Protocol of the Swiss Post Voting System

Computational Proof of Complete Verifiability and Privacy

Swiss Post*

Version 1.0.0

Abstract

The Swiss Post Voting System uses a cryptographic protocol for remote online voting that guarantees complete verifiability (individual and universal) and voting secrecy even if significant parts of the client and server-side system are under adversarial control. This report demonstrates that the protocol provides strong security guarantees under a challenging threat model. We extract precise requirements from the Federal Chancellery's Ordinance and the corresponding technical annex and model the Swiss Post Voting System. Our security analysis shows that voters or auditors can always detect the adversary's attempt to change individual votes or to interfere with the counting process—even if only one of the server-side components behaves honestly. Moreover, the server-side system cannot learn the voter's selections if one server-side component and the voting client are trustworthy.

E-Voting Community Program Material - please follow our Code of Conduct describing what you can expect from us, the Coordinated Vulnerability Disclosure Policy, and the contributing guidelines.

Revision chart

Version	Description	Author	Reviewer	Date
0.9.0	Original work by Scytl R&S for sVote 2.2.1	EL, TF, NC	JP	2020-05-27
0.9.8	Initial published version	OE	XM, JS	2021-01-19
0.9.9	See change log for version 0.9.9	OE	XM, JS	2021-05-03
0.9.10	See change log for version 0.9.10	OE	XM, JS	2021-06-25
0.9.11	See change log for version 0.9.11	OE	XM, JS	2021-10-15
0.9.12	See change log for version 0.9.12	OE	TF, FM	2022-06-24
1.0.0	See change log for version 1.0.0	OE	TF, FM	2022-07-29

Table of contents

List of figures

List of tables

Part I

Introduction and Threat Model

1 Introduction

1.1 Context

Switzerland has a longstanding tradition of direct democracy, allowing Swiss citizens to vote approximately four times a year on elections and referendums. In recent years, voter turnout hovered below 40 percent [19].

The vast majority of voters in Switzerland fill out their paper ballots at home and send them back to the municipality by postal mail, usually days or weeks ahead of the actual election date. Remote online voting (referred to as e-voting in this document) would provide voters with some advantages. First, it would guarantee the timely arrival of return envelopes at the municipality (especially for Swiss citizens living abroad). Second, it would improve accessibility for people with disabilities. Third, it would eliminate the possibility of an invalid ballot when inadvertently filling out the ballot incorrectly.

In the past, multiple cantons offered e-voting to a part of their electorate. Many voters would welcome the option to vote online - provided the e-voting system protects the integrity and privacy of their vote [21]. State-of-the-art e-voting systems alleviate the practical concerns of mail-in voting and, at the same time, provide a high level of security. Above all, they must display three properties [35]:

- Vote secrecy: do not reveal a voter's vote to anyone
- Individual verifiability: allow a voter to convince herself that the system correctly registered her vote
- Universal verifiability: allow an auditor to check that the election outcome corresponds to the registered votes

Following these principles, the Federal Chancellery defined stringent requirements for e-voting systems. The Ordinance on Electronic Voting (VEleS - Verordnung ueber die elektronische Stimmabgabe) and its technical annex (VEleS annex) [15] describes these requirements.

Swiss democracy deserves an e-voting system with excellent security properties. Swiss Post is thankful to all security researchers for their contributions and the opportunity to improve the system's security guarantees. We look forward to actively engaging with academic experts and the hacker community to maximize public scrutiny of the Swiss Post Voting System.

1.2 Two-Round Return Code Scheme

The Swiss Post Voting System is a return code scheme: every voter receives a printed code sheet prior to the election. Figure 1 shows an example code sheet, which contains the following types of codes (the codes are unique for every voter and every election event):

- A Start Voting Key to start the voting process
- A Choice Return Code for each voting option
- A Ballot Casting Key to confirm the vote
- A Vote Cast Return Code

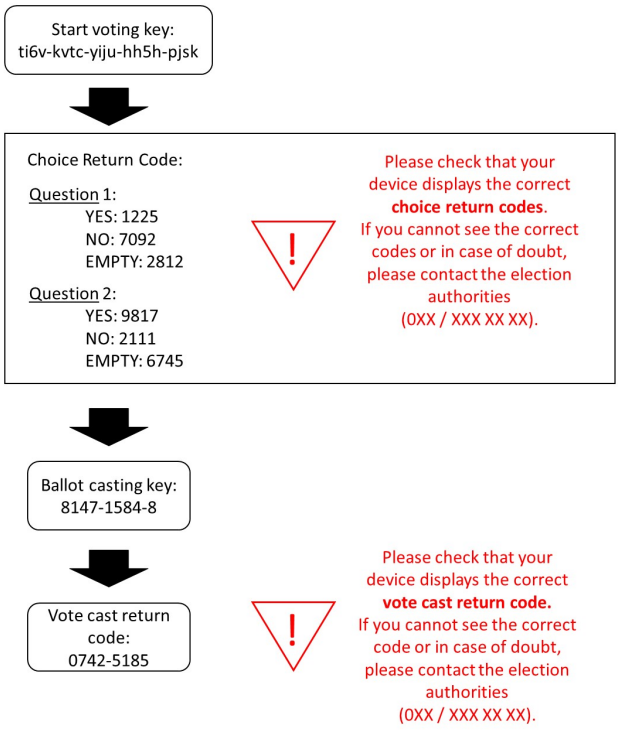

Figure 1: Printed code sheet. Codes are unique per voter and election event.

Figure 2 highlights the two rounds of the voting process from the voter's perspective.

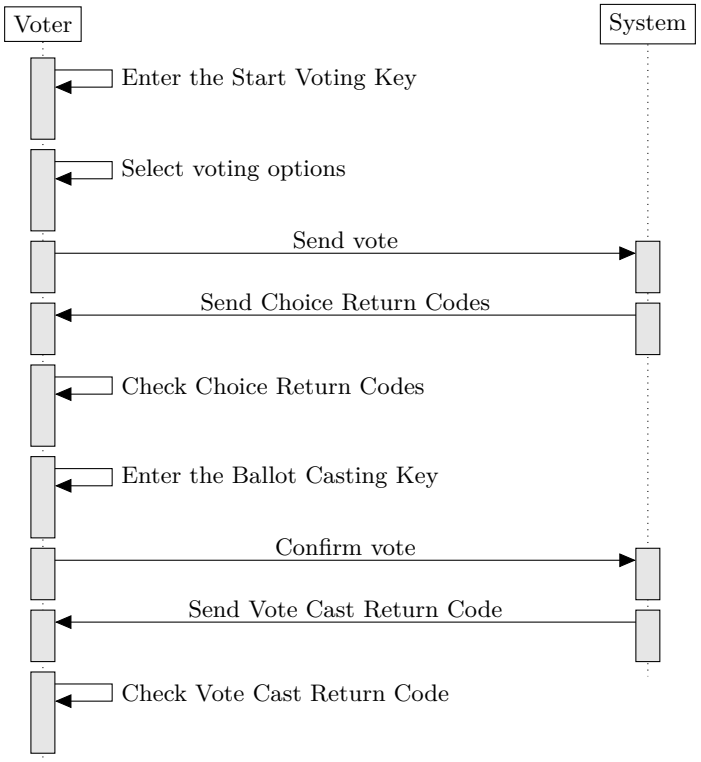

Figure 2: The voting process in a two-round return code scheme. In the implementation, an authentication step precedes the voter's voting options selection.

First, the voter enters the Start Voting Key to authenticate herself and selects the candidates she wants to vote for (or answers questions in the case of a referendum). In turn, the system responds with a Choice Return Code for each selected voting option. The voter checks that the Choice Return Codes match the ones printed on the voting card—otherwise, the voter aborts the process and alerts the election authorities. If the Choice Return Codes match, the submitted vote corresponds to the voter's intention, and the voter enters the Ballot Casting Key. Finally, the system acknowledges a successful confirmation by sending back the Vote Cast Return Code.

If the voter aborts the process after sending or confirming the vote, she can resume the process (potentially on a different voting device) by logging in again.

Figure 3 shows the process after the voter sent the vote in a previous session (but did not confirm it), while figure 4 shows the process after the voter confirmed the vote earlier on.

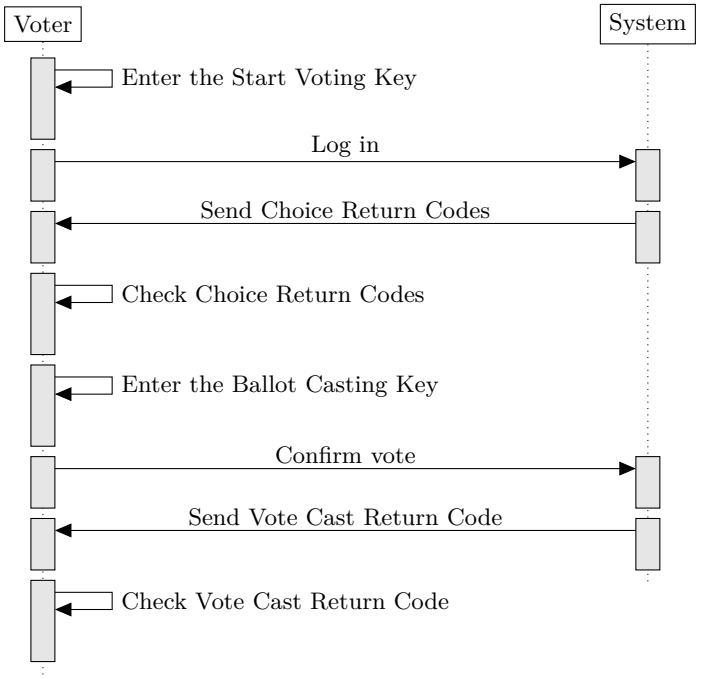

Figure 3: The voting process when the voter aborted voting in an earlier session after having sent the vote. In the earlier session, the voter did not yet confirm the vote. The Choice Return Codes returned by the system correspond to the selected voting options in the earlier session. Please note that at this stage, the voter can no longer change the selections. However, the vote is not yet considered final, and the voter might still decide to use a different voting channel.

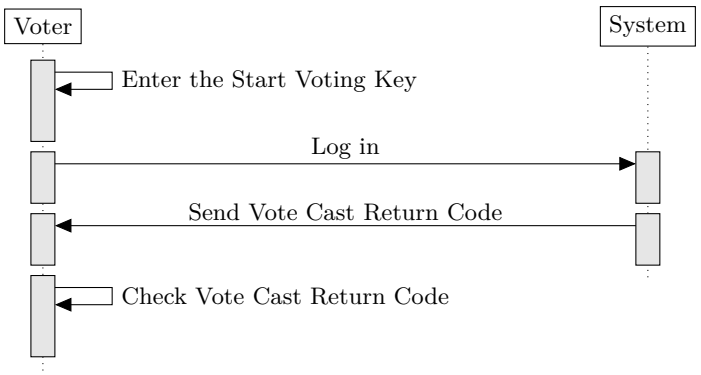

Figure 4: The voting process when the voter aborted voting in an earlier session after vote confirmation. The voter can no longer change the selections and the vote is considered final. The voter may repeat this process on multiple devices until she convinces herself that the system returns the correct Vote Cast Return Code.

2 Threat Model and Security Goals

We discuss the threat model for individual verifiability, universal verifiability and voting secrecy given by the Federal Chancellery's Ordinance. The extraction of precise assumptions and security objectives from the legislation's informally stated goals is essential for justifying that the computational proof matches the requirements. We support our interpretation of the threat model with quotes from relevant excerpts of the Federal Chancellery's Ordinance [15] and its explanatory report [16].

After discussing the assumptions on parties and communication channels, we derive the security goals related to complete verifiability and voting secrecy from the Ordinance.

2.1 Parties and Channels

2.1.1 Assumptions on Parties

The system specification explains the parties and components of the Swiss Post voting system [37]. At the same time, the Ordinance's abstract model defines the protocol participants [15]. Table 1 maps the Swiss Post voting system's parties to the ones from the Federal Chancellery's Ordinance.

Swiss Post Voting System	**Federal Chancellery's Ordinance**
Voters	Voters
Voting Client	User Device
Voting Server	Untrustworthy System
Control Components	Control Components
Setup Component	Setup Component
Printing Component	Printing Component
Tally Control Component	Untrustworthy System
Electoral Board	Untrustworthy System
Auditors	Auditors
Verifier	Auditor's technical aid

Table 1: Correspondence between the Swiss Post Voting System and the Federal Chancellery's protocol's participants.

2.1.2 Assumptions on Communication Channels

Analogously, the Federal Chancellery's Ordinance defines the parties' communication channels. Table 2 presents all possible communication channels and the corresponding trust assumptions.

Ordinance	**Trust Assumption**
Voters ↔ User Device	Trustworthy
User Device ↔ Untrustworthy System	Untrustworthy
Setup Component ↔ Untrustworthy System	Untrustworthy
Control Component ↔ Untrustworthy System	Untrustworthy
Untrustworthy System → Printing Component	Untrustworthy
Untrustworthy System → Auditor's Technical Aid	Untrustworthy
Auditor's Technical Aid ↔ Auditors	Trustworthy
Setup Component → Auditor's Technical Aid	Trustworthy
Bidirectional Channels between Control Components	Untrustworthy
Printing Component → Voter	Trustworthy

Table 2: The communication channels in the Federal Chancellery's model with the corresponding trust assumptions.

Figure 5 illustrates the communication model.

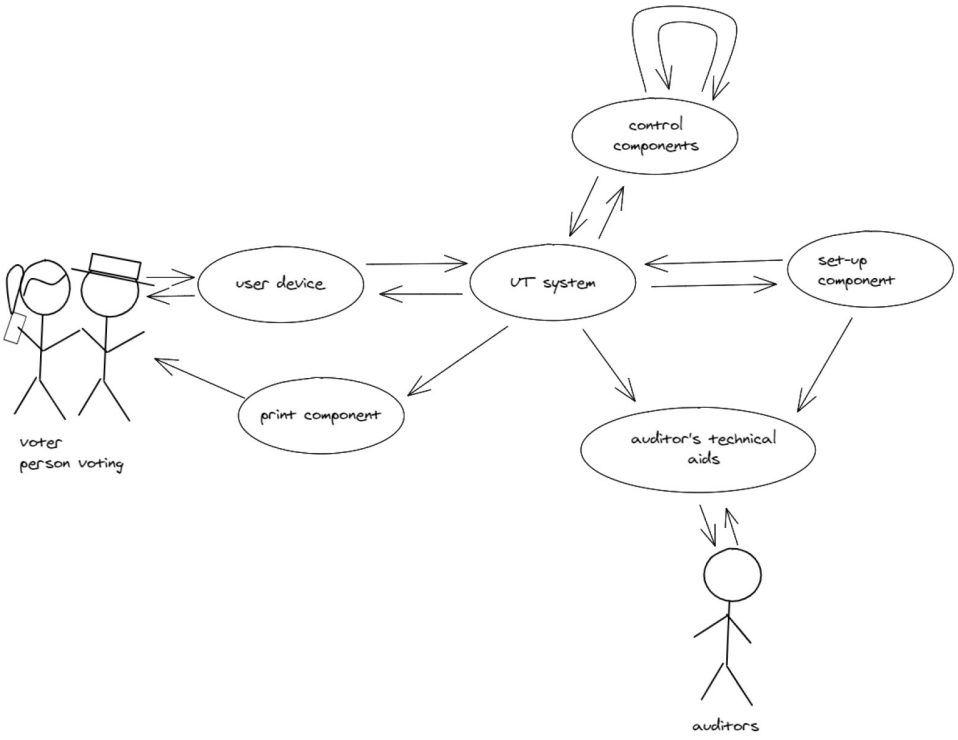

Figure 5: The Communication Model of the Protocol Participants. The image is taken from the Federal Chancellery's explanatory report [16].

2.2 Security Goals

2.2.1 Individual Verifiability

The Federal Chancellery's explanatory report [16] explains the purpose of individual verifiability:

> [Explanatory Report, Sec 4.2.1]: With individual verifiability, voters can detect any deliberate or inadvertent misuse of their voting rights. This should be possible even if the user device or the transmission path are not trustworthy. It must be assumed a priori that the user device or transmission path contains undetectable viruses or has been otherwise tampered with.

The Ordinance [15] requires the following:

> [VEleS Art 5.2]: The requirements for individual verifiability are as follows:
> a. The person voting is given the opportunity to ascertain whether the vote as entered on the user device has been manipulated or intercepted on the user device or during transmission; to this end, the person voting receives proof that the trustworthy part of the system (Art. 8) has registered the vote as it was entered by the person voting on the user device as being in conformity with the system; proof of correct registration is provided for each partial vote.

The technical annex details these requirements.

> [VEleS Annex 2.5]: The voter is given proofs in accordance with Article 5 paragraph 2 in conjunction with Article 6 letters a and b to confirm that no attacker:
> - has altered any partial vote before the vote has been registered as cast in conformity with the system;
> - has maliciously cast a vote on the voter's behalf which has subsequently been registered as a vote cast in conformity with the system and counted.

Furthermore, the Federal Chancellery's Ordinance defines an additional requirement.

> [VEleS Art 5.2]: The requirements for individual verifiability are as follows:
> b. A voter who has not cast his or her vote electronically can request proof after the electronic voting system is closed and within the statutory appeal deadlines that the trustworthy part of the system has not registered any vote cast using the client-side authentication credential of the voter.

The explanatory report clarifies that this proof of *non-participation* does *not* have to be provided by cryptographic means, but can be provided procedurally by the cantonal office.

> [Explanatory Report, Sec 4.2.1]: For reasons of efficiency, it is sufficient for the competent cantonal office to confirm to the voter that no vote has been cast on their behalf. The assumptions of trustworthiness set out in Number 2.9.1 apply to the examination by the competent body, and the auditors' technical aids may also be considered trustworthy.

Moreover, the Federal Chancellery's Ordinance introduces *effective authentication*:

> [VEleS Annex 2.8]: It must be ensured that no attacker can cast a vote in conformity with the system without having control over the voters concerned.

Table 3 highlights the trust assumptions for *individual verifiability* and *effective authentication*.

Protocol participant	**Trust Assumption**
User Device	Untrustworthy
Untrustworthy System	Untrustworthy
Control Components	One of four control components is trustworthy
Voters	Significant proportion is untrustworthy
Setup Component	Trustworthy
Printing Component	Trustworthy
Auditors	Untrustworthy
Auditor's technical aid	Untrustworthy

Table 3: Trust Assumptions for Individual Verifiability and Effective Authentication taken from VEleS Annex, 2.9.1 and VEleS Annex, 2.9.4

The Federal Chancellery's Ordinance provides concrete bounds on the acceptable probability of forging a proof for *individual verifiability*.

> [VEleS Annex, 2.11.1]: The probability of an attacker being able to falsify a proof under Number 2.5 if he changes a partial vote, suppresses a partial vote or casts a vote in someone else's name must not exceed 0.1%.

[Explanatory Report, Sec 4.2.1]: An implication of this provision is that a proof must be able to assume at least 1000 different values (for example, in the case of a numeric code, all values between 000 and 999). Thus, the probability of an attacker correctly guessing a proof would be exactly 0.1 per cent. By gathering information about the untrustworthy system participants and communication channels, he could gain an advantage so that he would not have to guess the code blindly, thus increasing the probability. With regard to such cases, a code must be able to assume sufficient values a priori so that the probability does not exceed 0.1 per cent.

2.2.2 Universal Verifiability

In contrast to individual verifiability, universal verifiability puts the role of the verifier into the hands of an auditor.

> [Explanatory Report, Sec 4.2.1]: With universal verifiability, deliberate or inadvertent manipulations (changes, additions, deletions) in the infrastructure can be detected. Unlike individual verifiability, it does not necessarily have to be offered to voters. Instead, auditors can be employed to apply universal verifiability. It must be possible to observe the auditing process. This means that the auditors should be able to understand the significance and the results of the individual steps in the voting process as far as possible. To this end, they must be able to witness that the steps in the process are correctly conducted as well as the test results, for example by going to the place of performance.

The Ordinance [15] requires the following:

> [VEleS Art 5.3]: The requirements for universal verifiability are as follows:
> a. The auditors receive proof that the result has been established correctly; the proof confirms that the result ascertained includes the following votes:
> 1. all votes cast in conformity with the system that have been registered by the trustworthy part of the system,
> 2. only votes cast in conformity with the system,
> 3. all partial votes in accordance with the proof generated in the individual verification process
> b. The auditors evaluate the proof in an observable procedure; to do so, they must use technical aids that are independent of and isolated from the rest of the system.

The technical annex details these requirements.

> [VEleS Annex 2.6]: The auditors receive a proof in accordance with Article 5 paragraph 3 letter a in conjunction with Article 6 letters a and c to confirm that no attacker:
> - after the votes were registered as cast in conformity with the system, has altered or misappropriated any partial votes before the result was determined;
> - has inserted any votes or partial votes not cast in conformity with the system which were taken into account in determining the result.

Therefore, an auditor must detect any manipulation or deletion of correctly confirmed votes (cast in conformity with the system) or notice an insertion of bogus votes while establishing the final tally.
The Federal Chancellery's Ordinance defines the combination of individual and universal verifiability as complete verifiability:

> [VEleS Art 5.1]: It must be possible to detect any manipulation that leads to a falsification of the result while preserving voting secrecy (complete verifiability). This is considered to be the case if requirements for individual and universal verifiability are met.

