt{k}_{\mathtt{id}}})\big)^{2\cdot\mathtt{k}_{h,\mathtt{id}}})$, and $y = \big(\mathsf{H}(\mathtt{CK}_{\mathtt{id}})\big)^{2\cdot\mathtt{kc}_{h,\mathtt{id}}}$.

Then, algorithm $\mathsf{GenCMTable}$ (see Section 12.1.1.7) constructs $\mathsf{CMtable}$ based on the extracted keys of the dishonest CCR_j, $\forall j \neq h$, and the group elements x_i, y of the internal table of CCR_h.

Let bad_1 be the event that some of the NIZKs $\{\boldsymbol{\pi}_{\mathsf{expPCC},j}, \boldsymbol{\pi}_{\mathsf{expCK},j}\}_{j=1,j\neq h}^{m}$ verify successfully but $\mathcal{E}_{\mathcal{A}}$ could not extract a witness. The game aborts if bad_1 is true. We have that:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,mtc.1}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,mtc}}| \leq Pr[\mathsf{bad}_1].$$

Now, $Pr[\mathsf{bad}_1] \leq \epsilon_{ext}$, where ϵ_{ext} is given by the weak simulation extractability property of the Exponentiation proof system Exp.

GAME rMTC.2. In this game, algorithm $\mathsf{GenEncLongCodeShares_h}$ (see section 12.1.1.5), controlled by the honest CCR_h, uses the canonical simulator $\mathcal{S}_{\mathsf{Exp}}$ to generate proofs for statements containing CCR_h's public keys $\mathtt{K}_{h,\mathtt{id}}$ and $\mathtt{Kc}_{h,\mathtt{id}}$. We have that

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,mtc.2}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,mtc.1}}| \leq \mathsf{Adv}_{\mathcal{B}}^{\mathsf{Exp,NIZK}},$$

where $\mathcal{B}$ is an adversary against the zero-knowledge property of the Exponentiation proof system Exp. In the reduction, $\mathcal{B}$ uses its own oracle to answer the queries of CCR_h and outputs what the (emulated) game rMTC outputs (see Figure 5).

Observe that from this game onwards, CCR_h's private keys $\mathtt{k}_{h,\mathtt{id}}, \mathtt{kc}_{h,\mathtt{id}}$ are used for exponentiation (but not as witnesses in the setup phase).

GAME rMTC.3. Algorithm $\mathsf{GenEncLongCodeShares_h}$ samples at random the Voter Choice Return Code Generation private key $\mathtt{k}_{h,\mathtt{id}}$ and the Voter Vote Cast Return Code Generation private key $\mathtt{kc}_{h,\mathtt{id}}$ instead

of deriving them from the CCR_j Choice Return Codes Generation private key $\mathtt{k'}_h$. We have that

$$|\mathsf{Adv}^{\mathsf{sVote,mtc.3}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,mtc.2}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{kdf}}_{\mathcal{B}},$$

where $\mathcal{B}$ is an adversary against the pseudorandom property of KDF. The reduction is straightforward.

Game rMTC.4. In this game, algorithm $\mathsf{GenEncLongCodeShares_h}$ from the previous game samples at random Voter Vote Cast Return Code Generation public key $\mathtt{K}_{h,\mathtt{id}}$ and the Voter Vote Cast Return Code Generation public key $\mathtt{Kc}_{h,\mathtt{id}}$ independent from the corresponding private keys $\mathtt{k}_{h,\mathtt{id}}$ and $\mathtt{kc}_{h,\mathtt{id}}$. We have that

$$|\mathsf{Adv}^{\mathsf{sVote,mtc.2}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,mtc.3}}_{\mathcal{B}}| \leq 2 \cdot \mathsf{Adv}^{\mathbb{Q}_p,\mathsf{SGSP}}_{\mathcal{A}}$$

Where $\mathcal{B}$ is an adversary against the SGSP problem (see Definition 9.2). In the reductions, $\mathcal{B}$ from the challenge instance $(\tilde{g}, \tilde{g}^{s_0}, \ell, \ell^{s_1})$, sets the generator $g = \ell$, and $\mathtt{K}_{h,\mathtt{id}} = \ell^{s_1}$ (or $\mathtt{Kc}_{h,\mathtt{id}} = \ell^{s_1}$).

Game rMTC.5. In this game GenVerDat samples $\mathbf{C}_{\mathtt{pcc}}, \mathbf{c}_{\mathtt{ck}}$ at random and independently from values $\mathtt{hpcc}_{\mathtt{id},1}, \ldots, \mathtt{hpcc}_{\mathtt{id},n}, \mathtt{hck}_{\mathtt{id}}$.

We have that

$$|\mathsf{Adv}^{\mathsf{sVote,mtc.5}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,mtc.4}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{ElGamal,INDCPA}}_{\mathcal{B}},$$

where $\mathcal{B}$ is an adversary against the IND-CPA property of the ElGamal.

Game rMTC.6. In this game $\mathsf{GenEncLongCodeShares_h}$ from the previous game samples $\mathbf{C}_{\mathtt{expPCC},h}$ and $\mathbf{c}_{\mathtt{expCK},h}$ at random and independent from the received input $\mathbf{C}_{\mathtt{pcc}}, \mathbf{c}_{\mathtt{ck}}$, and the keys $\mathtt{k}_{\mathtt{id}}$ and $\mathtt{k}_{h,\mathtt{id}}$. The honest CCR_h also sets the values x_i and y of its internal mapping table $\mathsf{CMtable}^{\text{CCR}_h}_{\mathtt{id}}$ to random group elements. We have that

$$|\mathsf{Adv}^{\mathsf{sVote,mtc.6}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,mtc.5}}_{\mathcal{A}}| \leq 2 \cdot \mathsf{Adv}^{F,\mathsf{PRF}}_{\mathcal{B}},$$

where $\mathcal{B}$ is an adversary against the weak pseudorandom property of the exponentiation functions $F(\mathtt{k}_{h,\mathtt{id}}, r) = r^{\mathtt{k}_{h,\mathtt{id}}}$ and $F(\mathtt{kc}_{h,\mathtt{id}}, r) = r^{\mathtt{kc}_{h,\mathtt{id}}}$. Technically this game is unfolded in two: one for key $\mathtt{k}_{h,\mathtt{id}}$, and another for key $\mathtt{kc}_{h,\mathtt{id}}$; hence the 2 factor in the bound. For the reductions to go through we shall see that all queries $\mathcal{B}$ asks are uniform; indeed ciphertexts $\mathbf{C}_{\mathtt{pcc}}$,$\mathbf{c}_{\mathtt{ck}}$ are honestly generated (by the Print Office) at random, and the queries to obtain x_i and y are hashed before, so in ROM they are random too.

Now, game rMTC.6 broadcasts meaningless ciphertexts $\mathbf{C}_{\mathtt{pcc}}, \mathbf{c}_{\mathtt{ck}}$ and computes the Codes mapping table via extracting private keys of CCR_j, $\forall j \neq h$, and using the internal table for CCR_h. In other words, it does exactly what IdealSetupVoting does, hence games rMTC.6 and sMTC are identical, so $\mathsf{Adv}^{\mathsf{sVote,mtc.6}}_{\mathcal{A}} = 0$. Collecting the bounds gives the result.

18.2 Proof of Lemma 2 (vote compliance)

We use a series of game hops to bound the probability of event $\mathsf{bad_{vc}}$ in Game vc of Figure 24.

GAME vc.1. This games adds an extra check to the original game. Let $\mathtt{b}_{\mathtt{id}} = (\mathtt{vcd}_{\mathtt{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{K}_{\mathtt{id}}, \boldsymbol{\pi}_{\mathsf{ballot}})$ be the adversarial submitted ballot. Abort if event bad_1 is true, where

$$\mathsf{bad}_1 = \begin{cases} \nu = \mathsf{Dec}(\mathtt{E1}, \mathtt{EL}_{\mathtt{sk}}) \wedge w = \mathsf{Dec}(\widetilde{\mathtt{E1}}, \mathtt{EL}_{\mathtt{sk}}) \\ \nu^{\mathtt{k}_{\mathtt{id}}} \neq w \\ 1 \leftarrow \mathsf{VerifyBallotCCR}_{\mathsf{h}} \end{cases}$$

Note, that since the $\mathcal{C}$ runs $\mathsf{IdealSetupTally}$, it controls $\mathtt{EL}_{\mathtt{sk}}$ and can perform required for bad_1 check decryptions. Also note, that vc doesn't take into account tally phase, thus $\mathcal{C}$ doesn't need to know $\mathtt{EB}_{\mathtt{sk}}$.

We have that $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,vc}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,vc.1}}| \leq Pr[\mathsf{bad}_1]$. Now,

$$Pr[\mathsf{bad}_1] \leq \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{Exp,sSOUND}},$$

where $\mathcal{B}_1$ is an adversary against the simulation soundness property of the exponentiation proof system Exp. In the reduction, $\mathcal{B}_1$ uses π_{Exp} from $\boldsymbol{\pi}_{\mathsf{ballot}}$ and submits $\Big((g, \mathtt{E1}), (\mathtt{K}_{\mathtt{id}}, \widetilde{\mathtt{E1}})), \pi_{\mathsf{Exp}}\Big)$ as the challenge statement/proof pair (see Figure 6).