Table 4 highlights the trust assumptions for *universal verifiability*.

Protocol participant	**Trust Assumption**
User Device	Untrustworthy
Untrustworthy System	Untrustworthy
Control Components	One of four control components is trustworthy
Voters	Significant proportion is untrustworthy
Setup Component	Untrustworthy
Printing Component	Untrustworthy
Auditors	One auditor is trustworthy
Auditor's technical aid	One technical aid from a trustworthy auditor is trustworthy

Table 4: Trust Assumptions for Universal Verifiability taken from VEleS Annex, 2.9.2

2.2.3 Voting Secrecy

The Federal Chancellery's Ordinance requires the following for *voting secrecy*:

> [VEleS Art 7]: Preservation of voting secrecy and exclusion of premature partial results
> The trustworthiness of:
> a. the trustworthy part of the system;
> b. the procedure for generating and printing the voting papers;
> is decisive in preserving voting secrecy and excluding premature partial results within the infrastructure.

The technical annex provides a precise definition of *voting secrecy* and *premature results*:

> [VEleS Annex 1.1 and 1.2]:
> 1.1: voting secrecy is a situation in which no person or component has the following data:
> 1.1.1 votes cast or data indicating the content of votes cast,
> 1.1.2 data allowing the persons voting to be identified (data on voters), and
> 1.1.3 data allowing the data on voters to be matched with the votes cast;
> 1.2: the exclusion of premature partial results is a situation in which no person or component has the votes cast or data indicating the votes cast prematurely;

Usually, the academic literature understands voting secrecy as the adversary's inability to learn information about the cast votes beyond what is unavoidably leaked by the election results [8]. If the operational procedures ensure that the adversary cannot run the tally phase prematurely, voting secrecy implies excluding early provisional results.

Of course, this notion of voting secrecy excludes trivial privacy breaches during the tally phase, such as when all honest voters select the same voting option or when the ballot box contains only one vote. The explanatory report highlights this point:

> [Explanatory Report, Sec 4.2.1]: The fewer votes that are counted in a counting district, the greater the probability that all the votes are the same. If an attacker has access to the result of a counting district with identical votes and also manages to find out the identity of the voters, he could break the secrecy of the vote without any additional effort. He could also learn how the voters did not vote. This is the situation with both conventional and electronic voting. In line with conventional voting, the Ordinance does not regulate the minimum size of the counting districts.

We assume that election administrators minimize the likelihood of trivial privacy breaches by selecting the counting districts appropriately.

Note that if auditors detect breaches of vote secrecy only *after* the final decryption of the votes, the attacker would already have broken the voting secrecy property. The explanatory report [16] elaborates on that:

> [Explanatory Report, Sec 4.2.1]: An attacker could attempt to use the untrustworthy system participants to mark votes before they are tallied and then use the decrypted votes to breach voting secrecy. The auditors could find after tallying that the votes were not processed as they were registered, but in marked form. By this time, however, voting secrecy would already have been compromised. This must be prevented by having a group of control components ensure that no marked votes are processed before tallying.

Table 5 details the trust assumptions for *voting secrecy*.

Protocol participant	**Trust Assumption**
User Device	Trustworthy
Untrustworthy System	Untrustworthy
Control Components	One of four control components is trustworthy
Voters	Significant proportion is untrustworthy
Setup Component	Trustworthy
Printing Component	Trustworthy
Auditors	Untrustworthy
Auditor's technical aid	Untrustworthy

Table 5: Trust Assumptions for Voting Secrecy taken from VEleS Annex, 2.9.3

In contrast to verifiability, the threat model for voting secrecy assumes a *trustworthy* voting client.

> [VEleS Annex 2.7.1]: It must be ensured that no attacker is able to breach voting secrecy or establish premature partial results unless he can control the voters or their user devices.

The explanatory report justifies this:

> [Explanatory Report, Sec 4.2.1]: The ability to protect user devices from misuse is much weaker than for components in a protected environment. However, it is a conscious decision not to use the cryptographic protocol to guarantee the secrecy of the vote and the exclusion of premature partial results. This takes user-friendliness into account. However, the protocol should provide protection where votes are centrally stored. The designation of the user device as 'trustworthy' signals that no attacks on the user device need be considered in the development and analysis of the cryptographic protocol [...].

Moreover, our model assumes that voters can satisfy themselves that the client-side application works correctly and uses the correct public key by comparing their hash values with a hash value published on trustworthy sources. This assumption is in line with the Federal Chancellery's Ordinance:

> [VEleS Annex 2.7.3]: It must be ensured that no attacker can take control of user devices unnoticed by manipulating the user device software on the server. The person voting must be able to verify that the server has provided his or her user device with the correct software with the correct parameters, in particular the public key for encrypting the vote.

Furthermore, the Federal Chancellery's Ordinance has the following additional requirement.

> [VEleS Annex 2.9.3.3]: If an entire group of control components is used by a private system operator, none of these control components is considered trustworthy.

In the Swiss Post Voting System, Swiss Post operates all four online control components. Hence, for voting secrecy, all online control components must be considered *untrustworthy*. Nevertheless, the Federal Chancellery's Ordinance excludes certain type of attacks even if all online control components are corrupted:

> [VEleS Annex 2.7.2]: There is no obligation to prevent attacks that limit the number of tallied votes to the degree that all partial votes for a question, list or candidate are the same.

> [Explanatory Report, Sec 4.2.1]: In cases where Number 2.9.3.3 applies, as a result of the exclusion in Number 2.7.2, [...], it would be permissible to assume that a control component correctly registers and subsequently does not delete a sufficient number of votes from trustworthy voters as sent by the trustworthy user platform. Alternatively, it would be permissible to assume that the secrecy of the vote is not endangered if not all votes cast are tallied, but simply an arbitrary subset.

Throughout this document, we use the terms voting secrecy and vote privacy interchangeably.

Part II

Building Blocks

3 Multi-Recipient ElGamal Encryption Scheme

The ElGamal encryption scheme [20] is a public-key encryption scheme instantiated over a cyclic group $\mathbb{G}_q$ of order q with generator g, where the decisional Diffie-Hellman (DDH) problem is believed to be hard (see section 9.1).

We can share the encryption randomness when encrypting messages for multiple recipients with different public keys. Bellare et al. showed that the ElGamal encryption scheme is secure in a multi-recipient setting [2]. The multi-recipient ElGamal encryption scheme has four algorithms. We refer the reader to the crypto-primitives specification [36] for a detailed pseudo-code specification of each algorithm.

GetEncryptionParameters(`seed`) Uses as input a `seed` value and outputs `gparams` containing a description of the group $\mathbb{G}_q$, including the group order q and a group generator $g \in \mathbb{G}_q$. The Swiss Post Voting System generates the encryption parameters deterministically and verifiably. The length of the group parameters `gparams` depends on the security level λ, see Section 19.1.

GenKeyPair(`gparams`, N) Uses as input the group parameters `gparams` and the desired number of key elements $N \in \mathbb{N}^*$ and outputs N pairs of secret/public keys:

$$((\mathsf{sk}_1, \mathsf{pk}_1), \ldots, (\mathsf{sk}_N, \mathsf{pk}_N)) \in (\mathbb{Z}_q \times \mathbb{G}_q)^N$$

The i-th secret key is randomly sampled $\mathsf{sk}_i \xleftarrow{\$} \mathbb{Z}_q$, and the i-th public key is set to $\mathsf{pk}_i = g^{\mathsf{sk}_i} \in \mathbb{G}_q$.

GetCiphertext($\mathbf{m}, r, \mathbf{pk}$) Requires a vector of messages $\mathbf{m} \in \mathbb{G}_q^\ell$, a randomly sampled $r \xleftarrow{\$} \mathbb{Z}_q$, and a vector of public keys $\mathbf{pk} \in \mathbb{G}_q^k$. GetCiphertext outputs

$$\mathbf{c} = (c_0, c_1, \ldots, c_\ell) = (g^r, \mathsf{pk}_1^r \cdot m_1, \ldots, \mathsf{pk}_\ell^r \cdot m_\ell) \in \mathbb{G}_q^{\ell+1}$$

If the public key $\mathbf{pk}$ has more elements than the message $\mathbf{m}$ ($\ell < k$), GetCiphertext drops the excess public key elements $\mathsf{pk}_{\ell+1}, \ldots, \mathsf{pk}_k$. If the message $\mathbf{m}$ has more elements than the public key $\mathbf{pk}$ ($\ell > k$), GetCiphertext outputs an error.

GetMessage($\mathbf{c}, \mathbf{sk}$) Uses a ciphertext $\mathbf{c} \in \mathbb{G}_q^{\ell+1}$ and a vector of secret keys $\mathbf{sk} \in (\mathbb{Z}_q)^k$, and outputs a vector of messages $\mathbf{m} \in \mathbb{G}_q^\ell$ such that $m_i = c_i \cdot c_0^{-\mathsf{sk}_i} \in \mathbb{G}_q$. If the ciphertext $\mathbf{c}$ has more elements than the secret key $\mathbf{sk}$ ($\ell < k$), GetMessage drops the excess secret keys $\mathsf{sk}_{\ell+1}, \ldots, \mathsf{sk}_k$. Similarly to GetCiphertext, if the ciphertext $\mathbf{c}$ has more elements than the secret key $\mathbf{sk}$ ($\ell > k$), GetMessage outputs an error.

The ElGamal encryption scheme has perfect correctness: for any given pair of keys $(\mathsf{sk}, \mathsf{pk})$ generated by algorithm GenKeyPair, it holds that $\mathsf{GetMessage}(\mathsf{GetCiphertext}(m, r, \mathsf{pk}), \mathsf{sk}) = m$ for all $m \in \mathbb{G}_q$. Evidently, perfect correctness carries over to the multi-recipient setting with multiple independent key pairs. Furthermore, the multi-recipient ElGamal encryption scheme is IND-CPA secure [2]: one cannot feasibly extract any information about the message from the ciphertext.

4 Symmetric Encryption Scheme

Let $\mathcal{K}$, $\mathcal{IV}$, $\mathcal{M}$ be a key, a fixed-length initialization vector, and a message space, and let $\mathsf{E} : \mathcal{K} \times \mathcal{IV} \times \mathcal{M} \rightarrow \mathcal{M}$ be a length-preserving permutation in $\mathcal{M}$ with an inverse function D. An initialization vector-based probabilistic symmetric encryption scheme [34] consists of three algorithms:

$\mathsf{KeyGen_s}(\lambda)$ Returns a random key $K \xleftarrow{\$} \mathcal{K}(\lambda)$. The security parameter λ determines the key size.

$\mathsf{Enc_s}(M, K)$ 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 $C = IV||C'$. C' designates the core part of the ciphertext using the fixed-length initialization vector IV.

$\mathsf{Dec_s}(C, K)$ 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 the plaintext message $M = \mathsf{D}(K, IV, C')$.

The symmetric encryption scheme is correct if for each $M \in \mathcal{M}$ it holds $\mathsf{Dec_s}(K, \mathsf{Enc_s}(K, M)) = M$. Moreover, our symmetric encryption scheme is semantically (IND-CPA) secure [3].

In practice, we use a symmetric authenticated encryption scheme that combines symmetric encryption (providing *secrecy*) and message authentication codes (providing *integrity*) and for which many well-tested implementations exist. It is best practice to simultaneously ensure secrecy and integrity in the symmetric encryption setting [26]. The cryptographic primitives specification details the symmetric encryption algorithms [36].

5 Weak Pseudorandom Functions and Modular Exponentiation

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; parameterized with the public parameters pp. A function is *pseudorandom* if its outputs on *arbirary* inputs are computationally indistinguishable from the outputs of a truly random function. The function F_{pp} is a *weak* pseudorandom function if every invocation on *uniformly random inputs* $x \xleftarrow{\$} \mathcal{X}$ (instead of arbitrary inputs) returns a pseudorandom output [30].

Definition 1 (Weak Pseudorandom Function) *A function* $F_{\mathsf{pp}} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ *is* a weak pseudorandom function *if for all probabilistic polynomial-time (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 6.*

$\underline{\mathsf{rPRF}_{\mathcal{A}}^{F}(\mathsf{pp})}$:	$\underline{\mathsf{sPRF}_{\mathcal{A}}^{F}(\mathsf{pp})}$:
1. Choose at random $k \xleftarrow{\$} \mathcal{K}$	1. Choose a random function $R : \mathcal{X} \to \mathcal{Y}$
2. Choose at random $x \xleftarrow{\$} \mathcal{X}$	2. Choose at random $x \xleftarrow{\$} \mathcal{X}$
3. Give x to $\mathcal{A}$	3. Give x to $\mathcal{A}$
4. $b \leftarrow \mathcal{A}^{F_{\mathsf{pp}}(k,x)}$	4. $b \leftarrow \mathcal{A}^{R(x)}$
5. Return b	5. Return b

Figure 6: **(Left)** Real game. The adversary queries adaptively the weak pseudorandom function F_{pp}. **(Right)** Simulated game. The adversary queries adaptively the random function R.

Since this is the first time we are using this kind of definition in the document, let us elaborate. The adversary interacts multiple times with an oracle (either $F_{\mathsf{pp}}(k, x)$ or $R(x)$) and then finally has to output a guess (bit b). The adversary sees the output of each oracle equally often as it is determined randomly. We say that the definition is satisfied if the adversary cannot distinguish between two views and thus outputs the bit b independently of them.

Assume the adversary can distinguish the outputs of $F_{\mathsf{pp}}(k, x)$ from $R(x)$; then it can decide to always output 0 if playing against $F_{\mathsf{pp}}(k, x)$ and 1 if playing against $R(x)$. This would break the definition, as $\mathsf{Adv}_{\mathcal{A}}^{F,\mathsf{PRF}}(\mathsf{pp}) = 1$. However, if the adversary cannot separate the outputs, it has to output the bit b independently of the game. In such a case, the definition is satisfied, as no matter what exactly the adversary outputs (all 0s, all 1, mixed, etc.), it will simply cancel out to $\mathsf{Adv}_{\mathcal{A}}^{F,\mathsf{PRF}}(\mathsf{pp}) \approx 0$.

Modular Exponentiation as a weak pseudorandom function in the random oracle model. Naor, Pinkas, and Reingold [30] 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 decisional Diffie-Hellman assumption over $\mathbb{G}_q$. Thus, it behaves as a pseudorandom function provided the inputs come from the random distribution. In the random

oracle model, randomness on input x (arbitrarily distributed) is enforced by feeding $\mathsf{HashAndSquare}(x)$ to F. Here $\mathsf{HashAndSquare}(\cdot)$ denotes a cryptographic hash function with domain and range set to $\mathbb{G}_q$ (see section 6), and pp includes a description of $\mathbb{G}_q$. Specifically, we select $\mathbb{G}_q$ as the group of quadratic residues $\mathbb{Q}_p \subset \mathbb{Z}_p^*$ (see Section 19.1).

6 Key Derivation and Hash Functions

The Swiss Post Voting System requires two types of key derivation functions: A key derivation function KDF to obtain key material from a high-entropy master key and a password-based key derivation function Argon2id to derive key material from a relatively low-entropy password. Moreover, we require a collision-resistant hash function with an appropriate domain and range (as required, for instance, in section 5).
The KDF algorithm follows RFC5869's HKDF-Expand approach [28] and produces a random key $k' \in \mathbb{Z}_q$ and uses the following input parameters, where k is a cryptographically-strong master secret key, info is context and application-specific information, and L is the length of output keying material in octets:

$$k' \leftarrow \mathsf{KDF}(k, \mathsf{info}, L)$$

Sometimes, we require to derive a key in $\mathbb{Z}_q$; therefore, we define a function KDFToZq that returns a derived key in $\mathbb{Z}_q$ (without incurring a modulo bias) and uses as input parameter the group order q instead of an octet length L.

$$\tilde{k}' \leftarrow \mathsf{KDFToZq}(k, \mathsf{info}, q), \tilde{k}' \in \mathbb{Z}_q$$

Argon2id is a memory-hard key derivation function that makes it computationally more expensive for an attacker to brute-force the password [12]. Using Argon2id, we squeeze additional entropy from the voter's Start Voting Key $\mathtt{SVK_{id}}$ and improve the usability of the voting process (see section 19.2).

$$\mathsf{key} \leftarrow \mathsf{Argon2id}(\mathsf{password})$$

Inspired by CHVote's RecHash method [24], we instantiate our cryptographic hash function with the RecursiveHash that accepts a list of inputs $v_0, \ldots, v_{k-1}$ from different domains and guarantees the following properties.

- The probability of finding collisions is negligible.
- Different inputs result in different outputs, even across different data types (domain separation).
- Length extension attacks are prevented.

$$y \leftarrow \mathsf{RecursiveHash}(v_0, \ldots, v_{k-1})$$

Like the algorithm KDFToZq, we sometimes require the hash function to return a uniform output in $\mathbb{Z}_q$. Therefore, we define the algorithm RecursiveHashToZq that takes the group order q as an additional input parameter.

$$\tilde{y} \leftarrow \mathsf{RecursiveHashToZq}(q, (v_0, \ldots, v_{k-1})), \tilde{y} \in \mathbb{Z}_q$$

The Swiss Post Voting protocol requires a uniform output in $\mathbb{G}_q$ from an input $x \in \mathbb{N}$. Therefore, we define the algorithm HashAndSquare with a single input argument $x \in \mathbb{N}$ and the mathematical group $\mathbb{G}_q$ as a context. The algorithm HashAndSquare executes $\mathsf{RecursiveHashToZq}(q, x)$, then squares the result (modulo p).

$$\hat{y} \leftarrow \mathsf{HashAndSquare}(x), x \in \mathbb{N}, \hat{y} \in \mathbb{G}_q$$

We prove our security properties in the random oracle model (ROM), where hash functions instantiate random oracles. Therefore, we assume that an adversary cannot distinguish the output of hash functions (or key derivation functions based on cryptographic hash functions) from truly random output.
We refer the reader to the cryptographic primitives [36] and system specification [37] for implementation details and concrete choices of algorithms and parameters for the presented functions.

7 Proof Systems

Zero-knowledge proofs underpin many modern e-voting schemes; they allow protocol participants to prove a particular fact without revealing information that might compromise vote secrecy. For instance, zero-knowledge proofs might demonstrate a voter's eligibility or that a ciphertext contains a valid vote. This section formalizes the zero-knowledge proof systems, explains how we turn them non-interactive, and details the prover's and verifier's algorithms.

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. The public transcript produced by $\mathcal{P}$ and $\mathcal{V}$ when interacting on inputs st and w is denoted by $tr \leftarrow \langle \mathcal{P}(st, w), \mathcal{V}(st) \rangle$. The last part of the transcript is either accept or reject from the verifier. We write $\langle \mathcal{P}(st, w), \mathcal{V}(st) \rangle = b$, $b \in \{0, 1\}$ for rejection or acceptance.[1]

An interactive proof system is *public coin* if the verifier chooses his messages uniformly at random and independently from the messages of the prover. Bayer and Groth [1] define the following properties that a proof system must display:

Definition 2 (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:*

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

Definition 3 (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 4 (Special honest verifier zero knowledge) *The protocol $\mathsf{ps} = (\mathcal{P}, \mathcal{V})$ is* computational special honest verifier zero knowledge (SHVZK) *for relation $\mathcal{R}$ if there exists a PPT algorithm $\mathcal{S}_{\mathsf{zk}}$, that always produces accepting transcripts, 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 figure 7 defines the games rSHVZK *and* sSHVZK.

[1] We follow Bayer and Groth's slight abuse of notation here.

$\underline{\mathsf{rSHVZK}^{\mathsf{ps}}_{\mathcal{A}}}$:	$\underline{\mathsf{sSHVZK}^{\mathsf{ps}}_{\mathcal{A}}}$:
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 7: **(left)** real SHVZK game **(right)** simulated SHVZK game.

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 - the commitment - is sent by the prover. The second message e - the challenge - is sent by the verifier (chosen uniformly and independently from c). The third message z - 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*.

Definition 5 (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 (see definition 3).

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 [5], [31].

The Fiat-Shamir heuristic (FS) [18] 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 [17] we denote with $\Sigma^{\mathsf{FS}}(\mathcal{P}, \mathcal{V})$ 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 RecursiveHash (see section 6).

A series of works [9–11, 17, 33, 40] refined the complex formalization of the Fiat-Shamir transform. Correctly transforming the zero-knowledge proof to its non-interactive form $\Sigma^{\mathsf{FS}}(\mathcal{P}, \mathcal{V})$ is paramount for its security guarantees [11]. Therefore, we choose the formalization from [17] as it comes with rigorous security proofs.

7.2.1 Non-interactive Sigma Protocols in ROM

In the explicitly programmable random oracle model (EPROM) [41], all 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. We write $\mathsf{RO}[\mathcal{T}]$ to emphasize that the random oracle is programmed (by $\mathcal{S}_{\mathsf{ps}}$) in $\mathcal{T}$ different queries [41].

In the non-interactive case $\Sigma^{\mathsf{FS}}(\mathcal{P}, \mathcal{V})$ the prover $\mathcal{P}$ computes the challenge by applying a hash function to the input pair (st, c). Faust et al. [17] exploit the general forking lemma [4] in EPROM [41] to show that the non-interactive case $\Sigma^{\mathsf{FS}}(\mathcal{P}, \mathcal{V})$ retains the security from the underlying Σ-protocol.