GAME vc.2. This games adds an extra check to the previous game. Let $\mathtt{b}_{\mathtt{id}}$ be the adversarial submitted ballot and $\mathbf{pCC}_{\mathtt{id}}$ be the vector of partial Choice Return Codes. This game aborts if event bad_2 is true, where

$$\mathsf{bad}_2 = \begin{cases} w = \mathsf{Dec}(\widetilde{\mathtt{E1}}, \mathtt{EL}_{\mathtt{sk}}) \\ w \neq \prod_{i=1}^{\psi} \mathtt{pCC}_{i,\mathtt{id}} \\ 1 \leftarrow \mathsf{VerifyBallotCCR}_{\mathsf{h}} \end{cases}$$

In other words, the adversary is restricted to submit compliant ballots (see Definition 17). We have that $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,vc.1}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,vc.2}}| \leq Pr[\mathsf{bad}_2]$. Now,

$$Pr[\mathsf{bad}_2] \leq \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{EqEnc,sSOUND}},$$

where $\mathcal{B}_2$ is an adversary against the simulation soundness property of the plaintext equality proof system EqEnc. In the reduction, $\mathcal{B}_2$ uses π_{EqEnc} from $\boldsymbol{\pi}_{\mathsf{ballot}}$ and submits $(g, \mathtt{EL}_{\mathtt{pk}}, \prod_{i=1}^{\psi} \mathtt{pk}_{\mathrm{CCR},i}, \widetilde{\mathtt{E1}}, \widetilde{\mathtt{E2}}), \pi_{\mathsf{EqEnc}})$ as the challenge statement/proof pair, where $\mathtt{E2} = (c_{2,0}, c_{2,1}, \ldots, c_{2,\psi})$, and $\widetilde{\mathtt{E2}} = \big(c_{2,0}, \prod_{i=1}^{\psi} c_{2,i}\big)$ and $\mathtt{pk}_{\mathrm{CCR},i}$ is the i-th global public key of the control components.

GAME vc.3. This game adds an extra check to the previous game. Let $(\mathbf{pCC}_{\mathtt{id}}, (\mathbf{lCC}_{j,\mathtt{id}}, \pi_{\mathsf{exp},j})_{j \neq h}))$ the partial Choice Codes and long Choice Code shares submitted by the adversary. This game aborts if event bad_3 is true, where

$$\mathsf{bad}_3 = \begin{cases} \exists \mathtt{pCC}_{i,\mathtt{id}}, \mathtt{lCC}_{j,i,\mathtt{id}} \text{ s.t. } \mathtt{lCC}_{j,i,\mathtt{id}} \neq (\mathsf{H}(\mathtt{pCC}_{i,\mathtt{id}}))^2)^{\mathtt{k}_{j,\mathtt{id}}} \\ 1 \leftarrow \mathsf{VerifyVotingPhase} \end{cases}$$

We have that $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,vc.2}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,vc.3}}| \leq Pr[\mathsf{bad}_3]$. Now,

$$Pr[\mathsf{bad}_3] \leq \mathsf{Adv}_{\mathcal{B}_3}^{\mathsf{Exp,sSOUND}},$$

where $\mathcal{B}_3$ is an adversary against the simulation soundness property of the exponentiation proof system Exp. In the reduction, $\mathcal{B}_3$ submits $\left(((g, \mathsf{H}(\mathtt{pCC}_{i,\mathtt{id}}))^2), (\mathtt{K}_{j,\mathtt{id}}, \mathtt{lCC}_{j,i,\mathtt{id}}), \pi^{(j)}_{\mathsf{exp},i}\right)$.

It remains to see that $\mathsf{Adv}^{\mathsf{sVote,vc.3}}_{\mathcal{A}} = 0$. We show that $\mathcal{A}$ has no other chance but submit a compliant registered ballot.

Case 1. $\mathcal{A}$ submits vector $\mathbf{pCC}_{\mathtt{id}}$ with some $\mathtt{pCC}_{i,\mathtt{id}} \neq v_{i'}^{\mathtt{k}_{\mathtt{id}}}$ for all voting options $v_{i'}$. Then since the Codes mapping table contains only correct entries (it is generated with the ideal setup protocol), at least one short Choice Return Code is not extractable (using that $\mathcal{A}$ exponentiates correctly to generate $\mathbf{lCC}_{j,\mathtt{id}}$), but the ballot is marked as sent and extractable in the ballot box, therefore $\mathsf{VerifyVotingPhase}$ outputs 0.

Case 2. $\mathcal{A}$ submits valid $\mathbf{pCC}_{\mathtt{id}} = (v_{j_1}^{\mathtt{k}_{\mathtt{id}}}, \ldots, v_{j_\psi}^{\mathtt{k}_{\mathtt{id}}})$. In this case, $\prod_{i=1}^{\psi} \mathtt{pCC}_{i,\mathtt{id}} = \prod_{i=1}^{\psi} v_{j_i}^{\mathtt{k}_{\mathtt{id}}}$ Since the ballot is compliant it follows that $\mathsf{Dec}(\widetilde{\mathtt{E1}}) = \prod_{i=1}^{\psi} \mathtt{pCC}_{i,\mathtt{id}} = \prod_{i=1}^{\psi} v_{j_i}^{\mathtt{k}_{\mathtt{id}}} = \nu^{\mathtt{k}_{\mathtt{id}}} = \mathsf{Dec}(\mathtt{E1}, \mathtt{EL}_{\mathtt{sk}})^{\mathtt{k}_{\mathtt{id}}}$. Last, using $\mathbb{Q}_p$ has prime order q, $\mathtt{k}_{\mathtt{id}}$ has a (multiplicative) inverse k^{-1} in $\mathbb{Z}_q^*$, so $\nu = \prod_{i=1}^{\psi} \mathtt{pCC}_{i,\mathtt{id}}^{k^{-1}} = \prod_{i=1}^{\psi} v_{j_i}$.

18.3 Proof of Theorem 1 (sent as intended)

We use five hops to bound the probability of event $\mathsf{bad}_{\mathsf{sai}}$ in Game sai of Figure 25.

GAME sai.1. In this game, algorithm $\mathsf{CreateLCCShare}_{\mathsf{h}}$ (see section 12.2.1.7) use the canonical simulator $\mathcal{S}_{\mathsf{Exp}}$ to generate a proof for statements containing $\mathtt{K}_{h,\mathtt{id}}$. We have that

$$|\mathsf{Adv}^{\mathsf{sVote,sai}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,sai.1}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{Exp,NIZK}}_{\mathcal{B}},$$

where $\mathcal{B}$ is an adversary against the zero-knowledge property of the Exponentiation proof system Exp. In the reduction, $\mathcal{B}$ uses its own oracle to answer the queries of CCR_h and outputs what the (emulated) game sai outputs (see Figure 5).

Observe that from this game onwards, the Voter Choice Return Code Generation private key $\mathtt{k}_{h,\mathtt{id}}$ is used for exponentiation (but not as a witness in the voting phase).

GAME sai.2. We now switch to the ideal setup protocol $\mathsf{IdealSetupVoting}$ (see Figure 21), which generates a correct Codes mapping table $\mathsf{CMtable}$ with a randomized transcript.

Also, in algorithm $\mathsf{CreateLCCShare}_{\mathsf{h}}$, the honest CCR_h uses its internal table, populated during the ideal setup protocol, to answer queries. (If the adversarial $\mathtt{pCC}_{i,\mathtt{id}}$ is not in the internal table, hash and exponentiate to $\mathtt{k}_{h,\mathtt{id}}$.) We have that:

$$|\mathsf{Adv}^{\mathsf{sVote,sai.1}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,sai.2}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{sVote,mtc}}_{\mathcal{B}}.$$

We use the bound given in Lemma 1 to bound $\mathsf{Adv}^{\mathsf{sVote,mtc}}_{\mathcal{B}}$.

Observe that in the ideal setup protocol, choosing independent and random long Choice Code shares $\mathbf{lCC}_{h,\mathtt{id}}$ implies that the symmetric keys $\mathtt{skcc}_{i,\mathtt{id}}$, encrypting the short Choice Return Codes in $\mathsf{CMtable}$, are also independent from each other, randomly distributed, and not known to $\mathcal{A}$.

GAME sai.3. We now switch to the ideal setup tally protocol $\mathsf{IdealSetupTally}$ (see Figure 22), which enables challenger to control $\mathtt{EL}_{\mathtt{sk}}$.

We have that $|\mathsf{Adv}^{\mathsf{sVote,sai.2}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,sai.3}}_{\mathcal{A}}| = 0$.

Observe that in the ideal setup tally protocol, the distributions of honestly generated $\mathtt{EB}_{\mathtt{pk}}$ and the one computed as $\frac{\mathtt{EL}_{\mathtt{pk}}}{\prod_{j=1}^{m'-1}\mathtt{EL}_{\mathtt{pk},j}}$ are identical, furthermore the corresponding $\mathtt{EB}_{\mathtt{sk}}$ is never used since sai never reaches tally phase.

GAME sai.4. This game adds an extra check to the previous game. It aborts if the adversary submits a non-compliant registered ballot. More formally, the game aborts if event $\mathsf{bad}_{\mathsf{vc}}$ is true.

We have that $|\mathsf{Adv}^{\mathsf{sVote,sai.3}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,sai.4}}_{\mathcal{A}}| \leq Pr[\mathsf{bad}_{\mathsf{vc}}] = \mathsf{Adv}^{\mathsf{sVote,vc}}_{\mathcal{B}}$. Note that we can use lemma 2 to bound $\mathsf{Adv}^{\mathsf{sVote,vc}}_{\mathcal{B}}$.

An important observation is that from now on $\mathcal{A}$ is restricted to submit ballots that are compliant registered ballots (see Definition 18). To reach the winning condition it must hold:

$$\begin{aligned}\mathsf{Dec}(\mathtt{E1}) &= \prod_{i=1}^{\psi} u_{j_i} \neq \prod_{i=1}^{\psi} v_{j_i} \\ \mathtt{pCC}_{i,\mathtt{id}} &= u_{j_i}^{\mathtt{k}_{\mathtt{id}}}\end{aligned}$$

Thus, for some $i \leq \psi$ we must have $\mathtt{pCC}_{i,\mathtt{id}} \neq (v_{j_i})^{\mathtt{k}_{\mathtt{id}}}$, where v_{j_i} is the i-th voting option selected by the honest voter in this game (using that there exists an inverse $\mathtt{k}_{\mathtt{id}}^{-1}$ in $\mathbb{Z}_q^*$ if q is prime). Up to re-ordering we can assume $\mathcal{A}$ sets $i = 1$.

GAME sai.5. This game encrypts a different code $\mathtt{xCC}_{1,\mathtt{id}} \neq \mathtt{CC}_{1,\mathtt{id}}$ in algorithm $\mathsf{GenCMTable}^{\$}$. We have that:

$$|\mathsf{Adv}^{\mathsf{sVote,sai.4}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,sai.5}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{SEnc,INDCPA}}_{\mathcal{B}},$$

The reduction is straightforward because the adversary $\mathcal{A}$ never submits $\mathtt{pCC}_{j_1,\mathtt{id}}$ to CCR_h (in algorithm $\mathsf{CreateLCCShare}_{\mathsf{h}}$) corresponding the the first selection of the honest voter, so $\mathtt{lCC}_{h,1,\mathtt{id}}$ and hence the symmetric key for the first short Choice Return Code $\mathtt{skcc}_{i,\mathtt{id}}$ is not used in the previous game nor in this one.

Last, in this game the adversary is given no information about $\mathtt{CC}_{1,\mathtt{id}}$, so he can only guess a value $\mathtt{CC}^*_{1,\mathtt{id}}$ for this code. Thus,

$$\mathsf{Adv}^{\mathsf{sVote,sai.5}}_{\mathcal{A}} \leq Pr[1 \leftarrow \mathsf{VerifyCRC}(\mathbf{v}_{\mathtt{id}}, \mathbf{cc}^*_{\mathtt{id}}, \mathbf{cc}_{\mathtt{id}})] \leq \frac{1}{|\mathcal{C}_{cc}|}.$$

Collecting the bounds of the hops concludes the proof.

18.4 Proof of Theorem 2 (recorded as confirmed - reject)

We use a series of game hops to bound the probability of event $\mathsf{bad}_{\mathsf{rej}}$ in Game $\mathsf{rac\text{-}rej}$ of Figure 26. The hops almost mimic those in the proof of sent as intended game, the difference is that we now argue over the Voter Vote Cast Return Code Generation keys of the honest control component CCR_h.

GAME rac-rej.1. In this game, algorithm $\mathsf{CreateLVCCShare_h}$ (see section 12.2.2.4) use the canonical simulator $\mathcal{S}_{\mathsf{Exp}}$ to generate a proof for statements containing $\mathtt{Kc}_{h,\mathtt{id}}$. We have that

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,rac\text{-}rej}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,rac\text{-}rej.1}}| \leq \mathsf{Adv}_{\mathcal{B}}^{\mathsf{Exp,NIZK}},$$

where $\mathcal{B}$ is an adversary against the zero-knowledge property of the Exponentiation proof system Exp. In the reduction, $\mathcal{B}$ uses its own oracle to answer the queries of CCR_h and outputs what the (emulated) game rac-rej outputs (see Figure 5).

Observe that from this game onwards, the Voter Vote Cast Return Code Generation private key $\mathtt{kc}_{h,\mathtt{id}}$ is used for exponentiation (but not as a witness in the voting phase).

GAME rac-rej.2. We now change to the ideal setup protocol IdealSetupVoting explained in Figure 21. Also, in algorithm $\mathsf{CreateLVCCShare_h}$, the honest CCR_h uses its internal table, populated during the ideal setup protocol, to answer queries. (If the adversarial confirmation key $\mathtt{CK_{id}}$ is not in the internal table, hash and exponentiate to $\mathtt{kc}_{h,\mathtt{id}}$.) We have:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,rac\text{-}rej.1}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,rac\text{-}rej.2}}| = \mathsf{Adv}_{\mathcal{B}}^{\mathsf{sVote,mtc}}.$$

We use the bound given in Lemma 1 to bound $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{sVote,mtc}}$.

We observe that in the ideal setup protocol, choosing long Vote Cast Return Code share $\mathtt{lVCC}_{\mathtt{id},h}$ independent from the keys $\mathtt{k_{id}}$, $\mathtt{kc}_{h,\mathtt{id}}$, and at random implies that the symmetric key $\mathtt{skvcc_{id}}$, encrypting the short Vote Cast Return Code in CMtable, is also randomly distributed, independent from the above keys, and not known to $\mathcal{A}$.

GAME rac-rej.3. This game adds an extra check to the previous game. It aborts if the following event is true.

$$\mathsf{bad}_3 = \begin{cases} \exists j \text{ s.t. } \mathtt{lVCC}_{\mathtt{id},j} \neq (\mathsf{H}(\mathtt{CK_{id}})^2)^{\mathtt{kc}_{j,\mathtt{id}}} \\ 1 \leftarrow \mathsf{VerifyVotingPhase} \end{cases}$$

We have that $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,rac\text{-}rej.2}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,rac\text{-}rej.3}}| \leq Pr[\mathsf{bad}_3]$. Now,

$$Pr[\mathsf{bad}_3] \leq \mathsf{Adv}_{\mathcal{B}}^{\mathsf{Exp,sSOUND}},$$

where $\mathcal{B}$ is an adversary against the simulation soundness property of the exponentiation proof system Exp. In the reduction, $\mathcal{B}$ submits $\Big((g, (\mathsf{H}(\mathtt{CK_{id}}))^2), (\mathtt{Kc}_{j,\mathtt{id}}, \mathtt{lVCC}_{\mathtt{id},j})), \pi_{\mathsf{expCK},j}\Big)$ as the challenge statement/proof pair, with valid proof $\pi_{\mathsf{expCK},j}$.

GAME rac-rej.4. This game adds an extra check to the previous game. It aborts if the following event is true.

$$\mathsf{bad}_4 = \begin{cases} \mathcal{A} \text{ submits the right Confirmation Key } \mathtt{CK_{id}} = \mathsf{H}(\mathtt{BCK_{id}^{k_{id}}})^2 \text{ to } \mathrm{CCR}_h \\ 1 \leftarrow \mathsf{VerifyVotingPhase} \end{cases}$$

We have that $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,rac\text{-}rej.3}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,rac\text{-}rej.4}}| \leq Pr[\mathsf{bad}_4]$. Now,

$$Pr[\mathsf{bad}_4] = 0,$$

because since the right Confirmation Key is submitted the auditors *can extract* something from the Codes mapping table, however the ballot is marked as non-extractable; the auditors can extract because in this game $\mathcal{A}$ is generating the long Vote Cast Return Code shares $(\mathtt{lVCC}_{\mathtt{id},j})_{j\neq h}$ correctly and the Codes mapping table is correct (generated with the ideal setup) .

GAME rac-rej.5. This game encrypts a different code $\mathtt{xVCC}_{\mathtt{id}} \neq \mathtt{VCC}_{\mathtt{id}}$ in algorithm $\mathsf{GenCMTable}^{\$}$. We have that:

$$|\mathsf{Adv}^{\mathsf{sVote,rac\text{-}rej.4}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,rac\text{-}rej.5}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{SEnc,INDCPA}}_{\mathcal{B}},$$

The reduction is straightforward because the adversary $\mathcal{A}$ never submits the right Confirmation Key $\mathtt{CK}_{\mathtt{id}} = \mathsf{H}(\mathtt{BCK}^{\mathtt{k}_{\mathtt{id}}}_{\mathtt{id}})^2$ to CCR_h (in algorithm $\mathsf{CreateLCCShare}_{\mathsf{h}}$), hence the symmetric key is not needed to emulate games rac-rej.4 and rac-rej.5.

Last, in this game the adversary is given no information about $\mathtt{VCC}_{\mathtt{id}}$, so he can only guess a value $\mathtt{VCC}^{*}_{\mathtt{id}}$ for this code. Thus,

$$\mathsf{Adv}^{\mathsf{sVote,rac\text{-}rej.}}_{\mathcal{A}} \leq Pr[1 \leftarrow \mathsf{VerifyVCC}(\mathtt{VCC}^{*}_{\mathtt{id}}, \mathtt{VCC}_{\mathtt{id}})] \leq \frac{1}{|\mathcal{C}_{vcc}|}.$$

Collecting the bounds of the hops concludes the proof.

18.5 Proof of Theorem 3 (recorded as confirmed - inject)

We use two hops to bound the probability of event $\mathsf{bad}_{\mathsf{rej}}$ in Game rac-inj of Figure 27.

GAME rac-inj.1. In this game, algorithm algorithm $\mathsf{CreateLVCCShare}_{\mathsf{h}}$ (see section 12.2.2.4) use the canonical simulator $\mathcal{S}_{\mathsf{Exp}}$ to generate a proof for statements containing $\mathtt{Kc}_{h,\mathtt{id}}$. We have that

$$|\mathsf{Adv}^{\mathsf{sVote,rac\text{-}inj}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,rac\text{-}inj.1}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{Exp,NIZK}}_{\mathcal{B}},$$

where $\mathcal{B}$ is an adversary against the zero-knowledge property of the Exponentiation proof system Exp. In the reduction, $\mathcal{B}$ uses its own oracle to answer the queries of CCR_h and outputs what the (emulated) game rac-inj outputs (see Figure 5).