More concretely, [17] shows that the FS transform enjoys zero-knowledge, simulation soundness and weak simulation extractability. Intuitively, simulation soundness says that even an adversary who 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 6 (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 $\mathsf{ps} = \Sigma^{\mathsf{FS}}(\mathcal{P}, \mathcal{V})$ 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 8.*

$\underline{\mathsf{rNIZK}_{\mathcal{A}}^{\mathsf{ps}}}$: 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 $b \leftarrow \mathcal{A}^{\mathsf{RO}[\mathcal{T}],\mathcal{S}_{\mathsf{ps,nizk}}(\cdot,\cdot)}$ 2. Return b
$\underline{\mathcal{S}_{\mathsf{ps,nizk}}(st, w)}$: 1. Init a programmable oracle $\mathsf{RO}[\mathcal{T}]$ and set its internal list $\mathcal{T}$ to an empty list 2. If $(st, w) \notin \mathcal{R}$ return $\perp$ 3. $\pi \leftarrow \mathcal{S}_{\mathsf{ps}}(st)$ 4. Return π	$\underline{\mathcal{S}_{\mathsf{ps}}(st)}$: 1. $\pi = (c, e, z) \leftarrow \mathcal{S}_{\mathsf{zk}}^{\mathsf{RO}}(st)$ 2.Update $\mathsf{RO}[\mathcal{T}]$ list $\mathcal{T}$ with pair $((st, c), e)$ as query/answer. If $\mathcal{T}$ is already defined in query (st, c) output $\perp$. 3. Return π

Figure 8: **(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 $\mathcal{S}_{\mathsf{ps,nizk}}$ of $\Sigma^{\mathsf{FS}}(\mathcal{P}, \mathcal{V})$. **(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 $\mathsf{RO}[\mathcal{T}]$.

For completeness, the *canonical* non-interactive zero-knowledge simulator $\mathcal{S}_{\mathsf{ps}}$ of $\mathsf{ps} = \Sigma^{\mathsf{FS}}(\mathcal{P}, \mathcal{V})$ using the zero-knowledge simulator $\mathcal{S}_{\mathsf{zk}}$ of the interactive proof system $\Sigma\langle\mathcal{P}, \mathcal{V}\rangle$ of [17] is also described in Figure 8.

Definition 7 (Simulation soundness) *The definition is a restatement of the Definition 5 of Faust et al. [17]. Let $\Sigma\langle\mathcal{P},\mathcal{V}\rangle$ be the proof system for the relation $\mathcal{R}$ with zero-knowledge simulator $\mathcal{S}_{\mathsf{ps}}$. The NIZK proof system $\Sigma^{\mathsf{FS}}(\mathcal{P},\mathcal{V})$ is* simulation sound *with respect to $\mathcal{S}_{\mathsf{ps}}$ in the random oracle model if for all PPT adversaries $\mathcal{A}$ the following 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 9.

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

1. $(st^*,\pi^*) \leftarrow \mathcal{A}^{\mathsf{RO}[\mathcal{T}](\cdot),\mathcal{S}'_{\mathsf{ps}}(\cdot)}$
2. $b \leftarrow 0$
3. Parse π^* as (c^*,e^*,z^*)
4. If $((st^*,c^*),e^*) \notin \mathcal{T} \wedge \mathcal{V}^{\mathsf{RO}[\mathcal{T}]}(st^*,\pi^*) = 1 \wedge st^* \notin \mathcal{L}_{\mathcal{R}}$ then $b \leftarrow 1$
5. Return b

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

1. $\pi = (c,e,z) \leftarrow \mathcal{S}_{\mathsf{ps}}(st)$
   Note that $\mathcal{S}_{\mathsf{ps}}$ controls $\mathsf{RO}[\mathcal{T}]$
   i.e. $((st,c),e) \in \mathcal{T}$
2. Return π

$\underline{\text{Oracle } \mathsf{RO}[\mathcal{T}](st,c)}$:

1. If (st,c) is defined in the list $\mathcal{T}$, return corresponding e.
2. Else return $\perp$

Figure 9: **(Left)** Simulation soundness game of $\Sigma^{\mathsf{FS}}(\mathcal{P},\mathcal{V})$. **(Right)** Generation of simulated proofs, for either true or false statements. The canonical simulator $\mathcal{S}_{\mathsf{ps}}$ is described in Figure 8.

Definition 8 (Weak simulation extractability) *Let the proof system $\Sigma\langle\mathcal{P},\mathcal{V}\rangle$ for relation $\mathcal{R}$. The NIZK proof system $\Sigma^{\mathsf{FS}}(\mathcal{P},\mathcal{V})$ is* weak simulation extractable *in the random oracle model with respect 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 $\mathcal{T}_{\mathsf{FS}}$, $\mathcal{T}$ made by $\mathcal{A}$ to oracles $\mathsf{RO}[\mathcal{T}]$ 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

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

Above, $\mathcal{S}'_{\mathsf{ps}}$ is the oracle given in Figure 9.

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:

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

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 9 (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* RecursiveHash *be a cryptographic hash function with the domain size κ (i.e. the hash function range is $\mathbb{Z}_\kappa$). The NIZK proof system $\Sigma_\phi^{\mathsf{FS}}(\mathcal{P}, \mathcal{V}) = (\mathsf{prove}, \mathsf{verify})$ resulting from applying the Fiat-Shamir transform is defined as follows:*

$\mathsf{prove}(st, w, \mathtt{aux})$**:** *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 $e = \mathsf{RecursiveHash}(\phi, y, c, \mathtt{aux})$ and $z = b \star w^e$*
3. *Output $\pi = (e, z)$*

$\mathsf{verify}(st, \pi, \mathtt{aux})$**:** *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 $e \in \mathbb{Z}_\kappa$ do:*

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

Note that RecursiveHash (see section 6) 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 [29] proved that the interactive version $\Sigma_\phi\langle\mathcal{P}, \mathcal{V}\rangle$, where the challenge $e \in \mathbb{Z}_\kappa$ is uniformly chosen by the verifier, is complete, SHVZK, and has special soundness. Special soundess implies that the protocol is a proof of knowledge with knowledge error $2^{-\kappa}$, where κ is the size of the challenge space. In the non-interactive case, κ is the size of the cryptographic hash function's domain that computes the challenge.

The assumptions for the above to hold[2] [29, 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. [17] 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 [29].

7.3.2 Non-Interactive Zero-Knowledge Proof Systems

Table 6 defines the Swiss Post Voting System' proof systems:

Proof System	Generation Algorithm	Verification Algorithm	Simulator
Schnorr Proof sch	GenSchnorrProof	VerifySchnorr	$\mathcal{S}_{\mathsf{sch}}$
Decryption Proof dec	GenDecryptionProof	VerifyDecryption	$\mathcal{S}_{\mathsf{dec}}$
Exponentiation Proof Exp	GenExponentiationProof	VerifyExponentiation	$\mathcal{S}_{\mathsf{Exp}}$
Plaintext Equality Proof EqEnc	GenPlaintextEqualityProof	VerifyPlaintextEquality	$\mathcal{S}_{\mathsf{EqEnc}}$

Table 6: Non-Interactive Zero-Knowledge Proof Systems of the Swiss Post Voting System.

The cryptographic primitives specification [36] elaborates on each proof system's purpose and details the generation and verification algorithm.

[2] As pointed out in [29], if ϕ is not one-way, the properties still hold though the protocol might not be useful at all.

8 Verifiable Mix Net

Verifiable mix nets underpin most modern e-voting schemes with non-trivial tallying methods since they hide the relationship between encrypted votes (potentially linked to the voter's identifier) and decrypted votes [25]. A re-encryption mix net consists of a sequence of mixers, each of which shuffles and re-encrypts an input ciphertext list $C_1, \ldots, C_N$ and returns a different ciphertext list $C'_1, \ldots, C'_N$ containing the same plaintexts. Each mixer proves knowledge of the permutation π and the randomness ρ (without revealing them to the verifier) such that $\{C'_i\}_{i=1}^N = \{C_{\pi(i)} \cdot \mathsf{GetCiphertext}(1, \rho_i, \mathsf{pk}_{\mathsf{mix}})\}_{i=1}^N$. Verifying these proofs guarantees that no mixer added, deleted, or modified a vote. The most widely used verifiable mix nets are the ones from Terelius-Wikström [39] and Bayer-Groth [1]. The Swiss Post Voting System uses the Bayer-Groth mix net, which we describe in this section.

8.1 Homomorphic Commitment Scheme

A cryptographic commitment allows a party to commit to a value (the opening), to keep the opening hidden from others, and to reveal it later [23].

We use the Pedersen commitment scheme [32] with a commitment key $\mathsf{ck}_{\mathsf{mix}} = (\mathbb{G}, G_1, \ldots, G_n, H)$, where the algorithm $\mathcal{K}$ generates random generators $G_1, \ldots, G_n, H$ of the group $\mathbb{G}$ in a verifiable manner.

Table 7 defines commitments to a *vector* of n elements $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_q^n$ or a *matrix* $A \in \mathbb{Z}_q^{n \times m}$ with columns $\mathbf{a}_1, \ldots, \mathbf{a}_m$. We can also commit to a vector with less than n elements by setting the remaining entries to 0.

Opening	Randomness	Commitment
vector $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_q^n$	$r \in \mathbb{Z}_q$	$\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}; r) = H^r \prod_{i=1}^n G_i^{a_i}$
matrix $A \in \mathbb{Z}_q^{n \times m}$	$\mathbf{r} \in \mathbb{Z}_q^N$, $N = mn$	$\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(A; \mathbf{r}) = (\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}_1; r_1), \ldots, \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}_m; r_m))$

Table 7: Pedersen commitments to vectors or matrices.

Furthermore, we follow Bayer and Groth's abuse of notation and commit to a *large vector* $\mathbf{a} \in \mathbb{Z}_q^N$, where $N = m \cdot n$, by converting the vector to a $n \times m$ matrix first (section 8.3.1 explains how we derive the dimensions m and n):
$\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}; \mathbf{r}) = (\mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(a_1, \ldots, a_n; r_1), \ldots, \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(a_{(m-1)n+1}, \ldots, a_N; r_m))$.

The Pedersen commitment scheme satisfies three properties that the Bayer-Groth mix net requires [1].

- *Perfectly hiding*: The commitment is uniformly distributed in $\mathbb{G}$.
- *Computationally binding*: It is computationally infeasible to find two different values producing the same commitment.

- *Homomorphic*: 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 random values r, s.

As an additional notation shorthand, we define the following two operations:

- $\mathbf{c^b} = (c_1, \ldots, c_m)^{(b_1,\ldots,b_m)^T} = \prod_{j=1}^{m} c_j^{b_j}$, where T transposes a matrix.
- $\mathbf{c}^B = (\mathbf{c^{b_1}}, \ldots, \mathbf{c^{b_m}})$, for matrix B with columns $\mathbf{b_1}, \ldots, \mathbf{b_m}$.

8.2 Fiat-Shamir Transform in the Mix Net

The Fiat-Shamir transform turns the interactive proof system into a non-interactive one by replacing the public coin challenges with a cryptographic hash function [1]. We discussed the security properties of the Fiat-Shamir transform in section 7.2.

We apply the Fiat-Shamir transform to each argument of the mixing proof independently:

$$x = \mathsf{RecursiveHash}(\mathsf{CRS}, \mathsf{Statement}, \mathsf{InitMessage})$$

CRS designates the Common Reference String (see section 8.3.1), followed by the proof's Statement, and the proof's initial message InitMessage. Please note that the above formula only indicates the order of the arguments: we do *not* group them into separate objects. Our approach is in line with existing non-interactive multi-round protocols [13, 27]. The hash function RecursiveHash prevents collisions: different inputs always result in different message digests.

The Hadamard Product argument requires the verifier to send two challenges x, y. Therefore, we prepend a constant to the hash input of y to ensure that x and y are independent:

$$x = \mathsf{RecursiveHash}(\mathsf{CRS}, \mathsf{Statement}, \mathsf{InitMessage})$$

$$y = \mathsf{RecursiveHash}(1, \mathsf{CRS}, \mathsf{Statement}, \mathsf{InitMessage})$$

Moreover, the Shuffle argument requires the verifier to send three challenges: x, y, z. In that case, we prepend the prover's first answer to the hash of y and an additional constant to the hash of z

$$x = \mathsf{RecursiveHash}(\mathsf{CRS}, \mathsf{Statement}, \mathsf{InitMessage})$$

$$y = \mathsf{RecursiveHash}(\mathsf{answer}, \mathsf{CRS}, \mathsf{Statement}, \mathsf{InitMessage})$$

$$z = \mathsf{RecursiveHash}(1, \mathsf{answer}, \mathsf{CRS}, \mathsf{Statement}, \mathsf{InitMessage})$$

8.3 Generating Mixing Proofs

8.3.1 Common Reference Strings and Matrix Dimensions

Bayer and Groth's original article [1] defines a Common Reference String (CRS): a string accessible to all players containing the publicly known information. The Common Reference String includes the following:

- The mixing public key $\mathtt{pk}_{\mathsf{mix}} \in \mathbb{H}$
- The commitment key $\mathtt{ck}_{\mathsf{mix}} \in \mathbb{G}$

Since the Swiss Post Voting System works with the group of quadratic residues $\mathbb{Q}_p \subset \mathbb{Z}_p^*$, the mixing public key's group $\mathbb{H}$ and the commitment key's group $\mathbb{G}$ are identical (see section 19.1). The cryptographic primitives specification [36] details the verifiable generation of the commitment key $\mathtt{ck}_{\mathsf{mix}}$ and the group parameters $\mathbb{G}$ and $\mathbb{H}$ respectively, and the system specification [37] describes how each component learns the mixing public key $\mathtt{pk}_{\mathsf{mix}} \in \mathbb{H}$.

Moreover, the Bayer-Groth mixnet requires the prover and the verifier to arrange the ciphertexts into a matrix with dimensions m and n such that $N = m \cdot n$. The cryptographic primitives specification [36] details how we select the matrix dimensions and section 8.3.9 elaborates on the special case $m = 1$.

8.3.2 $\mathsf{Mix}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C})$

This algorithm takes as input 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 with a zero-knowledge proof π_{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, the algorithm returns the shuffled and permuted ciphertexts $\mathbf{C}'$ and a proof of shuffle correctness π_{mix}.

8.3.3 $\mathsf{ShuffleArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \pi, \rho)$

This algorithm is an 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 $\mathbf{C}' = \mathsf{GetCiphertext}(\mathbf{1}, \rho, \mathsf{pk}) \cdot \mathbf{C}_\pi$. Figure 10 describes the interactive proof with the following input:

- Common Reference String: $\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}$
- Statement: $\mathbf{C}, \mathbf{C}'$

- Witness: π, ρ

In the non-interactive case, we compute the challenges as follows:

- $x \leftarrow \mathsf{RecursiveHash}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \mathsf{InitM}_{\mathsf{ShufArg}})$, where $\mathsf{InitM}_{\mathsf{ShufArg}} = \mathbf{c}_A$.
- $y \leftarrow \mathsf{RecursiveHash}(\mathsf{Answer1}_{\mathsf{ShufArg}}, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \mathsf{InitM}_{\mathsf{ShufArg}})$, where $\mathsf{InitM}_{\mathsf{ShufArg}} = \mathbf{c}_A$ and $\mathsf{Answer1}_{\mathsf{ShufArg}} = \mathbf{c}_B$[3].
- $z \leftarrow \mathsf{RecursiveHash}(1, \mathsf{Answer1}_{\mathsf{ShufArg}}, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \mathsf{InitM}_{\mathsf{ShufArg}})$, where $\mathsf{InitM}_{\mathsf{ShufArg}} = \mathbf{c}_A$ and $\mathsf{Answer1}_{\mathsf{ShufArg}} = \mathbf{c}_B$[4]
  Figure 10 highlights that $\mathsf{InitM}_{\mathsf{ShufArg}} = (\mathbf{c}_A)$ is the initial message and $\mathsf{Answer1}_{\mathsf{ShufArg}} = \mathbf{c}_B$ the first answer of the shuffle argument.

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

[3] We omit x as an input here; the values in the hash determine the value of x.
[4] We omit x and y as an input here; the values in the hash determine the values of x and y.

Prover		**Challenger**
Common Reference String: $\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}$ **Statement:** $\mathbf{C}, \mathbf{C}' \in \mathbb{H}^N$ with $N = mn$		
Witness: $\pi \in \Sigma_N$ and $\boldsymbol{\rho} \in \mathbb{Z}_q^N$		
Pick $\mathbf{r} \leftarrow \mathbb{Z}_q^m$ Set $\mathbf{a} = \{\pi(i)\}_{i=1}^N$ Compute $\mathbf{c}_A = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}; \mathbf{r})$		
	InitMessage: $\mathbf{c}_A$ →	
		$x \leftarrow \mathbb{Z}_q^*$
	← **Challenge:** x	
Pick $\mathbf{s} \in \mathbb{Z}_q^m$ Set $\mathbf{b} = \{x^{\pi(i)}\}_{i=1}^N$ Compute $\mathbf{c}_B = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{b}; \mathbf{s})$		
	1st Answer: $\mathbf{c}_B$ →	
		$y, z \leftarrow \mathbb{Z}_q^*$
	← **Challenges:** y, z	
Define $\mathbf{c}_{-z} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(-z, \ldots, -z; \mathbf{0})$ and $\mathbf{c}_D = \mathbf{c}_A^y \mathbf{c}_B$. Compute $\mathbf{d} = y\mathbf{a} + \mathbf{b}$ and $\mathbf{t} = y\mathbf{r} + \mathbf{s}$ Set $b = \prod_{i=1}^N (yi + x^i - z)$		
$\pi_{\mathsf{Prod}} \leftarrow \mathsf{ProductArg}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_D \mathbf{c}_{-z}, b, \mathbf{d} - \mathbf{z}, \mathbf{t})$ Engage in a Product Argument of opening $d_1 - z, \ldots, d_N - z$ and $\mathbf{t}$ s.t. $\mathbf{c}_D \mathbf{c}_{-z} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{d} - \mathbf{z}; \mathbf{t})$ and $\prod_{i=1}^N (d_i - z) = \prod_{i=1}^N (yi + x^i - z)$		
Compute $\rho = -\boldsymbol{\rho} \cdot \mathbf{b}$ Set $\mathbf{x} = (x, x^2, x^3, \ldots, x^N)^T$ $\pi_{\mathsf{MultiExp}} \leftarrow \mathsf{MultiExpArg}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{C}', \mathbf{C}^{\mathbf{x}}, \mathbf{c}_B, \mathbf{b}, \mathbf{s}, \rho)$ Engage in a multi-exponentiation argument of $\mathbf{b}, \mathbf{s}$ and ρ s.t. $\mathbf{C}^{\mathbf{x}} = \mathsf{GetCiphertext}(1, \rho, \mathsf{pk}_{\mathsf{mix}}) \cdot \mathbf{C}'^{\mathbf{b}}$		
	2nd Answer: $\pi_{\mathsf{Prod}}, \pi_{\mathsf{MultiExp}}$ →	

Figure 10: Shuffle argument: an argument of knowledge of a permutation π and randomness ρ s.t. $\mathbf{C}' = \mathsf{GetCiphertext}(\mathbf{1}, \rho, \mathsf{pk}_{\mathsf{mix}}) \cdot \mathbf{C}_\pi$.

8.3.4 $\mathsf{MultiExpArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, (\mathbf{C_1}, \ldots, \mathbf{C_m}), C, \mathbf{c}_A, \{\mathbf{a}_j\}_{j=1}^m, \mathbf{r}, \rho)$

This algorithm is an argument of knowledge of openings of commitments $\mathbf{c}_A$ to $A = \{\mathbf{a}_j\}_{j=1}^m$ such that $C = \mathsf{GetCiphertext}(1, \rho, \mathtt{pk}_{\mathsf{mix}}) \cdot \prod_{i=1}^m \mathbf{C}_i^{\mathbf{a}_i}$ and $\mathbf{c}_A = \mathsf{com}_{\mathtt{ck}_{\mathsf{mix}}}(A; \mathbf{r})$. Figure 11 describes an interactive proof with the following input:

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

In the non-interactive case, we compute the challenge as follows:

- $x \leftarrow \mathsf{RecursiveHash}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C_1}, \ldots, \mathbf{C_m}, C, \mathbf{c}_A, \mathsf{InitM}_{\mathsf{MultiExp}})$, where $\mathsf{InitM}_{\mathsf{MultiExp}} = (c_{A_0}, \{c_{B_k}\}_{k=0}^{2m-1}, \{E_k\}_{k=0}^{2m-1})$ is an initial message of a multi-exponentiation argument computed as described in Figure 11.