Observe that from this game onwards, the Voter Vote Cast Return Code Generation private key $\mathtt{kc}_{h,\mathtt{id}}$ is used for exponentiation (but not as a witness in the voting phase).

GAME rac-inj.2. We now changing the SetupVoting protocol to the ideal setup protocol IdealSetupVoting explained in Figure 21. Also, in algorithm $\mathsf{CreateLVCCShare}_{\mathsf{h}}$, the honest CCR_h uses its internal table, populated during the ideal setup protocol, to answer queries. (If the adversarial confirmation key $\mathtt{CK}_{\mathtt{id}}$ is not in the internal table, hash and exponentiate to $\mathtt{kc}_{h,\mathtt{id}}$.). We have:

$$|\mathsf{Adv}^{\mathsf{sVote,rac\text{-}inj.1}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,rac\text{-}inj.2}}_{\mathcal{A}}| = \mathsf{Adv}^{\mathsf{sVote,mtc}}_{\mathcal{B}}.$$

We use the bound given in Lemma 1 to bound $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{sVote,mtc}}$.

We observe that in game rac-inj.2 choosing at random $\mathtt{lVCC}_{\mathtt{id},h}$ in the ideal setup implies that the symmetric key, encrypting the short Vote Cast Return Code in CMtable, is also random and independent from the Ballot Casting Key, and the keys $\mathtt{k}_{\mathtt{id}}$, $\mathtt{kc}_{h,\mathtt{id}}$. Also, $\mathcal{A}$ is given no information about the Confirmation Key during setup.

Let the adversarial query $\mathtt{CK}_{\mathtt{id}}^*$, unless $\mathtt{CK}_{\mathtt{id}}^* = \mathtt{CK}_{\mathtt{id}}$ (e.g. if $\mathcal{A}$ guess $\mathtt{BCK}_{\mathtt{id}}$) or $\mathsf{CreateLVCCShare}_{\mathtt{h}}$ on query $\mathtt{CK}_{\mathtt{id}}^* \neq \mathtt{CK}_{\mathtt{id}}$ answers with the same (randomly chosen) value in $\mathbb{Q}_p$ as on query $\mathtt{CK}_{\mathtt{id}}$, the ballot is non-extractable for VCC, thus the adversary looses the game. We have that:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,rac\text{-}inj.3}} \leq \frac{1}{\mathcal{C}_{bck}} + \frac{1}{q}.$$

18.6 Proof of Theorem 4 (correct tally)

We use two hops to argue that the output result L_{votes}^* of the adversary $\mathcal{A}$ in step 1 (see Figure 28) and the output of the ideal tally in step 4 are the same up to a negligible quantity.

Game correctTally.1. In this game, for all $j \leq m'$, the adversary $\mathcal{A}$ submits on behalf of the Mixing control component CCM_j ciphertext list $\mathbf{c}_{\mathsf{mix},j}$ that really are re-encryptions of the permuted CCM_j's ciphertext input list $\mathbf{c}_{\mathsf{mix},j-1}$, together with NIZK Proofs $\pi_{\mathsf{mix},j}$ of the Shuffle proof system. These are part of the output of algorithm $\mathsf{MixDecOnline}_j$, for $j < m'$, or part of the output of algorithm MixDecOffline for $j = m'$. We claim that

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,correctTally.1}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,correctTally}}| \leq \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{mix,sound}},$$

where $\mathcal{B}_1$ is an adversary against the soundness of the shuffle argument explained in Section 8, thus its advantage is negligible provided the NIZKs verify, in other words, provided $1 \leftarrow \mathsf{VerifyOnlineTally}$ and $1 \leftarrow \mathsf{VerifyOfflineTally}$. For a formal proof of this claim see the original paper [2].

Game correctTally.2. In this game, for all $j < m'$, the adversary $\mathcal{A}$ submits on behalf of the Mixing control component CCM_j ciphertext list $\mathbf{c}_{\mathsf{Dec},j}$ that really are partial decryptions (with respect the mixing private key $\mathtt{EL}_{\mathtt{sk},j}$) of the shuffled ciphertexts $\mathbf{c}_{\mathsf{mix},j}$, together with NIZK Proofs $\pi_{\mathsf{dec},j,i}$. Also $\mathcal{A}$ submits on behalf of $\mathrm{CCM}_{m'}$ a list of plaintexts $\mathbf{m}$ that really are decryptions of the shuffled ciphertexts $\mathbf{c}_{\mathsf{mix},m'}$ (with respect the secret Electoral Board private key $\mathtt{EB}_{\mathtt{sk}}$), together with NIZK proofs $\pi_{\mathsf{dec},m',i}$. These are part of the output of algorithm $\mathsf{MixDecOnline}_j$, for $j < m'$, or part of the output of algorithm MixDecOffline for $j = m'$. We have that

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,correctTally.2}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,correctTally.1}}| \leq \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{dec,sSOUND}},$$

where $\mathcal{B}_2$ is an adversary against the soundness of the decryption proof system. Here also the advantage of $\mathcal{B}_2$ is negligible provided $1 \leftarrow \mathsf{VerifyOnlineTally}$ and $1 \leftarrow \mathsf{VerifyOfflineTally}$.

Now, in this game the list of plaintexts $\mathbf{m}$ submitted by $\mathcal{A}$ correspond, up to permutation, to decryptions of ciphertexts $\texttt{E1} \in \mathsf{bb}_{\mathsf{clean}}$. Lemma 3 ensures that after applying $\mathsf{extractVoteWithAuxInfo}$ iteratively a number of $|\mathsf{bb}_{\mathsf{clean}}| \leq N$ times gives the same list with error probability upper bounded by $\mathsf{Adv}^{\mathsf{sVote,mtc}}_{\mathcal{B}_1} + N \cdot \epsilon$, where $\epsilon = 3 \cdot \mathsf{Adv}^{\mathsf{Exp,sSOUND}}_{\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{EqEnc,sSOUND}}_{\mathcal{B}_3}$. Thus, we have that

$$\mathsf{Adv}^{\mathsf{sVote,correctTally}.2}_{\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{sVote,mtc}}_{\mathcal{B}_1} + N \cdot (3 \cdot \mathsf{Adv}^{\mathsf{Exp,sSOUND}}_{\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{EqEnc,sSOUND}}_{\mathcal{B}_3}).$$

18.7 Proof of Theorem 5 (ballot privacy)

We use a series of game hops to prove that the game $\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V}}(\texttt{crs})$ ($\mathcal{C}$ uses voting options $\mathbf{v}^0_{\mathtt{id}}$ and an honest CCM is executed normally) and the game $\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V}}(\texttt{crs})$ ($\mathcal{C}$ uses voting options $\mathbf{v}^1_{\mathtt{id}}$ and an honest CCM simulates shuffle and decryption proofs) are indistinguishable. Both games are defined in Figure 29.

GAME bpriv.1. Let this be the game where $\mathcal{C}$ always uses the canonical generators $\mathcal{S}_{\mathsf{dec}}$ and $\mathcal{S}_{\mathsf{Mix}}$ in $\mathcal{O}\mathsf{HonestMix}$ to generate all decryption and shuffle proofs respectively.

To support this change, the following adjustments to the functions are made:

- Both $\mathsf{MixDecOnline}_h$ and $\mathsf{MixDecOffline}$ functions shuffle the received list of ciphertexts ($\mathsf{bb}_{\mathsf{clean}}$ and $\mathbf{c}_{\mathsf{Dec},\mathcal{A}}$ respectively) to get $\mathbf{c}_{\mathsf{mix},h}$ and then run $\mathcal{S}_{\mathsf{Mix}}$ in order to generated simulated shuffle proof $\pi_{\mathsf{mix},h}$ instead of using Mix. Also, they use $\mathcal{S}_{\mathsf{dec}}$ instead of $\mathsf{ProveDec}$ to get $\boldsymbol{\pi}_{\mathsf{dec},h}$.

We have that

$$|\mathsf{Adv}^{\mathsf{sVote,bpriv}.1}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,bpriv}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{dec,NIZK}}_{\mathcal{B}.1} + \mathsf{Adv}^{\mathsf{mix,NIZK}}_{\mathcal{B}.2},$$

where $\mathcal{B}.1$ is an adversary against the zero-knowledge property of the Decryption proof system dec and $\mathcal{B}.2$ is an adversary against the zero-knowledge property of the shuffle argument mix.

GAME bpriv.2. In this game, in $\mathcal{O}\mathsf{HonestCCRexp}$ and $\mathcal{O}\mathsf{HonestCCRconfirm}$, the $\mathcal{C}$ uses the canonical simulator $\mathcal{S}_{\mathsf{Exp}}$ to generate proofs for statements containing $\mathtt{k}_{h,\mathtt{id}}$ and $\mathtt{kc}_{h,\mathtt{id}}$.

To support this change, the following adjustments to the functions are made:

- Algorithms $\mathsf{CreateLCCShare}_{\mathsf{h}}$ and $\mathsf{CreateLVCCShare}_{\mathsf{h}}$ use $\mathcal{S}_{\mathsf{Exp}}$ to generate $\pi_{\mathsf{exp},h}$ and $\pi_{\mathsf{expCK},h}$ respectively.

We have, that

$$|\mathsf{Adv}^{\mathsf{sVote,bpriv}.2}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,bpriv}.1}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{Exp,NIZK}}_{\mathcal{B}},$$

where $\mathcal{B}$ is an adversary against the zero-knowledge property of the Exp proof system.