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

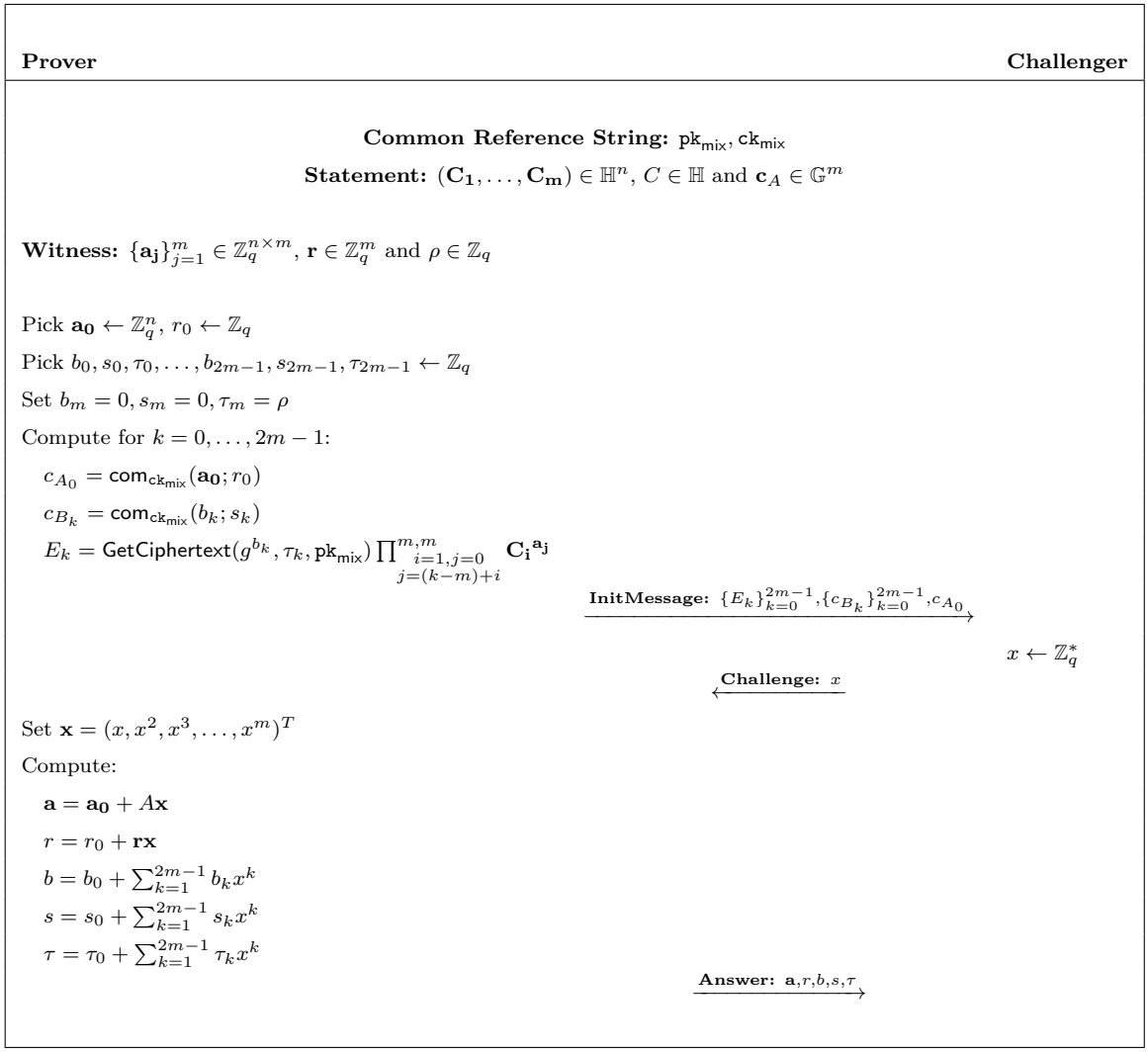

Figure 11: Multi-exponentiation argument: an argument of knowledge of openings of commitment $\mathbf{c}_A$ to $A = \{a_{ij}\}_{i,j=1}^{n,m}$ s.t. $C = \mathsf{GetCiphertext}(1, \rho, \mathsf{pk}_{\mathsf{mix}}) \prod_{i=1}^m \mathbf{C_i}^{\mathbf{a_i}}$ and $\mathbf{c}_A = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(A; \mathbf{r})$.

8.3.5 $\mathsf{ProductArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, b, A, \mathbf{r})$

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 $\prod_{i=1}^{n}\prod_{j=1}^{m} a_{ij} = b$. Figure 12 describes an interactive proof with the following input:

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

The product argument does not require a challenge.
The output is $\pi_{\mathsf{Prod}} = (c_b, \pi_{\mathsf{HadPA}}, \pi_{\mathsf{SingleVPA}})$.

8.3.6 $\mathsf{HadamardProdArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, c_b, (\mathbf{a_1}, \ldots, \mathbf{a_m}), \mathbf{r}, \mathbf{b}, s)$

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 $i = 1, \ldots, n$. Figure 13 describes an interactive proof with the following input:

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

In the non-interactive case, we compute the challenges as follows:

- $x \leftarrow \mathsf{RecursiveHash}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, c_b, \mathsf{InitM}_{\mathsf{HadPA}})$
- $y \leftarrow \mathsf{RecursiveHash}(1, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, c_b, \mathsf{InitM}_{\mathsf{HadPA}})^3$, where $\mathsf{InitM}_{\mathsf{HadPA}} = \mathbf{c}_B$.

  Figure 13 highlights that $\mathsf{InitM}_{\mathsf{HadPA}} = \mathbf{c}_B$ is the initial message of the Hadamard product argument.

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

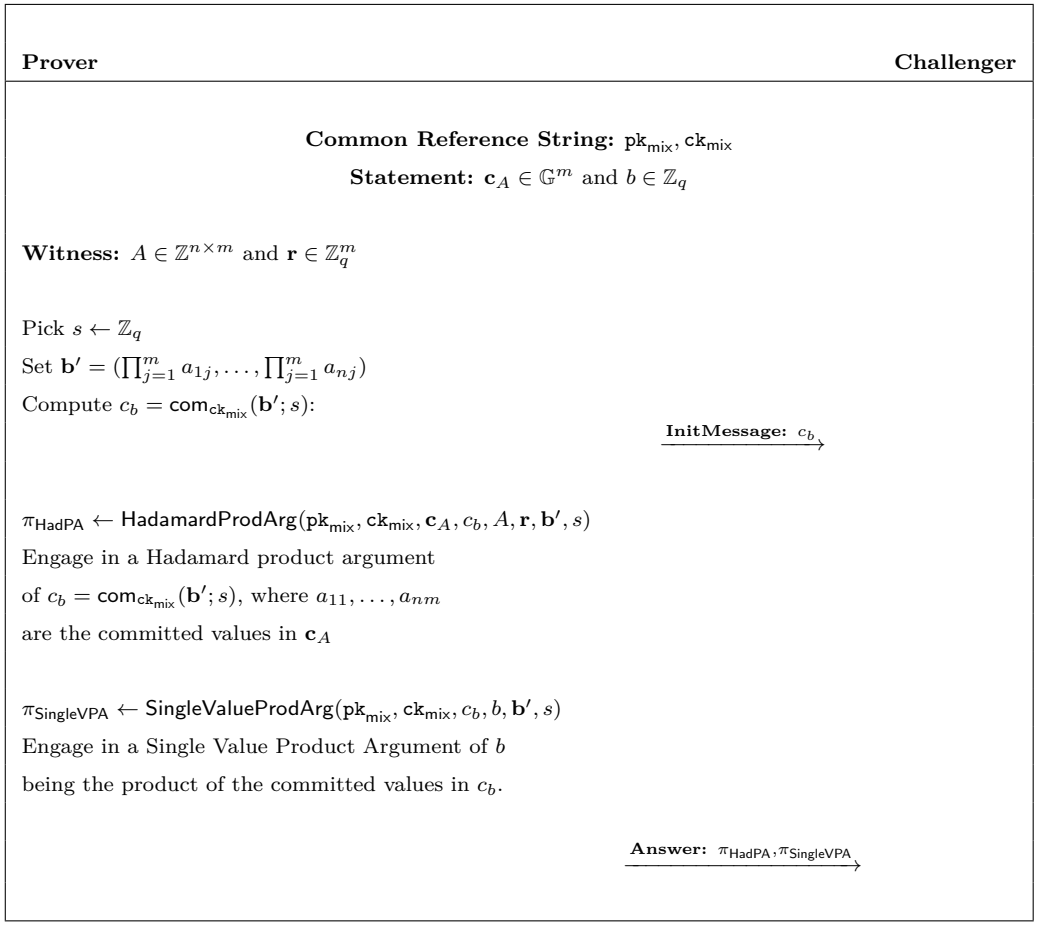

Figure 12: Product argument: an argument of knowledge of openings $a_{11}, \ldots, a_{nm}, r_1, \ldots, r_m$ to a given commitment $\mathbf{c}_A$ s.t. $\prod_{i=1}^n \prod_{j=1}^m a_{ij} = b$.

Prover		**Challenger**
Common Reference String: $\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}$ **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: $D = (\mathbf{d}_1, \ldots, \mathbf{d}_{m-1}, \mathbf{d})$ $\mathbf{t} = (t_1, \ldots, t_{m-1}, t)$ $F = (\mathbf{a}_2, \ldots, \mathbf{a}_m, -\mathbf{1})$ $\boldsymbol{\tau} = (r_2, \ldots, r_m, 0)$ $\mathbf{c}_f = (c_{A_2}, \ldots, c_{A_m}, c_{-1})$ $\mathbf{c}_d = (c_{D_1}, \ldots, c_{D_{m-1}}, c_D)$		
$\pi_{\mathsf{ZeroArg}} \leftarrow \mathsf{ZeroArg}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, \mathbf{c}_f, \mathbf{c}_d, \star, F, D, \boldsymbol{\tau}, \mathbf{t})$ 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 13: 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 $i = 1, \ldots, n$.

8.3.7 $\mathsf{ZeroArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, \mathbf{c}_B, \star, A, B, \mathbf{r}, \mathbf{s})$

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. $0 = \sum_{i=1}^{m} \mathbf{a}_i \star \mathbf{b}_{i-1}$. Figure 14 describes an interactive proof with the following input:

- Common Reference String: $\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}$
- 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: $A, B, \mathbf{r}, \mathbf{s}$

In the non-interactive case, we compute the challenges as follows:

- $x \leftarrow \mathsf{RecursiveHash}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathsf{InitM}_{\mathsf{ZeroArg}}, \mathbf{c}_B, \mathbf{c}_A)$, where $\mathsf{InitM}_{\mathsf{ZeroArg}} = (c_{A_0}, c_{B_m}, \mathbf{c}_D)$.

  Figure 14 shows that $\mathsf{InitM}_{\mathsf{ZeroArg}} = (c_{A_0}, c_{B_m}, \mathbf{c}_D)$ is the initial message of the zero argument.

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

8.3.8 $\mathsf{SingleValueProdArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, c_a, b, \mathbf{a}, r,)$

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 $b = \prod_{i=1}^{n} a_i$. Figure 15 describes an interactive proof with the following input:

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

In the non-interactive case, we compute the challenges as follows:

- $x \leftarrow \mathsf{RecursiveHash}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathsf{InitM}_{\mathsf{SingleVPA}}, b, c_a)$, where $\mathsf{InitM}_{\mathsf{SingleVPA}} = (c_\Delta, c_\delta, c_d)$

  Figure 15 shows that is the initial message of the single value product argument.

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

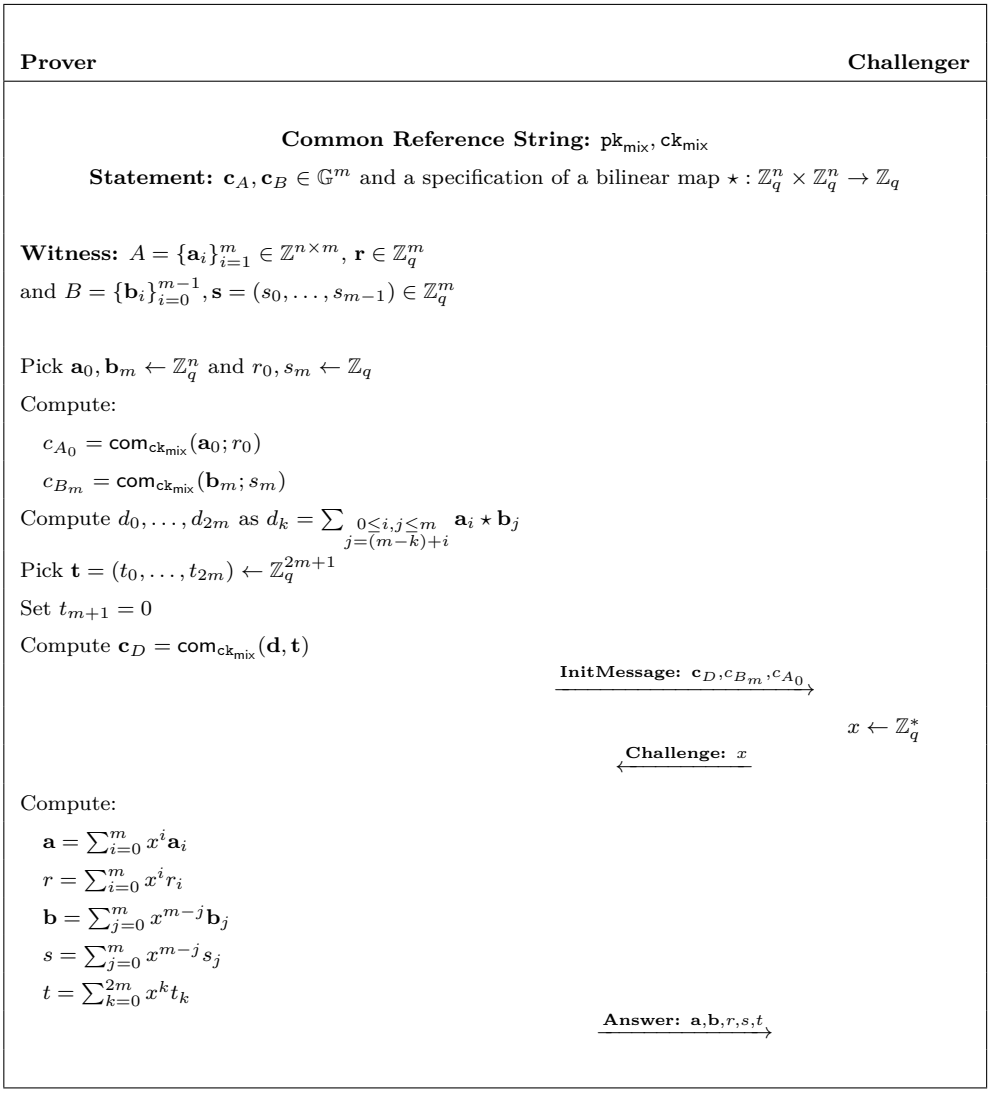

Figure 14: Zero argument: an argument of knowledge of the committed values $\mathbf{a}_1, \mathbf{b}_0, \ldots, \mathbf{a}_m, \mathbf{b}_{m-1}$ s.t. $0 = \sum_{i=1}^m \mathbf{a}_i \star \mathbf{b}_{i-1}$.

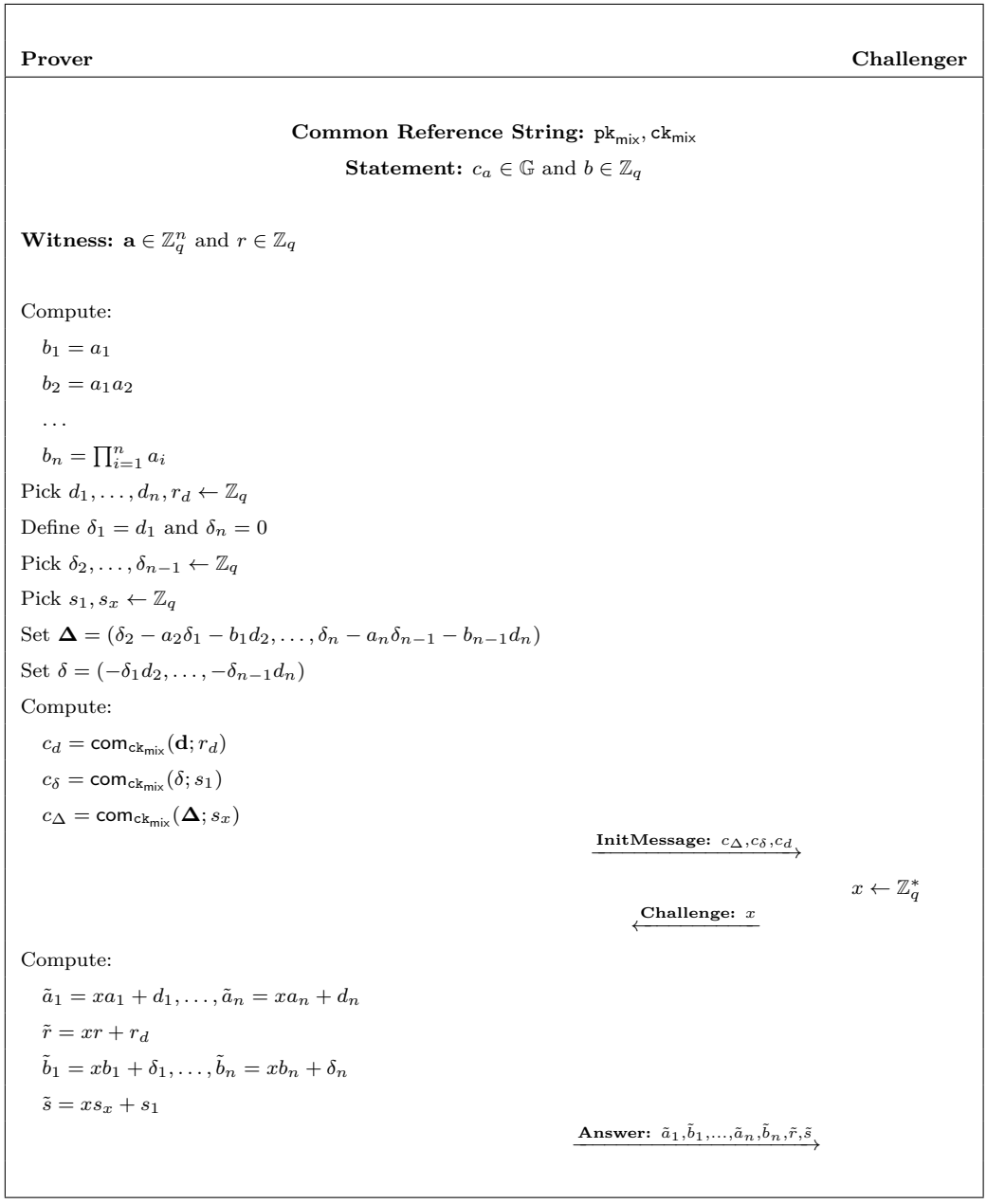

Figure 15: Single value product argument: an argument of knowledge of the opening $a_1, \ldots, a_n, r$ s.t. $c_a = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}; r)$ and $b = \prod_{i=1}^{n} a_i$.

8.3.9 Special Case m=1

A prime number of votes N precludes the mixer from arranging them into a matrix; in that case, one executes the mixing with $n = N$ and $m = 1$. In that case, we reduce the product argument to the single value product argument and omit the Hadamard argument since the matrix already has only one row.

Omitting the Hadamard product argument for $m = 1$ does not affect the shuffle argument's soundness; the product argument proves knowledge of $\prod_{i=1}^{n}\prod_{j=1}^{m} a_{ij} = b$ for the given value b and the commitment $\mathbf{c}_A$ to the matrix $A = \{a_{ij}\}_{i,j=1}^{n,m}$. For $m = 1$, the product argument demonstrates that $\prod_{i=1}^{n} a_i = b$, which corresponds exactly to the single value product argument. Therefore, the product argument is as sound as the single value product argument for $m = 1$.

Moreover, for $m = 1$, the Hadamard product argument becomes redundant since the Hadamard argument's initial message c_b is equal to the product argument's statement, and the zero argument proves a trivial statement. Recall that the Hadamard product argument's initial message is as follows:

$$\boldsymbol{b}_1 = \boldsymbol{a}_1,$$

$$\boldsymbol{b}_2 = \boldsymbol{a}_1\boldsymbol{a}_2,$$

$$\dots$$

$$\boldsymbol{b}_{m-1} = \boldsymbol{a}_1 \cdot \dots \cdot \boldsymbol{a}_{m-1},$$

$$\boldsymbol{b}_m = \boldsymbol{b}.$$

The product argument computes $\boldsymbol{b} = (\prod_{j=1}^{m} a_{1j}, \dots, \prod_{j=1}^{m} a_{nj})$ (see Figure 12) and the Hadamard's prover commits to $\boldsymbol{b}_1, \dots, \boldsymbol{b}_m$ (see Figure 13). If $m = 1$, the matrix A has only one row, which means $\boldsymbol{b} = \boldsymbol{a}_1 = A$. Hence, the Hadamard product argument defines $\boldsymbol{b}_1 = \boldsymbol{a}_1 = \boldsymbol{b}$ and outputs a commitment $c_B = c_b$ as the initial message, which equals the product argument's initial message.
After receiving the challenges x, y, the Hadamard product argument prover prepares the inputs $\mathbf{c}_f, \mathbf{c}_d, F, D, \boldsymbol{\tau}, \mathbf{t}$ for the zero argument. If $m = 1$, the zero argument's inputs are the following:

$$\mathbf{c}_f = c_{-1},$$

$$\mathbf{c}_d = (c_{D_1}, \dots, c_{D_{m-1}}, c_D) = c_D = \prod_{i=1}^{m-1} c_{B_{i+1}}^{x^i} = 0,$$

$$F = \boldsymbol{-1},$$

$$D = (\boldsymbol{d_1}, \dots, \boldsymbol{d_{m-1}}, \boldsymbol{d}) = \boldsymbol{d} = \sum_{i=1}^{m-1} x^i \boldsymbol{b}_{i+1} = \mathbf{0},$$

$$\boldsymbol{\tau} = 0,$$

$$\mathbf{t} = (t_1, \dots, t_{m-1}, t) = t = \sum_{i=1}^{m-1} x^i s_{m-1} = 0.$$

The zero argument proves that $0 = \sum_{i=1}^{m-1} \boldsymbol{a}_{i+1} \star d_i - \mathbf{1} \star \boldsymbol{d}$, which reduces—for $m = 1$—to the trivial statement $0 = \boldsymbol{-1} \star \boldsymbol{d}$, where $\boldsymbol{d} = \mathbf{0}$. Witnesses to such trivial statements are publicly known and the produced proof becomes redundant.

8.4 Verifying Mixing Proofs

8.4.1 $\mathsf{MixVerify}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \pi_{\mathsf{mix}})$

Section 8.3.1 explains how the verifier learns the Common Reference String and derives the matrix dimensions m and n. The MixVerify 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 shuffle as follows:

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

8.4.2 $\mathsf{verifyShuffleArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \pi_{\mathsf{mix}})$

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 π_{mix} as follows:

1. Parses π_{mix} as $(\mathbf{c}_A, \mathbf{c}_B, \pi_{\mathsf{Prod}}, \pi_{\mathsf{MultiExp}})$.
2. Computes challenges:
   - $x \leftarrow \mathsf{RecursiveHash}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \mathbf{c}_A)$.
   - $y \leftarrow \mathsf{RecursiveHash}(\mathbf{c}_B, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \mathbf{c}_A)$.
   - $z \leftarrow \mathsf{RecursiveHash}(1, \mathbf{c}_B, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C}, \mathbf{C}', \mathbf{c}_A)$.
3. Computes the following values:
   (a) $\mathbf{x} = (x, x^2, x^3, \ldots, x^N)^T$
   (b) $\mathbf{c}_D = \mathbf{c}_A^y \mathbf{c}_B$.
   (c) $\mathbf{c}_{-z} = \mathsf{com}_{\mathtt{ck}_{\mathsf{mix}}}(-z, \ldots, -z; \mathbf{0})$.
   (d) $\mathbf{C}^{\mathbf{x}}$.
   (e) $b = \prod_{i=1}^{N}(yi + x^i - z)$.
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 $\top$.
   (c) checks that $\mathsf{verifyProductArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_D\mathbf{c}_{-z}, b, \pi_{\mathsf{Prod}})$ outputs $\top$.

If and only if all verifications above are successful, the algorithm outputs $\top$. Otherwise, it outputs $\perp$.

8.4.3 $\mathsf{verifyMultiExpArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, (\mathbf{C_1}, \ldots, \mathbf{C_m}), C, \mathbf{c}_A, \pi_{\mathsf{MultiExp}})$

This algorithm verifies an argument of knowledge of openings of commitments $\mathbf{c}_A$ to $A = \{\mathbf{a}_j\}_{j=1}^{m}$ such that $C = \mathsf{GetCiphertext}(1, \rho, \mathtt{pk}_{\mathsf{mix}}) \prod_{i=1}^{m} \mathbf{C}_i^{\mathbf{a}_i}$ and $\mathbf{c}_A = \mathsf{com}_{\mathtt{ck}_{\mathsf{mix}}}(A; \mathbf{r})$.

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 $x \leftarrow \mathsf{RecursiveHash}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{C_1}, \ldots, \mathbf{C_m}, C, \mathbf{c}_A, c_{A_0}, \{c_{B_k}\}_{k=0}^{2m-1}, \{E_k\}_{k=0}^{2m-1})$
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) $c_{B_m} = \mathsf{com}_{\mathtt{ck}_{\mathsf{mix}}}(0; 0)$.
   (f) $E_m = C$.
   (g) $c_{A_0} \mathbf{c}_A^{\mathbf{x}} = \mathsf{com}_{\mathtt{ck}_{\mathsf{mix}}}(\mathbf{a}; r)$.
   (h) $c_{B_0} \prod_{k=1}^{2m-1} c_{B_k}^{x^k} = \mathsf{com}_{\mathtt{ck}_{\mathsf{mix}}}(b; s)$.
   (i) $E_0 \prod_{k=1}^{2m-1} E_k^{x^k} = \mathsf{GetCiphertext}(G^b, \tau, \mathtt{pk}_{\mathsf{mix}}) \prod_{i=1}^{m} \mathbf{C}_i^{x^{m-i}\mathbf{a}}$.

If and only if all verifications above are successful, the algorithm outputs $\top$. Otherwise, it outputs $\bot$.

8.4.4 $\mathsf{verifyProductArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, b, \pi_{\mathsf{Prod}})$

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 $\mathsf{verifyHadamardProdArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, c_b, \pi_{\mathsf{HadPA}})$ outputs $\top$.
   (c) Checks that $\mathsf{verifySingleValueProdArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, c_b, b, \pi_{\mathsf{SingleVPA}})$ outputs $\top$.

If and only if all verifications above are successful, the algorithm outputs $\top$. Otherwise, it outputs $\bot$.

8.4.5 $\mathsf{verifyHadamardProdArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, c_b, \pi_{\mathsf{HadPA}})$

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) $x \leftarrow \mathsf{RecursiveHash}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, c_b, \mathbf{c}_B)$.
   (b) $y \leftarrow \mathsf{RecursiveHash}(1, \mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, c_b, \mathbf{c}_B)$.
5. Computes $c_{D_i} = c_{B_i}^{x^i}$, $c_D = \prod_{i=1}^{m-1} c_{B_{i+1}}^{x^i}$ and $c_{-1} = \mathsf{com}_{\mathtt{ck}_{\mathsf{mix}}}(-\mathbf{1}; 0)$.
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 $\mathsf{verifyZeroArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_f, \mathbf{c}_d, \star, \pi_{\mathsf{ZeroArg}})$.

If and only if all verifications above are successful, the algorithm outputs $\top$. Otherwise, it outputs $\bot$.

8.4.6 $\mathsf{verifyZeroArg}(\mathtt{pk}_{\mathsf{mix}}, \mathtt{ck}_{\mathsf{mix}}, \mathbf{c}_A, \mathbf{c}_B, \star, \pi_{\mathsf{ZeroArg}})$

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

If and only if all verifications above are successful, the algorithm outputs $\top$. Otherwise, it outputs $\bot$.

8.4.7 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 $(c_d, c_\delta, c_\Delta, \tilde{a}_1, \tilde{b}_1, \ldots, \tilde{a}_n, \tilde{b}_n, \tilde{r}, \tilde{s})$.
2. Computes challenge $x \leftarrow \mathsf{RecursiveHash}(\mathsf{pk}_{\mathsf{mix}}, \mathsf{ck}_{\mathsf{mix}}, c_\Delta, c_\delta, c_d, b, c_a)$.
3. Check that:
   (a) $c_d, c_\delta, c_\Delta \in \mathbb{G}$.
   (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) $c_\Delta^x c_\delta = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(x\tilde{b}_2 - \tilde{b}_1\tilde{a}_2, \ldots, x\tilde{b}_n - \tilde{b}_{n-1}\tilde{a}_n; \tilde{s})$.
   (e) $\tilde{b}_1 = \tilde{a}_1$.
   (f) $\tilde{b}_n = xb$.

If and only if all verifications above are successful, the algorithm outputs $\top$. Otherwise, it outputs $\bot$.

8.5 Mix Net Security Properties

Bayer and Groth in [1] showed that their shuffle argument is a practical argument that provides the following properties:

- Completeness (Related to our formal definition 2).
- Soundness (Related to our formal definition 3).
- Perfect special honest verifier zero-knowledge (SHVZK) (Related to our formal definition 4).

Perfect special honest verifier zero-knowledge (SHVZK) requires an algorithm simulating the proofs. In this section, we briefly recall the process of the proof simulation outlined in the article.

Shuffle Argument: The perfect simulator $\mathcal{S}_{\mathsf{Mix}}$ of a shuffle argument relies on the perfectly hiding property of the commitment scheme and perfect SHVZK simulators of the underlying proofs.

$$\mathbf{r}, \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^m,$$

$$\mathbf{c}_A \leftarrow \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(0, \ldots, 0; \mathbf{r}),$$

$$\mathbf{c}_B \leftarrow \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(0, \ldots, 0; \mathbf{s}).$$

The perfect simulator $\mathcal{S}_{\mathsf{Mix}}$ outputs $\mathbf{c}_A, \mathbf{c}_B$ and two simulated proofs generated by the perfect SHVZK simulators for the product argument and multi-exponentiation argument respectively.

Multi-Exponentiation Argument: The perfect SHVZK simulator for the multi-exponentiation argument computes a simulated argument that is indistinguishable from the real argument on a challenge x:

$$\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^n,$$

$$r, b, s, s_1, \ldots, s_{m-1}, s_{m+1}, \ldots, s_{2m-1} \xleftarrow{\$} \mathbb{Z}_q,$$

$$c_{A_0} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}; r) \cdot c_A^{-\mathbf{x}},$$

$$s_m = 0,$$

$$c_{B_k} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(0; s_k) \quad \forall k \in (1, 2m-1),$$

$$c_{B_0} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(b; s) \cdot \prod_{k=1}^{2m-1} c_{B_k}^{-x^k},$$

$$E_1, \ldots, E_{m-1}, E_{m+1}, \ldots, E_{2m-1} \xleftarrow{\$} \mathbb{H},$$

$$\tau \xleftarrow{\$} \mathbb{Z}_q,$$

$$E_m = C,$$

$$E_0 = \mathsf{GetCiphertext}(g^b, \tau, \mathsf{pk}_{\mathsf{mix}}) \cdot \prod_{i=1}^{m} C_i^{x^{m-i}\mathbf{a}} \prod_{k=1}^{2m-1} E_k^{-x^k}.$$

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

Product Argument: The perfect SHVZK simulator for the product argument computes a random commitment and runs perfect SHVZK simulators for the underlying proofs.

$$s \xleftarrow{\$} \mathbb{Z}_q,$$

$$c_b \leftarrow \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(0, \ldots, 0; s).$$

The output is c_b and two simulated proofs generated by the perfect SHVZK simulators for the Hadamard product argument and the single value product argument, respectively.

Hadamard Product Argument: To simulate the Hadamard product argument for challenges x, y, the simulator computes random commitments and runs the perfect SHVZK simulator for the zero argument.

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

$$c_{B_2} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{0}; s_2), \ldots, c_{B_{m-1}} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{0}; s_{m-1}),$$

$$c_{B_1} = c_{A_1},$$

$$c_{B_m} = c_b,$$

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

The output is $\mathbf{c}_B$ and a simulated proof generated by the perfect SHVZK simulators for the zero product argument.

Zero Argument: The zero argument for challenge x is simulated as follows:

$$\mathbf{a}, \mathbf{b} \xleftarrow{\$} \mathbb{Z}_q^n,$$

$$r, s, t_0, t_1, \ldots, t_m, t_{m+2}, \ldots, t_{2m} \xleftarrow{\$} \mathbb{Z}_q,$$

$$t_{m+1} = 0,$$

$$c_{A_0} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a}; r) \prod_{i=1}^{m} c_{A_i}^{-x^i},$$

$$c_{B_m} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{b}; s) \prod_{j=0}^{m-1} c_{B_j}^{-x^{m-j}},$$

$$t = \sum_{k=0}^{2m} t_k x^k,$$

$$c_{D_0} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\mathbf{a} \star \mathbf{b}; t_0),$$

$$c_{D_1} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(0; t_1), \ldots, c_{D_{2m}} = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(0; t_{2m}),$$

$$\mathbf{c}_D = (c_{D_0}, \ldots, c_{D_{2m}}).$$

The simulated proof is $(c_{A_0}, c_{B_m}, \mathbf{c}_D, \mathbf{a}, \mathbf{b}, r, s, t)$.

Single Value Product Argument: To simulate the single value product argument for the challenge x, the simulator does the following:

$$\tilde{a}_1, \ldots, \tilde{a}_n, \tilde{r} \xleftarrow{\$} \mathbb{Z}_q,$$

$$c_d = c_a^{-x} \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(\tilde{a}_1, \ldots, \tilde{a}_n; \tilde{r}),$$

$$s_x \xleftarrow{\$} \mathbb{Z}_q,$$

$$c_\Delta = \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(0, \ldots, 0; s_x),$$

$$\tilde{b}_2, \ldots, \tilde{b}_{n-1}, \tilde{s} \xleftarrow{\$} \mathbb{Z}_q,$$

$$\tilde{b}_1 = \tilde{a}_1,$$

$$\tilde{b}_n = xb,$$

$$c_\delta = c_\Delta^{-x} \mathsf{com}_{\mathsf{ck}_{\mathsf{mix}}}(x\tilde{b}_2 - \tilde{b}_1\tilde{a}_2, \ldots, x\tilde{b}_n - \tilde{b}_{n-1}\tilde{a}_n; \tilde{s}).$$

The simulated proof is $(c_d, c_\delta, c_\Delta, \tilde{a}_1, \ldots, \tilde{a}_n, \tilde{b}_1, \ldots, \tilde{b}_n, \tilde{r}, \tilde{s})$.

9 Hard Problems

9.1 The Decisional Diffie-Hellman Problem

Definition 10 (DDH) *The decisional Diffie-Hellman problem (DDH) in a cyclic group $\mathbb{G}$ of order q with a generator g, informally, requires the indistinguishability of g_1^s from a random element, given (g^s, g, g_1) for random $s \in \mathbb{Z}_q^*$ and random $g_1 \in \mathbb{G}$. Figure 16 describes the* DDH *game formally for L different instances.*

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

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

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

Figure 16: DDH game for the cyclic group $\mathbb{G}$ defined by the order q and a generator g.

DDH OVER QUADRATIC RESIDUES. The group of quadratic residues $\mathbb{Q}_p \subset \mathbb{Z}_p^*$ 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 16 the group parameters (sampled at random) are set to $\texttt{gparams} = (p, q, g)$.

9.2 Subgroup Generated by Small Primes (SGSP)

Gjøsteen introduced the Subgroup Generated by Small Primes (SGSP) problem [22]. The SGSP problem is based on a stronger computational hardness assumption than DDH: solving the DDH problem implies solving the SGSP problem. However, the inverse is not necessarily true.

Definition 11 (SGSP) *The SGSP problem in the group of quadratic residues $\mathbb{Q}_p$ with a randomly chosen generator g, informally, requires the indistinguishability of ℓ_1^s from a random element, given (g^s, g, ℓ_1) for random $s \in \mathbb{Z}_q^*$, and $\ell_1 \in \mathbb{Q}_p$ the first prime that is a quadratic residue. Figure 17 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:*

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

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

1. $\ell_1, \ldots, \ell_L \leftarrow \mathbb{Q}_p$ // Find 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. $b' \leftarrow \mathcal{A}(g, \ell_1, \ldots, \ell_L, g^{s_0}, \ell_1^{s_1}, \ldots \ell_L^{s_L})$
7. Output 1 if $b = b'$. Else output 0.

Figure 17: SGSP game for the group of quadratic residues $\mathbb{Q}_p$ defined by the order q and a generator g chosen at random.

The following problem combines the DDH and SGSP problems.

Definition 12 (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 $\ell_1 \in \mathbb{Q}_p$ the first prime that is a quadratic residue. Figure 18 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:*

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

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

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. $b' \leftarrow \mathcal{A}(\mathtt{gparams}, g_1, \ldots, g_{L_1}, \ell_1, \ldots, \ell_{L_2}, g^{s_0}, g_1^{s_1}, \ldots g_{L_1}^{s_{L_1}}, \ell_1^{s_{L_1+1}}, \ldots \ell_{L_2}^{s_{L_1+L_2}})$
7. Output 1 if $b = b'$. Else output 0.

Figure 18: 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 SGSP-hard (and therefore also DDH-hard), then it is also ESGSP-hard.

Part III

Protocol of the Swiss Post Voting System

10 Protocol Description of the Swiss Post Voting System

We base the computational proof upon the following documents:

- The system specification [37] provides sequence diagrams and and explains the Swiss Post Voting System's algorithms, phases, and public parameters.
- The cryptographic primitives specifications [36] detail the underlying cryptographic building blocks.
- The verifier specification [38] elaborates on the auditors' role and specifies the verification algorithms.

These documents detail each party's algorithms, the exchanged messages, and the origin of the parameters to run an election event in the Swiss Post Voting System. They include pseudo-code representations of most algorithms and leave no room for interpretation while being independent of any programming language. We build our implementation upon these specification documents; therefore, we build our computational proof upon this basis as well and avoid redundancies between documents by omitting the protocol description in the computational proof. We assume that the reader is familiar with the documents mentioned above.

11 Abstractions and Assumptions

Ideally, computational proofs correspond precisely to the actual implementation of the protocol. In practice, however, some elements increase the system's robustness or availability but are not necessary for proving verifiability or privacy. On the other hand, some security-relevant but well-known mechanisms, if included in full detail, would bloat the computational model and hurt the analysis's readability and conciseness. In both cases, we abstract details for readability.
This section explains abstractions made and assumptions taken in the computational proof, making the modeling choices and limitations of the security analysis explicit.

11.1 Channel Security

We assume that the parties communicate via authenticated channels. Each outgoing message is digitally signed and linked to a specific context. The receiving party verifies the digital signature and ensures that the context is appropriate.
Given the importance of the topic, we detail the process of signing messages and verifying digital signatures in the cryptographic primitives specification [36]. Moreover, we detail each message and its context in the section "Channel Security" in the system specification [37].

11.2 Voter Authentication

The voting server authenticates the voters and sends the corresponding Verification Card Keystore $\texttt{VCks}_{\texttt{id}}$ to the authenticated voters. Voter authentication relies on one or multiple authentication factors that permit the voter to authenticate to the voting server.
However, according to our model, the voting server is untrustworthy, so we omit this authentication. Instead, we grant the adversary access to the Verification Card Keystores of all voters.

11.3 Write-ins

Certain elections in Switzerland allow voters to write a candidate's name in a form instead of selecting a predefined candidate (so-called write-in candidates). Usually, only a tiny 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 predefined candidates (which are still encoded as the product of primes and linked to a specific Choice Return Code using NIZK proofs). The computational proof shows that the existence of write-in candidates does not impact the protocol's security (Verifiability and Privacy) for predefined candidates. However, the protocol cannot provide sent-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 computational proof considers the presence of write-in and predefined candidates. However, the security guarantees regarding individual verifiability apply only to predefined voting options.

11.4 Trustworthy VerifyConfigPhase execution

At the end of the configuration phase, the auditors execute the VerifyConfigPhase algorithm, whereby the verifier checks the correctness of the system parameters and the control components' NIZK proofs. The computational models for individual verifiability and vote privacy assume a faithful execution of the VerifyConfigPhase algorithm. However, this assumption does not align with the trust model for individual verifiability (see section 2.2.1) and vote privacy (see section 2.2.3), which assume a trustworthy setup component, but an untrustworthy auditor. Nevertheless, we consider this point a minor discrepancy since the (trustworthy) setup component delegates the VerifyConfigPhase to the verifier for performance reasons. Delegating the time-consuming VerifyConfigPhase algorithm to the verifier allows for effective parallelization of some process steps during the configuration phase. The cantons run the setup component and the verifier in a similar, secure offline environment. Thus, we consider the verifier and the setup component equally secure and trustworthy in the configuration phase.

12 Handling Inconsistent Views of Confirmed Votes

The Swiss Post voting system distributes the calculation of Return Codes and the shuffling and decryption of votes among different control components. Since some components are untrustworthy in the threat model for universal verifiability (see section 2.2.2), the cryptographic protocol must account for cases where some control components claim to have recorded and confirmed different sets of votes. The Federal Chancellery's Ordinance and the explanatory report elaborate on that requirement:

> [VEleS Annex 11.11]: The canton anticipates any anomalies and, in consultation with the bodies concerned, draws up an emergency plan specifying the appropriate course of action. It creates transparency towards the public.

> [Explanatory Report, Sec 4.2.1]: As a condition for the successful examination of the proof referred to in Number 2.6, all control components must have recorded the same votes as having been cast in conformity with the system. Cases where the control components show inconsistencies in this respect must be anticipated in accordance with Number 11.11 and the procedure determined in advance.

This section demonstrates the Swiss Post voting system's compliance with the above requirement, highlights how the control components explicitly agree on the confirmed votes, and details how the protocol participants can resolve—after a careful forensic investigation—any inconsistent view on the confirmed votes in line with the threat model Federal Chancellery's Ordinance.