Observe that from this game onwards, CCR_h's private keys $\mathtt{k}_{h,\mathtt{id}}$, $\mathtt{kc}_{h,\mathtt{id}}$ are used only for exponentiation and as witnesses in the setup phase (but not as witnesses in the voting phase).

GAME bpriv.3. In this game, $\mathcal{C}$ instead of SetupVoting runs IdealSetupVoting, which generates a correct Codes mapping table CMtable with a randomized transcript. The honest CCR_h uses its internal table, populated during the ideal setup protocol, to answer queries.

We have, that

$$|\mathsf{Adv}^{\mathsf{sVote,bpriv.3}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,bpriv.2}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{sVote,mtc}}_{\mathcal{B}},$$

where $\mathcal{B}$ is an adversary against mapping table correctness property of sVote.

GAME bpriv.4. In this game, in $\mathcal{O}$CastBallot, the $\mathcal{C}$ uses canonical simulators $\mathcal{S}_{\mathsf{sch}}$, $\mathcal{S}_{\mathsf{Exp}}$ and $\mathcal{S}_{\mathsf{EqEnc}}$ to simulate all NIZK proofs in the cast ballot.

To support this change, the following adjustments to the functions are made:

- The algorithm CreateVote uses canonical simulators $\mathcal{S}_{\mathsf{sch}}$, $\mathcal{S}_{\mathsf{Exp}}$ and $\mathcal{S}_{\mathsf{EqEnc}}$ to generate π_{sch}, π_{Exp} and π_{EqEnc} respectively.

We have that

$$|\mathsf{Adv}^{\mathsf{sVote,bpriv.4}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,bpriv.3}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{sch,NIZK}}_{\mathcal{B}.1} + \mathsf{Adv}^{\mathsf{Exp,NIZK}}_{\mathcal{B}.2} + \mathsf{Adv}^{\mathsf{EqEnc,NIZK}}_{\mathcal{B}.3},$$

where $\mathcal{B}.1$, $\mathcal{B}.2$ and $\mathcal{B}.3$ are adversaries against the zero-knowledge properties of the sch, Exp and EqEnc proof systems respectively.

GAME bpriv.5. In this game the $\mathcal{C}$ does not put honest voters' Verification Card private keys $\mathtt{k}_{\mathtt{id}}$ into keystores GenCredDat and encrypt random values instead.

To support this change, the following adjustments to the functions are made:

- **For all honest voters** algorithm GenCredDat symmetrically encrypt a randomly selected value R with the keystore symmetric encryption key: $\mathtt{VCks}_{\mathtt{id}} \leftarrow \mathsf{Enc}_{\mathsf{s}}(R; \mathtt{KSkey}_{\mathtt{id}})$.

We have, that

$$|\mathsf{Adv}^{\mathsf{sVote,bpriv.5}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,bpriv.4}}_{\mathcal{A}}| \leq N_h \cdot \mathsf{Adv}^{\mathsf{SEnc,INDCPA}}_{\mathcal{B}},$$

where $\mathcal{B}$ is an adversary against IND-CPA property of symmetric encryption scheme defined in 1.

GAME bpriv.6. In this game, in $\mathcal{O}$HonestCCRexp and $\mathcal{O}$HonestCCRconfirm, the $\mathcal{C}$ outputs randomly sampled elements instead of honestly generating shares of the long Choice Return Codes $\mathbf{lCC}_{h,\mathtt{id}}$ and a share of the Voter Vote Cast Return Code $\mathtt{lVCC}_{\mathtt{id},h}$.

To support this change, the following adjustments to the functions are made:

- **For all honest voters** algorithm $\mathsf{GenCMTable}^{\$}$ (from IdealSetupVoting) uses randomly samples elements $\mathbf{lCC}_{h,\mathtt{id}}$ and $\mathtt{lVCC}_{\mathtt{id},h}$ from $\mathbb{G}$ instead of computing them as $((\mathtt{hpCC}_{1,\mathtt{id}})^{\mathtt{k}_{h,\mathtt{id}}}, \ldots, (\mathtt{hpCC}_{n,\mathtt{id}})^{\mathtt{k}_{h,\mathtt{id}}})$ and $\mathtt{hcm}_{\mathtt{id}}{}^{\mathtt{kc}_{h,\mathtt{id}}}$ respectively. The $\mathcal{C}$ keeps an internal table in order to use the same random values for honest CCR's replies as were used for CMtable. For example, if value $(\mathtt{hpCC}_{k,\mathtt{id}})^{\mathtt{k}_{h,\mathtt{id}}}$ were substituted by R_k during IdealSetupVoting and later on

(during voting phase) CCR_h receives $\texttt{pCC}_{K,\texttt{id}}$ such that $\mathsf{H}(\texttt{pCC}_{K,\texttt{id}}))^2 = \texttt{hpCC}_{k,\texttt{id}}$, then $\texttt{lCC}_{h,K,\texttt{id}}$ should be set to R_k.

- Functions $\mathsf{CreateLCCShare_h}$ and $\mathsf{CreateLVCCShare_h}$ from the game bpriv.2 take randomly sampled $\mathbf{lCC}_{h,\texttt{id}}$ and $\texttt{lVCC}_{\texttt{id},h}$ from an internal $\mathcal{C}$'s table.

We have that

$$|\mathsf{Adv}^{\mathsf{sVote,bpriv.6}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,bpriv.5}}_{\mathcal{A}}| \leq \mathsf{Adv}^{F,\mathsf{PRF}}_{\mathcal{B}}.$$

where $\mathcal{B}$ is an adversary against the weak pseudorandom property of the exponentiation functions $F(\texttt{k}_{h,\texttt{id}}, \texttt{hpCC}_{i,\texttt{id}}) = (\texttt{hpCC}_{i,\texttt{id}})^{\texttt{k}_{h,\texttt{id}}}$ and $F(\texttt{kc}_{h,\texttt{id}}, \texttt{hcm}_{\texttt{id}}) = (\texttt{hcm}_{\texttt{id}})^{\texttt{kc}_{h,\texttt{id}}}$.

Game bpriv.7. In this game, we substitute all exponentiations to $\texttt{k}_{\texttt{id}}$ by randomly sampled elements. In other words we change $(\widetilde{\texttt{E1}}, \texttt{K}_{\texttt{id}}, \texttt{CK}_{\texttt{id}}, \mathbf{pCC}_{\texttt{id}})$ that were computed as $\left(\texttt{E1}^{\texttt{k}}_{\texttt{id}}, g^{\texttt{k}}_{\texttt{id}}, (\texttt{BCK}_{\texttt{id}})^{2\cdot\texttt{k}_{\texttt{id}}}, (v^{\texttt{k}_{\texttt{id}}}_{j_1}, \ldots, v^{\texttt{k}_{\texttt{id}}}_{j_\psi})\right)$ to values randomly samples from $\mathbb{G}$.

To support this change, the following adjustments to the functions are made:

- The algorithm $\mathsf{GenVerDat}^{\$}$ (from $\mathsf{IdealSetupVoting}$) samples $\texttt{K}_{\texttt{id}}$ at random **for all honest voters**.
- **For all honest voters** algorithm $\mathsf{GenCMTable}^{\$}$ (from $\mathsf{IdealSetupVoting}$) randomly samples elements for the Confirmation Key $\texttt{CK}_{\texttt{id}}$ and the partial Choice Codes $\mathbf{pCC}_{\texttt{id}}$, instead of the elements $(\texttt{BCK}_{\texttt{id}})^{2\cdot\texttt{k}_{\texttt{id}}}$ and $(v^{\texttt{k}_{\texttt{id}}}_1, \ldots, v^{\texttt{k}_{\texttt{id}}}_n)$, respectively, to construct $\mathsf{CMtable}$. The $\mathcal{C}$ keeps an internal table with entries $(\texttt{id}, \texttt{CK}_{\texttt{id}}, \mathbf{pCC}_{\texttt{id}})$ to be consistent between the setup and voting phase.
- During vote casting ($\mathcal{O}\mathsf{CastBallot}$), GetKey is not executed, instead $\mathcal{C}$ uses its knowledge of $\texttt{k}_{\texttt{id}}$ to generate a ballot. Additionally, algorithm $\mathsf{CreateVote}$ is as in the Game bpriv.4, but samples $\widetilde{\texttt{E1}}$ at random and takes $\mathbf{pCC}_{\texttt{id}}$ from the internal table kept by the $\mathcal{C}$ (in order to be consistent with generated $\mathsf{CMtable}$).
- During vote confirmation ($\mathcal{O}\mathsf{ConfirmBallot}$), $\mathsf{CreateConfirmMessage}$ is not executed, instead the $\mathcal{C}$ outputs $\texttt{CK}_{\texttt{id}}$ from its internal table.

Observe that the only value that depends on the coin β is $\mathbf{v}^{\beta}_{\texttt{id}}$ that is encrypted with $\texttt{EL}_{\texttt{pk}}$.

We have that

$$|\mathsf{Adv}^{\mathsf{sVote,bpriv.7}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,bpriv.6}}_{\mathcal{A}}| \leq N_h \cdot \mathsf{Adv}^{\mathbb{Q}_p,\mathsf{ESGSP}}_{\mathcal{B}},$$

where $\mathcal{B}$ is an adversary against the ESGSP problem (see Definition 9.2).

Game bpriv.8. In this game, in $\mathcal{O}\mathsf{CastBallot}$, the $\mathcal{C}$ changes $\mathbf{v}^{\beta}_{\texttt{id}}$ to $\mathbf{v}^{1-\beta}_{\texttt{id}}$ for all honest voters $\texttt{id} \in N_h$.