12.1 Voting Phase Agreement Mechanisms

12.1.1 SendVote Explicit Agreement

During the execution of the SendVote protocol, the control components receive each other's partially decrypted partial Choice Return Codes $\mathbf{pCC}_{\mathtt{id}}$ together with proofs of correct exponentiation $\{\boldsymbol{\pi}_{\mathsf{decPCC},j}\}_{j=1}^{m}$. The control components verify these proofs in the algorithm $\mathsf{DecryptPCC_j}$. The auxiliary string $\mathbf{i}_{\mathtt{aux}}$ of the exponentiation proof contains the encrypted vote:

$$\mathbf{i}_{\mathtt{aux}} \leftarrow (\text{"}\mathtt{PartialDecryptPCC}\text{"}, \mathtt{ee}, \mathtt{vc}_{\mathtt{id}}, \mathsf{IntegerToString}(\phi_{2,0}), \ldots, \mathsf{IntegerToString}(\phi_{2,\psi-1}))$$

$$\mathbf{i}_{\mathtt{aux}} \leftarrow (\mathbf{i}_{\mathtt{aux}}, \mathsf{IntegerToString}(\gamma_1^{\mathtt{k}_{\mathtt{id}}}), \mathsf{IntegerToString}(\phi_{1,0}^{\mathtt{k}_{\mathtt{id}}}), \mathsf{IntegerToString}(\gamma_1))$$

$$\mathbf{i}_{\mathtt{aux}} \leftarrow (\mathbf{i}_{\mathtt{aux}}, \mathsf{IntegerToString}(\phi_{1,0}), \ldots, \mathsf{IntegerToString}(\phi_{1,\hat{\delta}-1}), \mathsf{IntegerToString}(k))$$

Hence the control components cannot verify the exponentiation proofs if they disagree with the auxiliary data and oblige the control components to commit to the encrypted vote.

If the verification of at least one exponentiation proof by a trustworthy control component fails, the voting server cannot retrieve the short Choice Return Codes in the algorithm ExtractCRC, and the trustworthy control component will exclude this vote in the tally process. The cryptographic protocol does not foresee any procedure to recover from a failed verification of the exponentiation proofs: the voter would have to use an alternative voting channel (e.g., mail-in voting).

12.1.2 Confirm Explicit Agreement

During the execution of the ConfirmVote protocol, the control components receive each other's hashed long Vote Cast Return Code shares $\{\mathtt{hlVCC}_{\mathtt{id},j}\}_{j=1}^{m}$ and execute the algorithm $\mathsf{VerifyLVCCHash}_{\mathsf{j}}$. The trustworthy control component will add the verification card ID $\mathtt{vc}_{\mathtt{id}}$ to the list of confirmed votes $\mathsf{L}_{\mathtt{confirmedVotes},j}$ if and only if the hashed long Vote Cast Return Code shares $\{\mathtt{hlVCC}_{\mathtt{id},j}\}_{j=1}^{m}$ result in a matching entry in the long Vote Cast Return Code allow list $\mathsf{L}_{\mathtt{lVCC}}$.

$$\mathtt{hhlVCC}_{\mathtt{id}} \leftarrow \mathsf{Base64Encode}(\mathsf{RecursiveHash}(\mathbf{i}_{\mathtt{aux}}, \mathtt{hlVCC}_{\mathtt{id},1}, \mathtt{hlVCC}_{\mathtt{id},2}, \mathtt{hlVCC}_{\mathtt{id},3}, \mathtt{hlVCC}_{\mathtt{id},4}))$$

Check the existence of $\mathtt{hhlVCC}_{\mathtt{id}}$ in $\mathsf{L}_{\mathtt{lVCC}}$

Therefore, a value of $\mathtt{hhlVCC}_{\mathtt{id}}$ matching an entry in the long Vote Cast Return Code allow list $\mathsf{L}_{\mathtt{lVCC}}$ proves to the trustworthy control component that the voter correctly confirmed her vote. Only in that case does the trustworthy control component send the long Vote Cast Return Code share $\mathtt{lVCC}_{\mathtt{id},j}$ to the voting server, allowing him to extract the short Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$ in the algorithm ExtractVCC. Similarly to the SendVote protocol, we do not foresee a recovery procedure for handling inconsistencies in the ConfirmVote protocol. An inconsistent view of the control components prevents the trustworthy control component from calculating an $\mathtt{hhlVCC}_{\mathtt{id}}$ matching an entry in the allow list $\mathsf{L}_{\mathtt{lVCC}}$; therefore, the voting server will be unable to extract the correct short Vote Cast Return code, and the trustworthy control component will omit the unconfirmed vote from the tally process. The voter could still use an alternative voting channel to cast a vote.

12.2 Tally Phase Agreement Mechanism

The trustworthy control component verifies the earlier online control components' shuffle and decryption proofs during the tally before mixing and partially decrypting the votes. This verification mechanism requires that the control components agree on the following:

1. The other control components' shuffle and decryption proofs verify.
2. The control components agree on the votes in the initial ballot box.

The first point is straightforward: given the completeness of the NIZK proofs (see section 7) and the verifiable mix net (see section 8), an honest verifier can objectively determine if a shuffle and decryption

proof is valid or not. Therefore, a false shuffle and decryption proof would require the party providing the incorrect proof to repeat the shuffle and decryption process. Only after validating all shuffle and decryption proofs would the trustworthy control component continue the tally process.
The second point implies a dispute about the set of initial votes[5]. The list of initial ciphertexts consists of all confirmed votes. A malicious (or malfunctioning) control component could claim a different list of initial ciphertexts by inserting, deleting, or modifying votes. If the trustworthy control component has a different list of initial ciphertexts, the shuffle proofs will not verify, and the control component will stop the tally process. However, one could resolve the disagreement about the set of initial votes in the following way (we assume that the trustworthy control component has index h):

- If the $h-1$th list of initial ciphertexts contains votes that are not in the h-th list of initial ciphertexts, the $h-1$-th control component must show evidence that all control components committed to the vote in the protocol SendVote (see section 12.1.1) and that there are hashed long Vote Cast Return Code shares $\{\texttt{hlVCC}_{\texttt{id},j}\}_{j=1}^{m}$ proving that the vote was correctly confirmed in ConfirmVote (see section 12.1.2).
- Conversely, if the h-th list of initial ciphertexts contains more votes than the $h-1$-th list of initial ciphertexts, the trustworthy control component provides evidence that they processed the additional in SendVote and ConfirmVote.
- We treat a modified vote as a simultaneous deletion and insertion of a vote.

Within our threat model, we assume that the trustworthy control component always establishes the correct list of initial ciphertexts. Therefore, we resolve the dispute by requiring the control component(s) preceding the trustworthy control component to repeat the mixing and decryption process and to provide shuffle and decryption proofs matching the trustworthy control component's view. Only after successfully verifying the shuffle and decryption proofs would the trustworthy control component mix and decrypt the votes. The procedure described above to resolve inconsistencies preserves privacy since it guarantees that the trustworthy control component decrypts a correct list of votes, guaranteeing vote privacy.

[5]The control components might process the votes during the voting phase in a different order. Therefore, we assume set equality of the confirmed votes between the control components and the algorithm GetMixnetInitialCiphertexts orders the votes by their verification card ID.

Part IV

Security Analysis

13 Security Framework

We conduct a thorough analysis in the provable secure framework [6], defining the Swiss Post Voting System's desired security properties with games. A game is a set of well-defined instructions between two entities—a challenger (i.e., the game itself) and an adversary—and finalizes with a decision signaling whether the adversary wins or loses. The game-playing framework provides an environment to analyze cryptographic schemes: the scheme under scrutiny exhibits a particular property if the corresponding game decides affirmatively with a 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 changes it to reach an (ideal) game where it is easy to argue that the adversary cannot win. We refer to each gradual change as a game hop. We demonstrate that the adversary cannot distinguish the game before or after the game hop using the security properties of the underlying building blocks, a computational hardness assumption, or an already established property of the Swiss Post Voting System. Ultimately, the entire chain of game hops from the original game to the ideal one boils down to the security properties of the underlying building blocks (sections 3 to 8) or the computational hardness assumptions (section 9).

13.1 Random Oracle Model

Section 6 elaborates on the Swiss Post Voting System's hash functions. We base our security analysis on the Random Oracle model [5], where we replace the hash functions with a publicly accessible random function: the random oracle. Therefore, the adversary cannot compute the hash function by himself but must query the random oracle. The Random Oracle Model is widely used in practice and constitutes an essential tool for proving the security of cryptosystems. Although not explicitly stated in the game definitions, all the games control the random oracle.

13.2 Extractors

In some game hops, we require *extractors*: functions that extract witnesses from NIZK proofs or plaintexts from ciphertexts. Their existence allows us to state clear mathematical definitions and precise game hops. Extractors are used only by the Challenger in our games and never run in reality; therefore, they can be computationally unbounded.

Section 7.2.1 elaborates on the NIZK proofs extractors. We always use these extractors in a *non-adaptive* setting, where the challenger determines the inputs and the adversary cannot repeatedly query the Challenger. Thus, we do not need straightline extractors required for adaptive settings [10].

In addition, we define the following extraction algorithm $\mathsf{Extract}(\mathbf{c}, \mathsf{pk})$ for obtaining the plaintext from the given ciphertext $\mathbf{c}$ using the public key[6] pk. $\mathsf{Extract}(\mathbf{c}, \mathsf{pk})$ recovers the secret key by computing a discrete logarithm and then decrypts the ciphertext $\mathbf{c} = (c_0, c_1, \ldots, c_n)$:

$$\mathbf{m} \leftarrow \mathsf{Extract}(\mathbf{c}, \mathsf{pk}) = \left(c_1 / c_1^{\mathsf{DLOG}(\mathsf{pk})}, \ldots, c_n / c_n^{\mathsf{DLOG}(\mathsf{pk})}\right)$$

We abuse the notation to return $\mathbf{c}$ in case $\mathsf{pk} = 1$.

$$\mathbf{c} \leftarrow \mathsf{Extract}(\mathbf{c}, 1)$$

13.3 Oracles Modeling the Swiss Post Voting System

To show the security of the Swiss Post Voting System, we allow the adversary to query the trustworthy protocol participants as oracles.
We omit oracles for the configuration phase for the following reasons:

- The setup component is trustworthy for individual verifiability (see section 2.2.1) and privacy (see section 2.2.3). The trustworthy control component only accepts queries from the trustworthy setup component. Conversely, the setup component requires the input of at least one trustworthy control component. Therefore, the adversary has nothing to query in the configuration phase.
- In our model for universal verifiability (see section 2.2.2) every party—apart from at least one trustworthy auditor—is under adversarial control. Thus, the adversary is under full control in the configuration phase and has no need to query oracles.

The challenger ensures the correct execution flow (corresponding to the expected protocol execution as outlined in section 1.2) by maintaining the voters' status in the list $\mathsf{L}_{\texttt{VoterStatus}}$. For instance, the challenger only checks the short Vote Cast Return Code $\texttt{VCC}_{\texttt{id}}$ for a specific voter if this voter previously confirmed her vote.

[6] If we need only the first x key components out of n to perform the extraction, the remaining key components are ignored.

Figures 19 and 20 define the available oracles corresponding to the SendVote and ConfirmVote procedures. There are the following oracles:

SendVote oracles:

- $\mathcal{O}$VoterGivesSVK: The adversary can ask an honest voter to provide a vector of encoded voting options $\hat{\mathbf{p}}_{\mathtt{id}}$, a vector of encoded write-ins $\mathbf{w}_{\mathtt{id}}$, and the Start Voting Key $\mathtt{SVK}_{\mathtt{id}}$. The challenger playing on behalf of all honest voters keeps track of all voters and their encoded voting options $\hat{\mathbf{p}}_{\mathtt{id}}$.
- $\mathcal{O}$HonestPartialDecPCC: The adversary can ask the honest control component to perform partial decryption of the encrypted partial Choice Return Codes $\mathbf{pCC}_{\mathtt{id}}$.
- $\mathcal{O}$HonestLCCShare: The adversary can ask the honest control component to generate the CCR_h's long Choice Return Code share $\mathbf{lCC}_{h,\mathtt{id}}$.
- $\mathcal{O}$VoterChecksCRC: The adversary can ask an honest voter to verify the short Choice Return Codes. In practice, the honest voter *should* check all short Choice Return Codes $\mathbf{CC}_{\mathtt{id}}$. However, the Federal Chancellery's Ordinance requires individual verifiability for every *partial vote*, i.e., an arbitrary subset of selected voting options (see section 2.2.1). Therefore, we allow the adversary to select which voting option(s) the honest voter verifies. The challenger playing on behalf of all honest voters keeps track of all voters who performed the verification.

ConfirmVote oracles:

- $\mathcal{O}$VoterGivesBCK: The adversary can ask an honest voter to confirm the vote by providing the Ballot Casting Key $\mathtt{BCK}_{\mathtt{id}}$. The challenger playing on behalf of all honest voters keeps track of all voters who confirmed their vote.
- $\mathcal{O}$HonestCreateLVCCShare: The adversary can ask the honest control component to generate the CCR_h's hashed long Vote Cast Return Code share $\mathtt{hlVCC}_{\mathtt{id},h}$.
- $\mathcal{O}$HonestVerifyLVCCShares: The adversary can ask the honest control component to reveal the CCR_h's long Vote Cast Return Code share $\mathtt{lVCC}_{\mathtt{id},h}$.
- $\mathcal{O}$VoterChecksVCC: The adversary can ask an honest voter to verify the short Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$. The challenger playing on behalf of all honest voters keeps track of all voters who successfully verified the short Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$.

$\mathcal{O}$VoterGivesSVK($\mathtt{id}, \mathcal{S}_{\mathsf{checked}}$):

1. If the set of voting options that the voter verifies $\mathcal{S}_{\mathsf{checked}} = \{\hat{\mathrm{p}}_{\mathtt{id},1}, \ldots, \hat{\mathrm{p}}_{\mathtt{id},k}\}$ is empty, abort and output $\perp$.
2. If $\mathtt{id} \notin \mathcal{ID}_h$ or the voter already voted i.e. $(\mathtt{id}, \mathsf{voted}) \in \mathsf{L}_{\mathtt{VoterStatus}}$, abort and output $\perp$.
3. Select encoded voting options $\hat{\mathbf{p}}_{\mathtt{id}}$ and encoded write-ins $\mathbf{w}_{\mathtt{id}}$ at random and store $(\mathtt{id}, \hat{\mathbf{p}}_{\mathtt{id}}, \mathcal{S}_{\mathsf{checked}})$.
4. Add entry $(\mathtt{id}, \mathsf{voted})$ to $\mathsf{L}_{\mathtt{VoterStatus}}$ and output $(\mathtt{SVK}_{\mathtt{id}}, \hat{\mathbf{p}}_{\mathtt{id}})$.

$\mathcal{O}$HonestPartialDecPCC($\mathtt{vc}_{\mathtt{id}}, \mathtt{E1}, \widetilde{\mathtt{E1}}, \mathtt{E2}, \pi_{\mathsf{Exp}}, \pi_{\mathsf{EqEnc}}$):

1. If $\mathsf{VerifyBallotCCR}_{\mathsf{h}}(\mathtt{vc}_{\mathtt{id}}, \mathtt{E1}, \widetilde{\mathtt{E1}}, \mathtt{E2}, \mathtt{K}_{\mathtt{id}}, \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, \pi_{\mathsf{Exp}}, \pi_{\mathsf{EqEnc}})$ fails, abort and output $\perp$.
2. $(\mathbf{d}_h, \boldsymbol{\pi}_{\mathsf{decPCC},h}) \leftarrow \mathsf{PartialDecryptPCC}_{\mathsf{h}}\left(\mathtt{vc}_{\mathtt{id}}, \mathtt{E1}, \widetilde{\mathtt{E1}}, \mathtt{E2}, \mathbf{pk}_{\mathrm{CCR}_h}, \mathbf{sk}_{\mathrm{CCR}_h}\right)$.
3. Store $(\mathtt{vc}_{\mathtt{id}}, \mathtt{E1}, \widetilde{\mathtt{E1}}, \mathtt{E2}, \mathbf{d}_h, \boldsymbol{\pi}_{\mathsf{decPCC},h})$ and output $\mathbf{d}_h, \boldsymbol{\pi}_{\mathsf{decPCC},h}$

$\mathcal{O}$HonestLCCShare $\left(\mathtt{vc}_{\mathtt{id}}, \{\mathbf{d}_k, \boldsymbol{\pi}_{\mathsf{decPCC},k}\}_{k=1,k\neq h}^{m}\right)$:

1. Get $\mathtt{vc}_{\mathtt{id}}, \mathtt{E1}, \widetilde{\mathtt{E1}}, \mathtt{E2}$ and $\mathbf{d}_h$ corresponding to $\mathtt{vc}_{\mathtt{id}}$. If there is no matching entry, abort and output $\perp$
2. $\mathbf{pCC}_{\mathtt{id}} \leftarrow \mathsf{DecryptPCC}_{\mathsf{h}}\left(\mathtt{vc}_{\mathtt{id}}, \mathbf{d}_{h,\mathtt{id}}, \left\{\mathbf{d}_{k,\mathtt{id}}, \boldsymbol{\pi}_{\mathsf{decPCC},k}, \mathbf{pk}_{\mathrm{CCR}_k}\right\}_{k=1,k\neq h}^{m}, \mathtt{E1}, \widetilde{\mathtt{E1}}, \mathtt{E2}\right)$
3. $(\mathbf{hpCC}_{\mathtt{id}}, \mathbf{lCC}_{h,\mathtt{id}}, \mathtt{K}_{h,\mathtt{id}}, \pi_{\mathsf{expLCC},h}) \leftarrow \mathsf{CreateLCCShare}_{\mathsf{h}}(\mathbf{pCC}_{\mathtt{id}}, \mathtt{k'}_h, \mathtt{vc}_{\mathtt{id}})$
4. Output the CCR_h's long Choice Return Code share $\mathbf{lCC}_{h,\mathtt{id}}$

$\mathcal{O}$VoterChecksCRC($\mathtt{id}, \mathbf{CC}^{*}_{\mathtt{id}}$):

1. If $(\mathtt{id}, \mathsf{voted}) \notin \mathsf{L}_{\mathtt{VoterStatus}} \vee (\mathtt{id}, \mathsf{goodCRC}) \in \mathsf{L}_{\mathtt{VoterStatus}} \vee (\mathtt{id}, \mathsf{wrongCRC}) \in \mathsf{L}_{\mathtt{VoterStatus}}$, abort and output $\perp$.
2. Retrieve the set of voting options $\mathcal{S}_{\mathsf{checked}}$ that the voter verifies .
3. For each $k, \hat{\mathrm{p}}_{\mathtt{id},k} \in \mathcal{S}_{\mathsf{checked}}$, retrieve the expected k-th short Choice Return Code $\mathtt{CC}_{\mathtt{id},k}$ corresponding to the k-th voting option $\hat{\mathrm{p}}_{\mathtt{id},k}$ from the printed voting card. If $\mathtt{CC}^{*}_{\mathtt{id},k} \neq \mathtt{CC}_{\mathtt{id},k}$, set the voter's entry in $\mathsf{L}_{\mathtt{VoterStatus}}$ to $(\mathtt{id}, \mathsf{wrongCRC})$, abort and output $\perp$.
4. Else, if all checked Choice Return Codes were correct, set the voter's entry in $\mathsf{L}_{\mathtt{VoterStatus}}$ to $(\mathtt{id}, \mathsf{goodCRC})$ and output $\top$.

Figure 19: Oracles given to the adversary $\mathcal{A}$ during the SendVote procedure.

$\mathcal{O}$VoterGivesBCK($\mathtt{vc}_{\mathtt{id}}$):

1. If $(\mathtt{id}, \mathsf{goodCRC}) \notin \mathsf{L}_{\mathtt{VoterStatus}}$, abort and output $\perp$.
2. Else, update entry in $\mathsf{L}_{\mathtt{VoterStatus}}$ to $(\mathtt{id}, \mathsf{confirmed})$ and output the Ballot Casting Key $\mathtt{BCK}_{\mathtt{id}}$.

$\mathcal{O}$HonestCreateLVCCShare($\mathtt{vc}_{\mathtt{id}}, \mathtt{CK}_{\mathtt{id}}$):

1. $(\mathtt{hCK}_{\mathtt{id}}, \mathtt{lVCC}_{\mathtt{id},h}, \mathtt{hlVCC}_{\mathtt{id},h}, \mathtt{Kc}_{h,\mathtt{id}}, \pi_{\mathsf{expLVCC},h}, \mathtt{attempts}_{\mathtt{id}}) \leftarrow \mathsf{CreateLVCCShare}_{\mathsf{h}}(\mathtt{CK}_{\mathtt{id}}, \mathtt{k'}_h, \mathtt{vc}_{\mathtt{id}})$.
2. Store $\mathtt{lVCC}_{\mathtt{id},h}$ for $\mathtt{vc}_{\mathtt{id}}$.
3. Output $\mathtt{hlVCC}_{\mathtt{id},j}$.

$\mathcal{O}$HonestVerifyLVCCShares $\left(\mathtt{vc}_{\mathtt{id}}, \{\mathtt{hlVCC}_{\mathtt{id},k}\}_{k=1,k\neq h}^{m}\right)$:

1. Get the last $\mathtt{lVCC}_{\mathtt{id},h}$ and $\mathtt{hlVCC}_{\mathtt{id},h}$ for $\mathtt{vc}_{\mathtt{id}}$. If there is no matching entry, abort and output $\perp$.
2. If $\mathsf{VerifyLVCCHash}_{\mathsf{h}}(\mathsf{L}_{\mathtt{lVCC}}, \mathtt{hlVCC}_{\mathtt{id},h}, \{\mathtt{hlVCC}_{\mathtt{id},k}\}_{k=1,k\neq h}^{m}, \mathtt{vc}_{\mathtt{id}})$ fails, abort and output $\perp$.
3. Else, output the CCR_h's long Choice Return Code share $\mathtt{lVCC}_{\mathtt{id},h}$

$\mathcal{O}$VoterChecksVCC($\mathtt{id}, \mathtt{VCC}^{*}_{\mathtt{id}}$):

1. If $(\mathtt{id}, \mathsf{confirmed}) \notin \mathsf{L}_{\mathtt{VoterStatus}}$, abort and output $\perp$.
2. Retrieve the expected Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$ from the printed voting card.
3. If $\mathtt{VCC}^{*}_{\mathtt{id}} \neq \mathtt{VCC}_{\mathtt{id}}$, update entry in $\mathsf{L}_{\mathtt{VoterStatus}}$ to $(\mathtt{id}, \mathsf{wrongVCC})$
4. Else, update entry in $\mathsf{L}_{\mathtt{VoterStatus}}$ to $(\mathtt{id}, \mathsf{goodVCC})$ and output $\top$.

Figure 20: Oracles given to the adversary $\mathcal{A}$ during the ConfirmVote procedure.

14 Preliminary Result: Return Codes Mapping Table Correctness

This section introduces a property *Return Codes Mapping Table Correctness*: a preliminary result for proving *individual verifiability* in section 16. We first describe an ideal protocol IdealSetupVoting, which constructs a correct Return Codes Mapping Table $\mathbf{CMtable}^{\star}$ without leaking the secret keys of the honest control component. Then we prove that the real protocol SetupVoting is indistinguishable from IdealSetupVoting.

14.1 Idealized Protocol IdealSetupVoting

In this section, we define an *idealized* protocol using *idealized* algorithms, where the Challenger does the following:

- randomizes the output by substituting all exponentiations with random elements,
- simulates the related NIZK proofs,
- extracts the keys from the adversarial outputs to generate the Challenger's outputs.

We use the following conventions when describing *idealized algorithms*:

- we mark *idealized* algorithms with \$ (e.g. $\mathsf{GenVerDat}^{\$}$),
- we mark randomly sampled elements from $\mathbb{Q}_p$ with \$ (e.g. $\mathbf{c}_{\mathtt{ck}}^{\$}$),
- we mark elements that the Challenger generates from extracted adversarial keys with $\star$ (e.g. $\mathsf{L}_{\mathtt{1vcc}}^{\star}$)

In the *idealized* protocol IdealSetupVoting the Challenger randomizes certain outputs of the SetupVoting protocol and extracts the Adversary's keys to generate the Return Codes Mapping table **CMtable** and the long Vote Cast Return Code allow list $\mathsf{L}_{\mathtt{1vcc}}$. Figure 21 shows the protocol IdealSetupVoting.

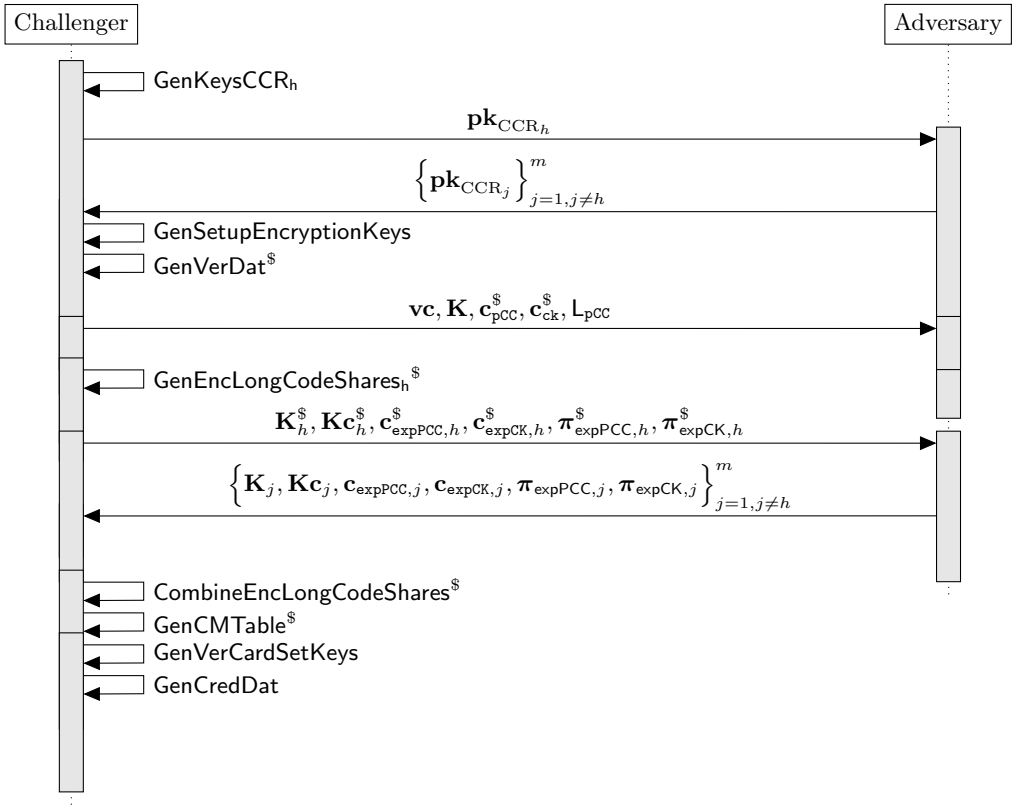

Figure 21: Protocol IdealSetupVoting used in the simulated game sMTC of Figure 22. The Challenger controls the setup component and the honest control component.

The idealized protocol IdealSetupVoting aims to construct a Return Codes Mapping Table $\mathbf{CMtable}^\star$, where the left-hand side of the table is based on randomized elements. We omit the randomization of the right-hand side of the table containing the symmetrically encrypted Choice Return Codes and Vote Cast Return Codes in the idealized protocol IdealSetupVoting. The hops in the individual verifiability games sent-as-intended (section 16.3), vote rejection (section 16.5) and vote injection (section 16.7) will randomize the right-hand side of the Return Codes Mapping Table $\mathbf{CMtable}^\star$.

We randomize only the elements necessary for generating the Return Codes Mapping table **CMtable**; hence we do not randomize the CCR_h Choice Return Codes encryption public key $\mathbf{pk}_{\text{CCR}_h}$ and the vector of verification card public keys $\mathbf{K}$. Moreover, we do not randomize the partial Choice Return Codes allow list $\mathsf{L}_{\mathtt{pCC}}$, as its elements are outputs of the RecursiveHash method and therefore already assumed random in the random oracle model (see section 13.1).

Sometimes, the input and output elements of the idealized algorithms deviate slightly from the actual ones. For instance, CombineEncLongCodeShares$^\$$ requires the NIZK proofs as an input to extract the adversary's secret keys. Moreover, the output of CombineEncLongCodeShares$^\$$ and the input to GenCMTable$^\$$ differ slightly from the actual ones to facilitate the security analysis. However, since the setup component invokes the two algorithms in sequence, the result is equivalent.

14.1.1 $\mathsf{GenVerDat}^{\$}(\mathtt{N_E}, \tilde{\mathbf{p}}, \tilde{\mathbf{v}})$

The idealized algorithm $\mathsf{GenVerDat}^{\$}$ modifies the real $\mathsf{GenVerDat}$ algorithm by substituting $\mathtt{c}^{\$}_{\mathtt{pCC,id}} \xleftarrow{\$} \mathbb{Q}_p^{1+n}$ and $\mathtt{c}^{\$}_{\mathtt{ck,id}} \xleftarrow{\$} \mathbb{Q}_p^2$ for $\forall \mathtt{id} \in \mathcal{ID}$.
$\mathsf{GenVerDat}^{\$}$ outputs $(\mathbf{vc}, \mathbf{SVK}, \mathbf{K}, \mathbf{k}, \mathsf{L}_{\mathtt{pCC}}, \mathbf{BCK}, \mathbf{c}^{\$}_{\mathtt{pCC}}, \mathbf{c}^{\$}_{\mathtt{ck}}, \mathsf{pTable})$.

14.1.2 $\mathsf{GenEncLongCodeShares_h}^{\$}(\mathbf{vc})$

The idealized algorithm $\mathsf{GenEncLongCodeShares_h}^{\$}$ differs from the real $\mathsf{GenEncLongCodeShares_h}$ by sampling random public keys ($\mathtt{K}^{\$}_{h,\mathtt{id}} \xleftarrow{\$} \mathbb{Q}_p$ and $\mathtt{Kc}^{\$}_{h,\mathtt{id}} \xleftarrow{\$} \mathbb{Q}_p$) and ciphertexts ($\mathtt{c}^{\$}_{\mathtt{expPCC},h,\mathtt{id}} \xleftarrow{\$} \mathbb{Q}_p^{1+n}$ and $\mathtt{c}^{\$}_{\mathtt{expCK},h,\mathtt{id}} \xleftarrow{\$} \mathbb{Q}_p^2$). Hence, the idealized $\mathsf{GenEncLongCodeShares_h}^{\$}$ does not require any input arguments besides the vector of verification card IDs $\mathbf{vc}$. Moreover, $\mathsf{GenEncLongCodeShares_h}^{\$}$ simulates the required proofs using a simulator $\mathcal{S}_{\mathsf{Exp}}$ (see section 7.3.2):
$\pi^{\$}_{\mathtt{expPCC},h,\mathtt{id}} \leftarrow \mathcal{S}_{\mathsf{Exp}}\big((g, \mathtt{c}^{\$}_{\mathtt{pCC,id}}), (\mathtt{K}^{\$}_{h,\mathtt{id}}, \mathtt{c}^{\$}_{\mathtt{expPCC},h,\mathtt{id}}), \mathbf{i}_{\mathtt{aux}}\big)$
$\pi^{\$}_{\mathtt{expCK},h,\mathtt{id}} \leftarrow \mathcal{S}_{\mathsf{Exp}}\big((g, \mathtt{c}^{\$}_{\mathtt{ck,id}}), (\mathtt{Kc}^{\$}_{h,\mathtt{id}}, \mathtt{c}^{\$}_{\mathtt{expCK},h,\mathtt{id}}), \mathbf{i}_{\mathtt{aux}}\big)$
$\mathsf{GenEncLongCodeShares_h}^{\$}$ outputs $(\mathbf{K}^{\$}_h, \mathbf{Kc}^{\$}_j, \mathbf{c}^{\$}_{\mathtt{expPCC},j}, \boldsymbol{\pi}^{\$}_{\mathtt{expPCC},j}, \mathbf{c}^{\$}_{\mathtt{expCK},j}, \boldsymbol{\pi}^{\$}_{\mathtt{expCK},j})$.

14.1.3 $\mathsf{CombineEncLongCodeShares}^{\$}(\{\mathbf{c}_{\mathtt{expPCC},j}, \boldsymbol{\pi}_{\mathtt{expPCC},j}, \mathbf{c}_{\mathtt{expCK},j}, \boldsymbol{\pi}_{\mathtt{expCK},j}\}_{j=1,j\neq h}^{m}, \mathbf{vc})$

The idealized algorithm $\mathsf{CombineEncLongCodeShares}^{\$}$ extracts the adversarial secret keys using the extractor of the exponentiation proof system $\mathcal{E}_{\mathsf{Exp}}$ (see section 13.2). Then, $\mathsf{CombineEncLongCodeShares}^{\$}$ uses the extracted Voter Choice Return Code Generation secret key $\mathtt{k}^{\star}_{j,\mathtt{id}}$ and Voter Vote Cast Return Code Generation secret key $\mathtt{kc}^{\star}_{j,\mathtt{id}}$ to generate correct values:

1. Extract the adversary's Voter Choice Return Code Generation secret key $\mathtt{k}^{\star}_{j,\mathtt{id}}$

$$\mathtt{k}^{\star}_{j,\mathtt{id}} \leftarrow \mathcal{E}_{\mathsf{Exp}}(\pi_{\mathtt{expPCC},j,\mathtt{id}}) \quad \forall j \in (1,m), j \neq h, \quad \mathtt{id} \in \mathcal{ID}$$

2. Extract the adversary's Voter Vote Cast Return Code Generation secret key $\mathtt{kc}^{\star}_{j,\mathtt{id}}$

$$\mathtt{kc}^{\star}_{j,\mathtt{id}} \leftarrow \mathcal{E}_{\mathsf{Exp}}(\pi_{\mathtt{expCK},j,\mathtt{id}}) \quad \forall j \in (1,m), j \neq h, \quad \mathtt{id} \in \mathcal{ID}$$

3. Generate the adversary's CCR_j's long Choice Return Code share $\mathbf{lCC}^{\star}_{j,\mathtt{id}}$

$$\mathtt{lCC}^{\star}_{j,\mathtt{id},i} \leftarrow \mathtt{hpCC}^{\mathtt{k}^{\star}_{j,\mathtt{id}}}_{\mathtt{id},i} \quad \forall j \in (1,m), j \neq h, \quad \forall i \in (1,\ldots,n), \quad \mathtt{id} \in \mathcal{ID}$$

4. Generate the adversary's CCR_j's long Vote Cast Return Code share $\mathtt{lVCC}^{\star}_{\mathtt{id},j}$

$$\mathtt{lVCC}^{\star}_{\mathtt{id},j} \leftarrow \mathtt{hCK}^{\mathtt{kc}^{\star}_{j,\mathtt{id}}}_{\mathtt{id}} \quad \forall j \in (1,m), j \neq h, \quad \mathtt{id} \in \mathcal{ID}$$

5. Sample the honest CCR_h's long Choice Return Code share $\mathbf{lCC}^{\$}_{h,\mathtt{id}}$

$$\mathtt{lCC}^{\$}_{h,\mathtt{id},i} \xleftarrow{\$} \mathbb{Q}_p \quad \forall i \in (1,\ldots,n), \quad \mathtt{id} \in \mathcal{ID}$$

6. Sample the honest CCR_h's long Vote Cast Return Code share $\texttt{lVCC}^{\$}_{\texttt{id},h}$

$$\texttt{lVCC}^{\$}_{\texttt{id},h} \xleftarrow{\$} \mathbb{Q}_p \quad \texttt{id} \in \mathcal{ID}$$

7. $\texttt{hlVCC}^{\star}_{\texttt{id},j} \leftarrow \mathsf{Base64Encode}(\mathsf{RecursiveHash}(\mathbf{i}_{\texttt{aux},1}, \texttt{lVCC}^{\star}_{\texttt{id},j})) \quad \forall j \in (1,\dots,m), j \neq h$

8. $\texttt{hlVCC}^{\$}_{\texttt{id},h} \leftarrow \mathsf{Base64Encode}(\mathsf{RecursiveHash}(\mathbf{i}_{\texttt{aux},1}, \texttt{lVCC}^{\$}_{\texttt{id},h}))$

9. Calculate $\texttt{hhlVCC}^{\star}_{\texttt{id}}$. For instance, if the honest control component is the last one, $\texttt{hhlVCC}^{\star}_{\texttt{id}}$ is calculated as follows:

$$\texttt{hhlVCC}^{\star}_{\texttt{id}} \leftarrow \mathsf{Base64Encode}(\mathsf{RecursiveHash}(\mathbf{i}_{\texttt{aux},2}, \texttt{hlVCC}^{\$}_{\texttt{id},1}, \dots, \texttt{hlVCC}^{\$}_{\texttt{id},m-1}, \texttt{hlVCC}^{\star}_{\texttt{id},m}))$$

10. Create the long Vote Cast Return Code allow list

$$\mathsf{L}^{\star}_{\texttt{lVCC}} = \{\texttt{hhlVCC}^{\star}_{\texttt{id}}\}_{\texttt{id}\in\mathcal{ID}}$$

11. $\texttt{pC}^{\$}_{\texttt{id},i} = \texttt{lCC}^{\$}_{h,\texttt{id},i} \cdot \prod_{j=1,j\neq h}^{m} \texttt{lCC}^{\star}_{j,\texttt{id},i} \quad \forall i \in (1,\dots,n) \quad \texttt{id} \in \mathcal{ID}$

12. $\texttt{pVCC}^{\$}_{\texttt{id}} \leftarrow \texttt{lVCC}^{\$}_{\texttt{id},h} \cdot \prod_{j=1,j\neq h}^{m} \texttt{lVCC}^{\star}_{\texttt{id},j} \quad \forall \texttt{id} \in \mathcal{ID}$

$\mathsf{CombineEncLongCodeShares}^{\$}$ outputs $(\mathsf{L}^{\star}_{\texttt{lVCC}}, \{\texttt{pC}^{\$}_{\texttt{id},1}, \dots, \texttt{pC}^{\$}_{\texttt{id},n}, \texttt{pVCC}^{\$}_{\texttt{id}}\}_{\texttt{id}\in\mathcal{ID}})$.

14.1.4 $\mathsf{GenCMTable}^{\$}(\mathbf{vc}, \{\texttt{pC}^{\$}_{\texttt{id},1}, \dots, \texttt{pC}^{\$}_{\texttt{id},n}, \texttt{pVCC}^{\$}_{\texttt{id}}\}_{\texttt{id}\in\mathcal{ID}})$

The idealized algorithm $\mathsf{GenCMTable}^{\$}$ constructs the Return Codes Mapping Table $\mathbf{CMtable}^{\star}$ as follows:

1. $\texttt{lCC}^{\$}_{\texttt{id},i} \leftarrow \mathsf{RecursiveHash}(\texttt{pC}^{\$}_{\texttt{id},i}, \texttt{vc}_{\texttt{id}}, \texttt{ee}, \mathsf{GetCorrectnessIdForVotingOptionIndex}(i)) \quad \forall i \in (1,\psi) \quad \forall \texttt{id} \in \mathcal{ID}$

2. $\texttt{lVCC}^{\$}_{\texttt{id}} \leftarrow \mathsf{RecursiveHash}(\texttt{pVCC}^{\$}_{\texttt{id}}, \texttt{vc}_{\texttt{id}}, \texttt{ee}) \quad \forall \texttt{id} \in \mathcal{ID}$

3. $\texttt{CC}_{\texttt{id},i} \xleftarrow{\$} \mathcal{C}_{cc} \quad \forall i \in (1,n) \quad \forall \texttt{id} \in \mathcal{ID}$, set $\mathbf{CC} = (\texttt{CC}_{\texttt{id},1}, \dots, \texttt{CC}_{\texttt{id},n})_{\texttt{id}\in\mathcal{ID}}$

4. $\texttt{VCC}_{\texttt{id}} \xleftarrow{\$} \mathcal{C}_{vcc} \quad \forall \texttt{id} \in \mathcal{ID}$, set $\mathbf{VCC} = (\texttt{VCC}_{\texttt{id}})_{\texttt{id}\in\mathcal{ID}}$

5. Compute the Return Codes Mapping Table $\mathbf{CMtable}^{\star}$ as:

$$\Big\{ \Big\{ \Big[\mathsf{Base64Encode}(\mathsf{RecursiveHash}(\texttt{lCC}^{\$}_{\texttt{id},i})), \mathsf{Base64Encode}(\texttt{ctCC}_{\texttt{id},i}) \Big] \Big\}_{i=1}^{n},$$
$$\Big[\mathsf{Base64Encode}(\mathsf{RecursiveHash}(\texttt{lVCC}^{\$}_{\texttt{id}})), \mathsf{Base64Encode}(\texttt{ctVCC}_{\texttt{id}}) \Big] \Big\}_{\texttt{id}\in\mathcal{ID}}$$
$$\texttt{ctCC}_{\texttt{id},i} \leftarrow \mathsf{Enc}_{\mathsf{s}}(\texttt{CC}_{\texttt{id},i}; \texttt{skcc}_{\texttt{id},i}) \text{ and } \texttt{skcc}_{\texttt{id},i} \leftarrow \mathsf{KDF}(\texttt{lCC}^{\$}_{\texttt{id},i}, (), 32)$$
$$\texttt{ctVCC}_{\texttt{id}} \leftarrow \mathsf{Enc}_{\mathsf{s}}(\texttt{VCC}_{\texttt{id}}; \texttt{skvcc}_{\texttt{id}}) \text{ and } \texttt{skvcc}_{\texttt{id}} \leftarrow \mathsf{KDF}(\texttt{lVCC}^{\$}_{\texttt{id}}, (), 32)$$

6. Lexicographically order the $\mathbf{CMtable}^{\star}$ entries by the first column.

$\mathsf{GenCMTable}^{\$}$ outputs $(\mathbf{CMtable}^{\star}, \mathbf{CC}, \mathbf{VCC})$.

14.2 Definition and Lemma

The Return Codes Mapping Table property guarantees that SetupVoting is indistinguishable from IdealSetupVoting (section 14.1) and leads to extractable Choice Return Codes and Vote Cast Return Codes.