We have that,

$$|\mathsf{Adv}^{\mathsf{sVote,bpriv.8}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{sVote,bpriv.7}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{ElGamal,INDCPA}}_{\mathcal{B}},$$

where $\mathcal{B}$ is an adversary against the IND-CPA property of the distributed ElGamal [19].

Note, that $\mathcal{A}$ doesn't know nor controls $\mathtt{EL}_{\mathtt{sk}}$ since the offline (last) CCM's private share $\mathtt{EB}_{\mathtt{sk}}$ is generated by the honest Print Officeafter all online CCMs key shares are submitted and given to the offline CCM if and only if VerifyOnlineTally is successful. Therefore in bpriv game, where only two CCMs are modeled, $\mathcal{A}$ either doesn't participate in election key pair generation at all (if it chooses to corrupt an offline CCM whose key is generated by $\mathcal{C}$ regardless) or has to pass VerifyOnlineTally in order to get the result of the offline mixing.

Observe that after this change the adversarial view that was corresponding to the case $\beta = 0$ is changed to $\beta = 1$, and the view that was corresponding to the case $\beta = 1$ is changed to $\beta = 0$. Thus, adversarial views for $\beta = 0$ and $\beta = 1$ are identical, and the adversary can win the Game bpriv.8 only by guessing the coin β at random.

Therefore,

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sVote,bpriv}} \leq\ & \mathsf{Adv}_{\mathcal{B}}^{\mathsf{dec,NIZK}} + \mathsf{Adv}_{\mathcal{B}}^{\mathsf{mix,NIZK}} + 2 \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{Exp,NIZK}} + \mathsf{Adv}_{\mathcal{B}}^{\mathsf{sVote,mtc}} + \mathsf{Adv}_{\mathcal{B}}^{\mathsf{sch,NIZK}} + \mathsf{Adv}_{\mathcal{B}}^{\mathsf{EqEnc,NIZK}} \\ & + N_h \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{SEnc,INDCPA}} + \mathsf{Adv}_{\mathcal{B}}^{F,\mathsf{PRF}} + N_h \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{sgsp}} + \mathsf{Adv}_{\mathcal{B}}^{\mathsf{ElGamal,INDCPA}}\end{aligned}$$

Part V

Parameters and further remarks

19 Choice of parameters

19.1 ElGamal encryption scheme

Parameters p, q for ElGamal are generated verifiably similarly to FIPS 186-4 [37] and g is chosen as the smallest prime number that is member of the group $\mathbb{Q}$. The bitlength of p and q are set to 2048 and 2047 bits respectively.

For the client-side operations, we use short exponents of 256 bits for performance optimization. The security of this optimization is shown in [46], [29].

19.2 Symmetric key encryption scheme

Our protocol uses AES encryption scheme in GCM mode that is defined in NIST SP 800-38D [18]. As analyzed in [32] and [5], the confidentiality (IND-CPA security) of this cryptosystem relies on the (well-accepted) assumption of modeling the AES block cipher as a keyed random function.

19.3 Key Derivation functions

As KDF algorithm we use the construction defined in ISO-18033-2 [27] and in PKCS#1v2.2 [33] with the name MGF1. Specifically we use MGF1-SHA2-256.

As PBKDF algorithm we use the PBKDF2 algorithm defined in PKCS#5v2.0 [28]. Specifically, we use the function with HMAC-SHA2-256 and 32000 iterations. The salt `KEYseed` is provided as a parameter.

19.4 Hash functions

We use SHA2-256 defined in FIPS 180-4 [39] and SHAKE-128 (for verifible parameters generation only) defined in FIPS 202 [38].

19.5 Pseudo-random functions

We use the HMAC function defined in FIPS 198-1 [36], that is composed of a SHA-256 hash function, parameterized by the symmetric key k. As detailed in [3], HMAC is a PRF whose resistance against collision is the one of the underlying hash scheme. Up to date, the collision probability of the SHA-256 hash function is considered to be negligible.

19.6 Verifiable mixnet

The verifiable re-encryption mixnet proposed by Stephanie Bayer and Jens Groth [2] requires the underlying generalized Pedersen commitment scheme to be computationally binding and to have the same prime order q as the ElGamal encryption scheme.

In order to ensure those properties, the protocol uses a verifiable group element generation scheme for generating the public key of the Pederson commitment scheme $\mathsf{ck}_{\mathsf{mix}} = (\mathbb{G}, G_1, ..., G_n, H)$. This algorithm was inspired by FIPS 186-4 (algorithm in Appendix A.2.3) [37] but includes the following modifications:

- To generate G_i for $i = (1, \ldots, n)$ it executes the steps proposed in FIPS, but uses the extendable-output function SHAKE128 instead of a hash function, and sets $index = i - 1$ and $domain\_parameter\_seed =$ commitment_seed.
- To generate H it executes the steps proposed in FIPS, but uses the extendable-output function SHAKE128 instead of a hash function, and sets $index = n$ and $domain\_parameter\_seed =$ commitment_seed.

SHAKE128 has been standarized in FIPS-202.

19.7 Return codes spaces

The return codes should be large enough for security, and small enough for usability.

Start voting keys. These codes are 20-character strings encoded in base 32. The space $\mathcal{C}_{svk}$ consists of 32^{20} values.

Ballot casting keys. These codes are 8-digit non-zero numbers. The space $\mathcal{C}_{bck}$ consists of $10^8 - 1$ values. The 8-digit number is concatenated with an extra checksum digit in EAN13 format.

Long choice and vote cast return codes. These codes are sometimes seen as ElGamal plaintexts. The space $\mathcal{C}_{lc}$ is the group $\mathbb{G}$, so there are $Ord(\mathbb{G}) = q$ different possible values.

Short choice return codes. These codes are 4-digit numbers. The space $\mathcal{C}_{cc}$ consists of 10^4 values.

Short vote cast return codes. These codes are 8-digit numbers. The space $\mathcal{C}_{vcc}$ consists of 10^8 values.

20 Abstractions

To focus our model and analyis, we discuss the abstractions made in the description of the voting system presented in Section 12.

20.1 PKI and authenticated channels

Constitution of a platform root CA, and generation of credentials for the different system contexts and tenants that wish to run an election is omitted. For more details, see [43]

Furthermore, we have abstracted away authenticated channels, i.e. we are not modeling signatures and related cryptographic keys. In the actual implementation, authenticated channels are implemented with signatures and established public key infrastructure.

20.2 Voter authentication

We have not covered details on how eligible voters authenticate in the system. This might happen via a dedicated protocol, or via a third party. In any case, the focus in this document is to show that privacy is maintained regardless of authentication procedure, so how the user authenticates is considered out of the scope. This is consistent with the current state-of-the-art in computational security proofs for e-voting systems, where eligibility verifiability [30] is very rarely analyzed.

In the actual implementation, users authenticate in the system using a challenge-response mechanism. The goal of the authentication procedure is to ensure that only a voter who proved that he opened an encrypted key store gets a valid authentication token. Namely, a voter sends a request that includes his Credentials ID to the server. The server sends back the corresponding encrypted Verification Card keystore and a challenge. The voter replies by sending a client message, which includes the server's challenge signed with the key retrieved from the encrypted keystore. If the reply to the challenge is valid, the server generates an Authentication Token containing the Voter Information, a timestamp etc. and sends it to the voter. During the voting phase, the voter includes this token with every request they send to the server. See [43, Sections 5.1, 5.2] for more details. Since authentication tokens are generated by the untrustworthy voting server, the adversary can learn all authentication tokens and can generate new ones for arbitrary voters at will. Authentication tokens do not strengthen the security of the system under the given trust assumptions, therefore they were omitted in the model

Details about the authentication layer have been deliberately omitted in the proofs for the sake of clarity, and given the fact that they are not relevant for proving cast-as-intended verifiability, universal verifiability or privacy. We emphasize that our model is independent of the way encrypted Credential Data Keystore is delivered to the voters. We actually make the adversary stronger by omitting authentication because the goal of the authentication mechanism in our protocol is to restrict access to the verification card keystores. By omitting authentication, we are granting the adversary free access to the keystores. Our only requirements are: Verification Card Keystore is generated and encrypted as described in our model and Voting Cards are delivered to the voters via a trusted channel (i.e. postal mail).

Please note that a Verification Card Keystore can only be opened by the person in possession of the voting card: the keystore is encrypted with the key that is derived from the Start Voting Key ($\texttt{SVK}_{\texttt{id}}$) printed in the voting card.

In the protocol model we assume, that all encrypted Verification Card Keystore are public. Moreover, the Attacker can open a keystore if he controls the Voting Client or corrupts a voter and access his voting card. In the event that a voter is honest and Voting Client is not leaking any information to the Attacker, it is assumed that a voter's Start Voting Key ($\texttt{SVK}_{\texttt{id}}$) that is printed in the voting card is private. Therefore, three different cases can arise:

Case 1: Non-Voter & Verifiability model (untrustworthy Voting Client). The non-voter does not reveal his Start Voting Key $\texttt{SVK}_{\texttt{id}}$ to the adversary, therefore the polynomially-bounded adversary cannot open the keystore.

Case 2: Voter & Verifiability (untrustworthy Voting Client). In that scenario, the adversary has access to the verification card private key $\texttt{k}_{\texttt{id}}$. This corruption is accounted for in the proofs of verifiability.

Case 3: Voter & Privacy (Trustworthy Voting Client). Under the assumption that the Voting Client is trustworthy, the start voting key $\texttt{SVK}_{\texttt{id}}$ is not known to the adversary, therefore it is computationally unfeasible for him to open the keystore. No information about the start voting key $\texttt{SVK}_{\texttt{id}}$ is leaked to the adversary during the authentication mechanism. Prior to the GetKey algorithm (see Section 12), the start voting key $\texttt{SVK}_{\texttt{id}}$ is used in two occasions as an input (together with a cryptographic salt) to a password-based key derivation function executed on a trustworthy (in case of privacy) Voting Client for deriving the *credential ID* and a keystore key. Only the *credential ID* is known to the untrustworthy voting server. Due to the security of the key derivation function (PBKDF2), it is not possible to infer anything about the start voting key $\texttt{SVK}_{\texttt{id}}$. Without knowledge of the Start Voting Key $\texttt{SVK}_{\texttt{id}}$, the adversary is only left with the possibility of breaking the keystore. Breaking a 128-bits symmetric encryption (to open the keystore) is computationally equivalent (or requires at least the same amount of computations) to solve the 2048-bit discrete logarithm problem. Therefore, an adversary able to get access to the contents of a keystore can also compromise the (ElGamal) key of a trusted control component. This type of adversary is not compliant with the security model of the ordinance (where at least one control component is trustworthy)

20.3 Offline mixing control component

We have abstracted away the generation and recovery of the Election key private key share $\texttt{EB}_{\texttt{sk}}$ corresponding to the last Mixing control component $\text{CCM}_{m'}$. In the actual implementation, during configuration this private key is secret-shared using a Shamir secret sharing scheme among the electoral board members, and during tally, it is reconstructed accordingly.

This detail is irrelevant in the security analysis: it only matters that the adversary learns the key $\texttt{EB}_{\texttt{sk}}$ in case the last CCM is compromised. See [43, Sections 4.5, 6.1.2, Annex 9.1.9, 9.1.10] for more details.

20.4 Write-ins

Certain elections in Switzerland give voters the possibility to write the name of a candidate in a form instead of selecting a predefined candidate (so called write-in candidates). Usually, only a small fraction of voters select write-in candidates. The protocol supports write-in candidates: using multi-recipient ElGamal encryption, the system encodes and encrypts write-in candidates as separate messages along with the pre-defined candidates (which are still encoded as the product of primes and linked to a specific choice return code using NIZK proofs). Therefore, the existence of write-in candidates does not impact the security of the protocol. However, the protocol cannot provide cast-as-intended for the write-in candidates, since it is impossible to map all possible write-in values to a choice return code. For all these reasons, the voting system description and the security analysis do not cover write-in candidates and the security guarantees regarding individual verifiability apply only to predefined voting options.

20.5 Re-login after sending or confirming the vote

Normally, the voter performs the voting process - from logging in, selecting options, sending the vote, checking the choice return codes, confirming the vote, and checking the vote cast return code - in one go. However, it might be possible that the voter aborts the session after sending the vote but before confirming it (either voluntarily or if for instance the browser crashed) and wishes to re-login to continue the process and to confirm the vote. Alternatively, the voter might re-login after a successful confirmation to check the vote cast return code a second time. The system covers both scenarios since the voting server stores the necessary information to repeat the ExtractCRC or ExtractVCC algorithm. No interaction with the Choice Return Code control components is necessary in this process. For the sake of clarity, we have omitted these alternative protocol flows in the voting system description, since they do not impact our security goals.

20.6 Validations in untrusted components

For the sake of simplicity and readability of the voting system description, we have abstracted away some validations in the voting server and the voting client. While these validations increase the robustness and usability of the system, they are irrelevant for the security analysis, since the adversary controls untrusted components and can omit those validations.

21 Conclusion

21.1 Final remarks and improvements

Observe that given the format of the choice return codes, it is important to take into account some considerations: a vote cannot contain repeated voting options, and the number of voting options inside the vote has to be fixed. In case the voter can perform a variable number of selections, the maximum will be included in the vote, filling those not selected by the voter with blank options (which are all different). This means that the voter will receive choice return codes for both the options she selected and the voting options she left blank. Finally, only one vote is allowed per voter, and this has an effect on the security of the cast-as-intended verification mechanism, as shown in the analysis. These considerations are all enforced by the protocol implementation.

21.2 Verifiability

Our analysis shows that the four attack types described in Section 13 are unlikely to occur. Security against the first three type of attacks model *cast-as-intended* and *recorded-as-cast* properties. The last type of attack models the *counted-as-recorded* property.

Regarding the first and third types of attacks, leveraging on the entropy of the output of the return codes control component allowed us to argue that the (encrypted and sorted) codes mapping table does not leak meaningful information on the Choice Return Codes corresponding to the selected voting options, nor on the Vote Cast Return Code of the voter.

Security against the second type of attack tells us that an adversary - even with the knowledge of the ballot casting key - cannot discard the vote of an honest user. Without this guarantee it is not possible to tell apart the case where the voter types incorrectly the ballot casting key from the case where the adversary tampers with it. We have not considered the case in which the adversary tampers with the start voting key because this would be detected (either the content of the encrypted ballot will not be well-formed or we are in a similar situation as for type 1 attacks).

The fourth type of attack leverages the difficulty of carrying out the first three attacks so that it can be assumed the ballot box after the voting phase is in a good state. From this, an argument for the correctness of the result is given due to the presence of *verifiable* mixnets.

We point out that the assurance stemming from our security definitions does not rule out other attacks. However, *it rules out those considered in the ordinance*, as shown in our analysis. There are several possibilites to strengthen sVote, including distributing the mapping table across control components (avoiding a centralized setup), and augmenting the entropy of the return control components for the computation of the long choice return codes. These improvements may allow moving checks from auditors to honest voters, among other advantages.

21.3 Privacy

The protocol has been proven to provide ballot privacy under the assumptions for complete verifiability given in Federal Council: voting client of an honest voter, one of the control components and the print office are trustworthy. Furthermore, it is assumed that all voters receive their credentials via an untappable channel.

It needs to be pointed out that our adversarial model does not account for trivial cases of privacy breaches such as when all honest voters vote for the same options, election authorities run the decryption procedure before results from other voting channels are obtained or if the adversary controls the user platform. We emphasize that no voting channel electronic or not can preserve the privacy of honest voters if they all chose identical options. Similarly, we are not considering as 'provisional' results known by authorities before results from other channels are acquired as it is solely a procedural issue. Also, we highlight that no system that relies on client-side encryption can satisfy ballot privacy property in cases when an attacker controls the user platform that is used for encryption the ballot as he instantly learns the voter's preferences.

Another thing worth mentioning is the importance of auditing cleansing and mixing procedures before performing the last decryption (or executing the last control component). This verification ensures the result would be checked before decryption and if any manipulation attempt detected privacy of individual votes would not be affected. It is important to stress that auditing intermediate results before executing the last CCM is the requirement only due to privacy concerns, as an attacker may attempt to send re-encryptions of a particular ballot instead of cleansing output to break the privacy of a specific voter. Even though it is arguable that ballot privacy is broken in scenarios when re-election is required, we prevent any interpretation ambiguity by demanding verification.

21.4 VEleS security objectives

We believe that our interpretation of the security objectives [16] fits with what we have proven. We review the ordinance, and in Table 6 provide a many-to-one relationship between the security objectives of the ordinance and our cryptographic properties defined in Part IV.

21.4.1 Individual verifiability

The ordinance states the following requirements for authorizations for more than 30 percent, but less than 50 percent, of the cantonal electorate.

> **VEleS Art. 4.2:** *For the purpose of individual verification, voters must receive proof that the server system has registered the vote as it was entered by the voter on the user platform as being in conformity with the system. [...]*

Security Objective	**Technical annex of VEleS**	**Our interpretation**
Sent as intended. Section 15.1.	Changing a vote before its registration by the trustworthy part of the system	The voter must receive the short Choice Return Codes that correspond to her selected voting options and that were derived by the trustworthy part of the system
Recorded as confirmed (rejection). Section 15.2.1.	Misappropriating a vote before its registration by the trustworthy part of the system	The voter must receive the short Vote Cast Code that corresponds to her Ballot Casting Key and that was derived by the trustworthy part of the system
Recorded as confirmed (injection). Section 15.2.2.	Casting a vote	It shall not be possible to cast a vote in a voter's name without knowledge of the Ballot Casting Key
Sent as intended. Section 15.1.	Changing a vote cast in conformity with the system, the casting of which has been registered by the trustworthy part of the system	The auditor must be able to check that a vote was not altered after its registration by the trustworthy part of the system
Recorded as confirmed (rejection). Section 15.2.1.	Misappropriating a vote cast in conformity with the system, the casting of which has been registered by the trustworthy part of the system	The auditor must be able to check that all votes that were correctly confirmed are included in the final tally.
Recorded as confirmed (injection). Section 15.2.2.	Inserting a vote	The auditor must be able to check that no votes that were cast were not confirmed using a valid voter's Ballot Casting key
Correct tally. Section 16.	Auditors receive proof that the result has been ascertained correctly	Correctness of tally
Privacy. Section 17.	Under the trust assumptions for complete verifiability of the protocol, the attacker is unable to breach voting secrecy or to obtain early provisional results without changing the voters or their user platforms maliciously	It shall be not possible to learn information about the votes cast beyond what is unavoidably leaked by the election results

Table 6: Identifying VEleS security objectives with our defined cryptographic properties

It follows that only the voter can detect these type of attacks. Therefore, neither the voting client nor the system should be able to forge a proof to the voter that convinces him that his vote was registered correctly by the trustworthy part of the system. Since the voter has a central role in individual verifiability, we need to formulate a number of assumptions about a honest voter's behavior.

21.4.1.1 Assumptions on honest voter's behavior

Our assumption is that the voter receives a voting card containing the following information over a trusted channel (postal mail). All codes are different for each voter and the attacker cannot learn these codes in advance from an honest voter. However, it is possible that dishonest voters collude with the attacker and divulge their codes to the attacker.

Based on an honest voter's behavior, our security objectives closely follow the technical annex of the ordinance.

> **VEleS Annex, Chapter 4.3:** *The attacker is unable to achieve the following objectives under the trust assumptions for the complete verifiability of the protocol without a voter or a trustworthy auditor being highly likely to recognise that an attack has been carried out:*
>
> - *changing a vote before its registration by the trustworthy part of the system*
> - *misappropriating a vote before its registration by the trustworthy part of the system*
> - *casting a vote*
> - *[...]*

Further extending the requirements of Art 4., Art. 5.3 states additional requirements for authorizations up to a 100 per cent of the cantonal electorate.

> **VEleS Art 5.3:** *For individual verification the following requirements must be met in addition to those in Article 4:*
>
> a. *The proof must also confirm to the voters that the data relevant to universal verification has reached the trustworthy part of the system (para. 6).*
> b. *Voters must be able to request proof after the electronic voting system is closed that the trustworthy part of the system has not already registered a vote cast using their client-sided authentication measure.*
> c. *The substantiveness of the proof must not depend on the trustworthiness of the entire server system. It may however be based on the trustworthiness of the trustworthy part of the system.*

Based on our interpretation, Art 5.3.a is covered by the security against type 1 attacks. We understand Art 5.3.b as an extension of security against type 3 attacks, for instance when the voter is not sure that

somebody used his ballot casting key in his name in case he did not want to participate in the vote. We would like to highlight that we proof under the given trust assumptions (Print office is trusted, channel print office to voter is trusted, non-voter does not divulge his codes) that the adversary cannot successfully cast a vote for a non-voter, since he does not have knowledge of his ballot casting key. Nonetheless, the cryptographic protocol allows covering the requirement from Art 5.3.b procedurally, since the trustworthy auditor verifies the correctness of the list $L_{checked}$ that was established during the cleansing process. $L_{checked}$ contains all voting card identifiers that correspond to ballots that are valid (due to ZKP verifications) and confirmed (due to the verification of signatures of the corresponding Vote Cast Code, see Section 12.3.1.1. The absence of a voting card identifier from the list $L_{checked}$ proofs that a voting card was not used to successfully cast a vote. How the information from $L_{checked}$ is made available to the voter depends on the canton and is not explicitly covered in this document. One could imagine that instead of publishing the list $L_{checked}$, the auditors verify $L_{checked}$ and make the list available to a trusted canton's representative. A voter could then contact the canton's representative and ask him to check if his voting card is present in $L_{checked}$. This is in line with the technical annex which states the following:

> **VEleS Annex. Supplementary provision 4.4.2:** *At the end the voting process, the public board displays the result and the proof that the result is correct, which has been issued in the context of universal verifiability. The voters could accordingly assume the role of "auditors" in the spirit of maximum possible transparency. Various risk considerations, connected not least with the practice orientated assumption that user platforms need to be regarded as untrustworthy, can be used as arguments that the data for the trustworthy part of the system that is relevant to verifiability should not be published without restriction. It is therefore permitted to make the data available to a restricted group of auditors*

Art 5.3.c follows naturally from the trust assumptions of universal verifiability in the technical annex.

21.4.2 Universal verifiability

In contrast to individual verifiability, universal verifiability puts the role of the verifier into the hands of an auditor. Other approaches in academia go even further and would allow everyone to assume the role of this auditor. However, as stated in the supplementary provision 4.4.2, the group of auditors can be restricted.

> **VEleS , Annex. Chapter 4.3::** *The attacker is unable to achieve the following objectives under the trust assumptions for the complete verifiability of the protocol without a voter or a trustworthy auditor being highly likely to recognise that an attack has been carried out:*
>
> - *[...]*

- *changing a vote cast in conformity with the system, the casting of which has been registered by the trustworthy part of the system*
- *misappropriating a vote cast in conformity with the system, the casting of which has been registered by the trustworthy part of the system*
- *inserting a vote*

21.4.2.1 Assumptions on auditors

We explicitly do not cover the process of selecting and designating auditors and their technical aids. However, we assume that at least one honest auditor is verifying the results using a trustworthy technical aid.

> **VEleS Chapter 4.3:** *In addition, it is assumed that at least one trustworthy auditor will review the proof with the assistance of a trustworthy technical aid.*

> **VEleS Annex: Supplementary provision 4.4.1. (On Art. 5 para. 1):** *The use of auditors serves transparency. Voters should be able to assume that auditors in the event of any doubt would point out irregularities. However the groups which people who act as auditors should be recruited from is deliberately left open.*

Of course, the result of the tally should be also correct.

> **VEleS Art. 5.4:** *For universal verification, the auditors receive proof that the result has been ascertained correctly.*

References

[1] Ben Adida and C. Andrew Neff. Ballot casting assurance. In Dan S. Wallach and Ronald L. Rivest, editors, *2006 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06, Vancouver, BC, Canada, August 1, 2006*. USENIX Association, 2006.

[2] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2012.

[3] Mihir Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. Cryptology ePrint Archive, Report 2006/043, 2006.

[4] Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. Randomness re-use in multi-recipient encryption schemeas. In *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, pages 85–99, 2003.

[5] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000.

[6] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[7] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 390–399, 2006.

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

[9] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 409–426, 2006.

[10] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *IEEE Symposium on Security and Privacy 2015*. IEEE Computer Society, 5 2015.

[11] David Bernhard, Marc Fischlin, and Bogdan Warinschi. Adaptive proofs of knowledge in the random oracle model. In *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, pages 629–649, 2015.

[12] David Bernhard, Ngoc Khanh Nguyen, and Bogdan Warinschi. Adaptive proofs have straightline extractors (in the random oracle model). In *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings*, pages 336–353, 2017.

[13] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In X. Wang and K. Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012.

[14] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 410–428, 2013.

[15] Swiss Federal Chancellery. Explications relatives à l'ordonnance de la chancellerie fédérale sur le vote électronique (OVotE). Available at `http://www.bk.admin.ch/themen/pore/evoting/07979`, 2013.

[16] Swiss Federal Chancellery. Technical and administrative requirements for electronic vote casting. Annex of the Federal Chancellery Ordinance on Electronic Voting 2.0. `https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting.html`, July 2018.

[17] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.

[18] Lily Chen. Recommendation for Key Derivation Using Pseudorandom Functions (Revised) . Special Publication 800-108, National Institute of Standards and Technology, U.S. Department of Commerce, October 2009.

[19] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Distributed ElGamal à la Pedersen: Application to helios. In Ahmad-Reza Sadeghi and Sara Foresti, editors, *WPES*, pages 131–142. ACM, 2013.

[20] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for Helios under weaker trust assumptions. In Miroslaw Kutylowski and Jaideep Vaidya, editors,

Computer Security - ESORICS 2014 Proceedings, Part II, volume 8713 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2014.

[21] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the fiat-shamir transform. In *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, pages 60–79, 2012.

[22] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[23] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[24] Jan Gerlach and Urs Gasser. Three case studies from Switzerland: E-voting, 2009.

[25] Kristian Gjosteen. The Norwegian internet voting protocol. Cryptology ePrint Archive, Report 2013/473, 2013.

[26] IETF. RFC 8018. PKCS#5: Password-Based Cryptography Specification Version 2.1, 2017.

[27] Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers. Standard, International Organization for Standardization, Geneva, CH, 2006.

[28] Burt Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898, September 2000.

[29] Takeshi Koshiba and Kaoru Kurosawa. Short exponent diffie-hellman problems. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *Public Key Cryptography – PKC 2004*, pages 173–186, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[30] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, volume 6345, pages 389–404. Springer, 2010.

[31] Ueli Maurer. Unifying zero-knowledge proofs of knowledge. In B. Preneel, editor, *Advances in Cryptology - AfricaCrypt 2009*, Lecture Notes in Computer Science. Springer-Verlag, June 2009.

[32] David A. McGrew and John Viega. Flexible and efficient message authentication in hardware and software, 2003.

[33] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, RFC Editor, November 2016.

[34] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and kdcs. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 327–346, 1999.

[35] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 111–126, 2002.

[36] NIST Computer Security Division. The Keyed-Hash Message Authentication Code (HMAC). FIPS Publication 198-1, National Institute of Standards and Technology, U.S. Department of Commerce, July 2008.

[37] NIST Computer Security Division. Digital Signature Standard (DSS). FIPS Publication 186-4, National Institute of Standards and Technology, U.S. Department of Commerce, July 2013.

[38] NIST Computer Security Division. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. FIPS Publication 202, National Institute of Standards and Technology, U.S. Department of Commerce, May 2014.

[39] NIST Computer Security Division. Secure Hash Standard (SHS). FIPS Publication 180-4, National Institute of Standards and Technology, U.S. Department of Commerce, August 2015.

[40] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[41] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pages 387–398, 1996.

[42] Phillip Rogaway. Evaluation of some blockcipher modes of operation. In *Technical report - CRYPREC*, 2011.

[43] Scytl R&S. Scytl sVote. Protocol Specifications. 2018. v13.0.

[44] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.

[45] Dominique Unruh. Post-quantum security of fiat-shamir. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 65–95, 2017.

[46] Paul C. van Oorschot and Michael J. Wiener. On diffie-hellman key agreement with short exponents. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 332–343, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[47] Hoeteck Wee. Zero knowledge in the random oracle model, revisited. In *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, pages 417–434, 2009.