Definition 13 (ReturnCodesMappingTableCorrectness) SetupVoting *generates a correct Return Codes Mapping table* **CMtable** *and correct allow lists* $\mathsf{L}_{\mathtt{pCC}}$ *and* $\mathsf{L}_{\mathtt{lVCC}}$ *if no Adversary* $\mathcal{A}$ *can distinguish the real setup* rMTC *from the ideal setup* sMTC *(see Figure 22):*

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

$\mathsf{rMTC}_{\mathcal{A}}^{\mathsf{SPVS}}(\mathtt{context})$:

1. Execute $\mathsf{SetupVoting}_{\mathcal{A}}$
2. If $\perp \leftarrow$ VerifyConfigPhase, output $\perp$ (abort)
3. Give $\mathsf{L}_{\mathtt{pCC}}$, $\mathsf{L}_{\mathtt{lVCC}}$ and **CMtable** to $\mathcal{A}$.
4. $b \leftarrow \mathcal{A}$ // The Adversary $\mathcal{A}$ outputs a decision
5. Output b

$\mathsf{sMTC}_{\mathcal{A}}^{\mathsf{SPVS}}(\mathtt{context})$:

1. Execute IdealSetupVoting
2. If $\perp \leftarrow$ VerifyConfigPhase, output $\perp$ (abort)
3. Give $\mathsf{L}_{\mathtt{pCC}}$, $\mathsf{L}^{\star}_{\mathtt{lVCC}}$ and **CMtable*** to $\mathcal{A}$.
4. $b \leftarrow \mathcal{A}$ // The Adversary $\mathcal{A}$ outputs a decision
5. Output b

Figure 22: **(left)** real $\mathsf{rMTC}_{\mathcal{A}}^{\mathsf{SPVS}}$ game **(right)** simulated $\mathsf{sMTC}_{\mathcal{A}}^{\mathsf{SPVS}}$ game. Section 14.1 describes the protocol IdealSetupVoting.

Lemma 1 *In the random oracle model*

$$Adv_{\mathcal{A}}^{\mathsf{SPVS,mtc}} \leq 2 \cdot \mathtt{N_E} \cdot (m-1) \cdot \epsilon_{ext} + 2 \cdot \mathtt{N_E} \cdot Adv_{\mathcal{B}_1}^{\mathsf{Exp,NIZK}} + 2 \cdot \mathtt{N_E} \cdot Adv_{\mathcal{B}_2}^{\mathsf{ElGamal,INDCPA}} + 2 \cdot \mathtt{N_E} \cdot Adv_{\mathcal{B}_3}^{\mathbb{Q}_p,\mathsf{DDH}}$$

where $\mathtt{N_E}$ *is the number of voters,* m *is the total number of control components,* ϵ_{ext} *is the extraction error of the exponentiation proof system,* $\mathcal{B}_1$ *is an adversary against the zero-knowledge property of the exponentiation proof system* Exp*,* $\mathcal{B}_2$ *is an adversary against the semantic security of the ElGamal encryption scheme and* $\mathcal{B}_3$ *is an adversary against the decisional Diffie-Hellman problem (DDH) in a cyclic group* $\mathbb{Q}_p$*.*

14.3 Proof of Lemma 1 (Return Codes Mapping Table Correctness)

Proof Outline

In the proof, we start with the real game rMTC, then, we gradually convert it to the simulated sMTC without the Adversary noticing.

In the first hop rMTC.1, the Challenger uses an extractor to obtain adversarial secret keys and construct a correct Return Codes Mapping Table **CMtable***. Then, in rMTC.2 – rMTC.4, we randomize all challenger's outputs related to the Return Codes Mapping Table **CMtable*** generation:

- The setup component's ciphertexts $\mathbf{c}^{\$}_{\mathtt{pCC}}, \mathbf{c}^{\$}_{\mathtt{ck}}$.
- The honest control component's outputs $(\mathbf{K}^{\$}_h, \mathbf{Kc}^{\$}_j, \mathbf{c}^{\$}_{\mathsf{expPCC},j}, \boldsymbol{\pi}^{\$}_{\mathsf{expPCC},j}, \mathbf{c}^{\$}_{\mathsf{expCK},j}, \boldsymbol{\pi}^{\$}_{\mathsf{expCK},j})$.

After the transformation is completed, the real rMTC and simulated sMTC games are identical.

Game rMTC.1

Objective: In this game, the Challenger ignores responses from all control components; instead, it extracts the secret keys from the NIZK proofs and uses them to construct the Return Codes Mapping Table $\mathbf{CMtable}^{\star}$ and the long Vote Cast Return Code allow list $\mathsf{L}^{\star}_{\mathtt{lVCC}}$.

Adjustments: For constructing the Return Codes Mapping Table $\mathbf{CMtable}^{\star}$, the Challenger executes $\mathsf{CombineEncLongCodeShares}^{\$}$ and $\mathsf{GenCMTable}^{\$}$ and needs to know $\left\{\mathtt{hpCC}_{\mathtt{id},i}^{\sum_{j=1}^{m}\mathtt{k}_{j,\mathtt{id}}}\right\}_{i=1}^{n}$ and $\mathtt{hCK}_{\mathtt{id}}^{\sum_{j=1}^{m}\mathtt{kc}_{j,\mathtt{id}}}$ for all voters $\mathtt{id} \in \mathcal{ID}$. To obtain those values, the Challenger needs the secret keys of the adversarial control components for all voters $\mathtt{id} \in \mathcal{ID}$ i.e. $\{\mathtt{k}_{j,\mathtt{id}}\}_{j=1,j\neq h}^{m}$ and $\{\mathtt{kc}_{j,\mathtt{id}}\}_{j=1,j\neq h}^{m}$.

Therefore, the Challenger extracts the secret keys—using the extractor $\mathcal{E}_{\mathcal{A}}$ of the exponentiation proof system (see section 13.2)—from each of the NIZK proofs $\boldsymbol{\pi}_{\mathsf{expPCC},j}$ ($\mathtt{N_E}$ proofs per component) and $\boldsymbol{\pi}_{\mathsf{expCK},j}$ ($\mathtt{N_E}$ proofs per component) for all $m-1$ control components under adversarial control. As a result the Challenger obtains the extracted witnesses $\left\{\mathtt{k}^{\star}_{j,\mathtt{id}}, \mathtt{kc}^{\star}_{j,\mathtt{id}}\right\}_{j=1,j\neq h}^{m}$.

After that, the following adjustments are made:

- The algorithm CombineEncLongCodeShares (see section 4.1.5 of the system specification [37]) will use the extracted keys $\left\{\mathtt{kc}^{\star}_{j,\mathtt{id}}\right\}_{j=1,j\neq h}^{m}$ to compute:

$$\mathtt{lVCC}^{\star}_{\mathtt{id},j} \leftarrow \mathtt{hCK}_{\mathtt{id}}^{\mathtt{kc}^{\star}_{j,\mathtt{id}}} \quad \forall j \in (1,\ldots,m), j \neq h \quad \forall \mathtt{id} \in \mathcal{ID}$$

  Then, with the honest control component's Voter Voter Cast Return Code Generation secret key $\mathtt{kc}_{h,\mathtt{id}}$, the algorithm computes $\mathtt{pVCC}^{\star}_{\mathtt{id}}$:

$$\mathtt{pVCC}^{\star}_{\mathtt{id}} \leftarrow \mathtt{lVCC}_{\mathtt{id},h} \cdot \prod_{j=1,j\neq h}^{m} \mathtt{lVCC}^{\star}_{\mathtt{id},j} \quad \forall \mathtt{id} \in \mathcal{ID}$$

  The algorithm generates the Vote Cast Return Codes allow list $\mathsf{L}^{\star}_{\mathtt{lVCC}}$ accordingly.

- The algorithm GenCMTable (see section 4.1.6 of the system specification [37]) uses the extracted keys $\left\{\mathtt{k}^{\star}_{j,\mathtt{id}}\right\}_{j=1,j\neq h}^{m}$ and the honest control component's Voter Choice Return Codes Generation secret key $\mathtt{k}_{h,\mathtt{id}}$ to compute:

$$(\mathtt{pC}^{\star}_{\mathtt{id},1}, \ldots, \mathtt{pC}^{\star}_{\mathtt{id},n}) \leftarrow \left\{\mathtt{hpCC}_{\mathtt{id},i}^{\left(\mathtt{k}_{h,\mathtt{id}}+\sum_{j=1,j\neq h}^{m}\mathtt{k}^{\star}_{j,\mathtt{id}}\right)}\right\}_{i=1}^{n}$$

  Then the Challenger uses the computed values $(\mathtt{pC}^{\star}_{\mathtt{id},1}, \ldots, \mathtt{pC}^{\star}_{\mathtt{id},n})$ and $\mathtt{pVCC}^{\star}_{\mathtt{id}}$ (from the previous function) to construct the correct Return Codes Mapping Table $\mathbf{CMtable}^{\star}$.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,mtc.1}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,mtc}}| \leq 2 \cdot \mathtt{N_E} \cdot (m-1) \cdot \epsilon_{ext}$$

where $\mathtt{N_E}$ is the number of voters, m is the total number of control components and ϵ_{ext} is given by the weak simulation extractability property of the Exponentiation proof system Exp.

Reasoning: The algorithm VerifyConfigPhase verifies the control components' exponentiation proofs $\left\{\boldsymbol{\pi}_{\mathsf{expPCC},j}, \boldsymbol{\pi}_{\mathsf{expCK},j}\right\}_{j=1}^{m}$. Therefore, the games $\mathsf{rMTC}_{\mathcal{A}}^{\mathsf{SPVS}}$ and $\mathsf{sMTC}_{\mathcal{A}}^{\mathsf{SPVS}}$ abort, if the exponentiation proofs do not verify (see Figure 22). Therefore, the Adversary can only win if VerifyConfigPhase is successful, which implies that all exponentiation proofs verify.
The Adversary can notice the change only if some of the adversarial NIZK proofs $\{\boldsymbol{\pi}_{\mathsf{expPCC},j}, \boldsymbol{\pi}_{\mathsf{expCK},j}\}_{j=1,j\neq h}^{m}$ verify successfully but $\mathcal{E}_{\mathcal{A}}$ could not extract the corresponding witness. However, the probability of not extracting the witness from a valid proof is bounded by the weak simulation extractability property of the Exponentiation proof system Exp (see definition 8).

Game rMTC.2

Objective: In this game, the Challenger simulates all exponentiation proofs done by the honest control component in SetupVoting.

Adjustments: The algorithm $\mathsf{GenEncLongCodeShares_h}$ (see section 4.1.4 of the system specification [37]), run by the honest control component, uses the canonical simulator $\mathcal{S}_{\mathsf{Exp}}$ to generate $\boldsymbol{\pi}_{\mathsf{expPCC},h}, \boldsymbol{\pi}_{\mathsf{expCK},h}$, i.e., proofs for statements containing the honest control component's Voter Choice Return Code Generation public key $\mathtt{K}_{h,\mathtt{id}}$ and Voter Vote Cast Return Code Generation public key $\mathtt{Kc}_{h,\mathtt{id}}$.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,mtc.2}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,mtc.1}}| \leq 2 \cdot \mathtt{N_E} \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{Exp,NIZK}}$$

where $\mathtt{N_E}$ is the number of voters and $\mathcal{B}_1$ is an adversary against the zero-knowledge property of the Exponentiation proof system Exp (see definition 6).

Reasoning: The Adversary can notice the change only if it can distinguish real and simulated exponentiation proofs; which amounts to breaking the zero-knowledge property of the Exponentiation proof system Exp.

Effect on Future Games: From this game onwards, the honest control component's Voter Choice Return Code Generation public key $\mathtt{K}_{h,\mathtt{id}}$ and Voter Vote Cast Return Code Generation public key $\mathtt{Kc}_{h,\mathtt{id}}$ are no longer used as witnesses in the configuration phase.

Game rMTC.3

Objective: In this game the Challenger, playing on behalf of the setup component, outputs random elements $\mathbf{c}^{\$}_{\texttt{pCC}}, \mathbf{c}^{\$}_{\texttt{ck}}$ instead of the vector of encrypted, hashed partial Choice Return Codes $\mathbf{c}_{\texttt{pCC}}$ and the vector of encrypted, hashed Confirmation Keys $\mathbf{c}_{\texttt{ck}}$.

Adjustments: The algorithm GenVerDat samples the vector of encrypted, hashed partial Choice Return Codes $\mathbf{c}_{\texttt{pCC}}$ and the vector of encrypted, hashed Confirmation Keys $\mathbf{c}_{\texttt{ck}}$ at random; independently from the hashed partial Choice Return Codes $\texttt{hpCC}_{\texttt{id},1}, \ldots, \texttt{hpCC}_{\texttt{id},n}$ and the hashed confirmation key $\texttt{hCK}_{\texttt{id}}$:

$$\mathbf{c}^{\$}_{\texttt{pCC},\texttt{id}} \xleftarrow{\$} \mathbb{Q}_p^{1+n} \; \forall \texttt{id} \in \mathcal{ID}$$

$$\mathbf{c}^{\$}_{\texttt{ck},\texttt{id}} \xleftarrow{\$} \mathbb{Q}_p^{2} \; \forall \texttt{id} \in \mathcal{ID}$$

Claim:

$$|\mathsf{Adv}^{\mathsf{SPVS,mtc.3}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{SPVS,mtc.2}}_{\mathcal{A}}| \leq 2 \cdot \texttt{N}_\texttt{E} \cdot \mathsf{Adv}^{\mathsf{ElGamal,INDCPA}}_{\mathcal{B}_2}$$

where $\texttt{N}_\texttt{E}$ is the number of voters and $\mathcal{B}_2$ is an adversary against the IND-CPA property of the ElGamal multi-recipient encryption scheme (see section 3).

Reasoning: The Adversary can notice the change only if it can distinguish between randomly sampled group elements and ElGamal ciphertexts; which amounts to winning in the IND-CPA game.

Reduction: Technically, the reduction unfolds in two steps: One for the ciphertexts $\mathbf{c}_{\texttt{pCC}}$ and one for $\mathbf{c}_{\texttt{ck}}$; hence the factor of 2 in the bound. We describe only one of these reductions; the other can be done by analogy.

Suppose the Adversary $\mathcal{A}$ can detect the change from rMTC.2 to rMTC.3: i.e., it outputs 0 if there was no change and 1 otherwise. Then we can construct a new Adversary $\mathcal{B}$ that breaks the IND-CPA property of ElGamal:

1. The Adversary $\mathcal{B}$ sends to the IND-CPA game two messages $\mathbf{m}_0 = \left\{\mathsf{HashAndSquare}(\mathrm{p}_i^{\texttt{k}_\texttt{id}})\right\}_{i=1}^{n}$ and $\mathbf{m}_1 = \{R_i\}_{i=1}^{n}$, where $R_i \xleftarrow{\$} \mathbb{Q}_p$.
2. The IND-CPA game replies with $\mathbf{c}_\beta$, which is either an encryption of $\mathbf{m}_0$ (same as in rMTC.2) or an encryption of random messages i.e. a vector of random elements (same as in rMTC.3).
3. The Adversary $\mathcal{B}$ repeats the query $\texttt{N}_\texttt{E}$ times to get $\mathbf{c}^{\beta}_{\texttt{pCC}}$, which is either identical to the one from rMTC.2 or rMTC.3, and sends $\mathbf{c}^{\beta}_{\texttt{pCC}}$ to the Adversary $\mathcal{A}$.
4. The Adversary $\mathcal{A}$ replies with a coin β.
5. The Adversary $\mathcal{B}$ forwards β to the IND-CPA game.

Adversary $\mathcal{B}$ wins in at least the cases where Adversary $\mathcal{A}$ wins, therefore $\mathsf{Adv}_{\mathcal{B}} \geq \mathsf{Adv}_{\mathcal{A}}$. However, as $\mathcal{B}$ attempts to break the IND-CPA security of the multi-recipient ElGamal encryption scheme, it must also hold that $\mathsf{Adv}_{\mathcal{B}} \leq \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{ElGamal,INDCPA}}$ for any Adversary $\mathcal{B}'$. We conclude that $\mathsf{Adv}_{\mathcal{A}} \leq \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{ElGamal,INDCPA}}$.

Game rMTC.4

Objective: In this game the Challenger will change the public keys as well as the results of all exponentiations done by the honest control component to random elements.

Adjustments: To support the change, the following adjustments are made:

- The algorithm $\mathsf{GenEncLongCodeShares_h}$ samples at random the Voter Choice Return Code Generation public key $\mathtt{K}_{h,\mathtt{id}}^{\$}$, the Voter Vote Cast Return Code Generation public key $\mathtt{Kc}_{h,\mathtt{id}}^{\$}$, the vector of exponentiated, encrypted, hashed partial Choice Return Codes $\mathbf{c}_{\mathtt{expPCC},h}$, and the vector of exponentiated, encrypted, hashed Confirmation Keys $\mathbf{c}_{\mathtt{expCK},h}$ from $\mathbb{Q}_p$:

$$\mathtt{K}_{h,\mathtt{id}}^{\$} \xleftarrow{\$} \mathbb{Q}_p \ \forall \mathtt{id} \in \mathcal{ID}$$

$$\mathtt{Kc}_{h,\mathtt{id}}^{\$} \xleftarrow{\$} \mathbb{Q}_p \ \forall \mathtt{id} \in \mathcal{ID}$$

$$\mathtt{c}_{\mathtt{expPCC},h,\mathtt{id}}^{\$} \xleftarrow{\$} \mathbb{Q}_p^{1+n} \ \forall \mathtt{id} \in \mathcal{ID}$$

$$\mathtt{c}_{\mathtt{expCK},h,\mathtt{id}}^{\$} \xleftarrow{\$} \mathbb{Q}_p^{2} \ \forall \mathtt{id} \in \mathcal{ID}$$

  instead of:

$$\mathtt{K}_{h,\mathtt{id}} \leftarrow g^{\mathtt{k}_{h,\mathtt{id}}} \ \forall \mathtt{id} \in \mathcal{ID}$$

$$\mathtt{Kc}_{h,\mathtt{id}} \leftarrow g^{\mathtt{kc}_{h,\mathtt{id}}} \ \forall \mathtt{id} \in \mathcal{ID}$$

$$\mathtt{c}_{\mathtt{expPCC},h,\mathtt{id}} \leftarrow \mathtt{c}_{\mathtt{pCC},\mathtt{id}}^{\mathtt{k}_{h,\mathtt{id}}} \ \forall \mathtt{id} \in \mathcal{ID}$$

$$\mathtt{c}_{\mathtt{expCK},h,\mathtt{id}} \leftarrow \mathtt{c}_{\mathtt{ck},\mathtt{id}}^{\mathtt{kc}_{h,\mathtt{id}}} \ \forall \mathtt{id} \in \mathcal{ID}$$

- The algorithm $\mathsf{GenCMTable}$ constructs the Return Codes Mapping Table $\mathbf{CMtable}^{\star}$ based on values $\left\{x_{i,\mathtt{id}}^{\$} \cdot \mathtt{hpCC}_{\mathtt{id},i}^{\sum_{j=1,j\neq h}^{m} \mathtt{k}_{j,\mathtt{id}}}\right\}_{i=1}^{n}$ and $y_{\mathtt{id}}^{\$} \cdot \mathtt{hCK}_{\mathtt{id}}^{\sum_{j=1,j\neq h}^{m} \mathtt{kc}_{j,\mathtt{id}}}$ instead of $\left\{\mathtt{hpCC}_{\mathtt{id},i}^{\sum_{j=1}^{m} \mathtt{k}_{j,\mathtt{id}}}\right\}_{i=1}^{n}$ and $\mathtt{hCK}_{\mathtt{id}}^{\sum_{j=1}^{m} \mathtt{kc}_{j,\mathtt{id}}}$, where all $x_{i,\mathtt{id}}^{\$}$ and $y_{\mathtt{id}}^{\$}$ are sampled at random from $\mathbb{Q}_p$.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,mtc.4}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,mtc.3}}| \leq 2 \cdot \mathtt{N_E} \cdot \mathsf{Adv}_{\mathcal{B}_3}^{\mathbb{Q}_p,\mathsf{DDH}}$$

where $\mathtt{N_E}$ is the number of voters and $\mathcal{B}_3$ is an adversary against the decisional Diffie-Hellman (DDH) problem in a cyclic group $\mathbb{Q}_p$ (see section 9.1).

Reasoning: The adversary can notice the change only if the adversary can distinguish between randomly sampled elements and the results of the exponentiation; which amounts to breaking the DDH problem in $\mathbb{Q}_p$.

Reduction: Technically, this game unfolds in two steps: one for the Voter Choice Return Code Generation secret key $\mathtt{k}_{h,\mathtt{id}}$, and another for the Voter Vote Cast Return Code Generation secret key $\mathtt{kc}_{h,\mathtt{id}}$; hence the factor of 2 in the bound. We describe only one reduction; the other one can be done by analogy.

Suppose the Adversary $\mathcal{A}$ can detect the change from rMTC.3 to rMTC.4: i.e., the adversary outputs 0 if there was no change and 1 otherwise. Then we can construct a new Adversary $\mathcal{B}$ that breaks the DDH assumption in $\mathbb{Q}_p$:

1. The Adversary $\mathcal{B}$ gets from the DDH game a tuple of $2 \cdot (2n+2)$ elements:

$$(g, g_1, \ldots, g_n, g_{n+1}, \ldots, g_{2\cdot n}, g^{s_0}, g_1^{s_1}, \ldots, g_n^{s_n}, g_{n+1}^{s_{n+1}}, \ldots, g_{2n+1}^{s_{2n+1}})$$

   Now, the Adversary $\mathcal{B}$ uses the tuple to generate a view for the Adversary $\mathcal{A}$:

   - $\mathcal{B}$ uses g^{s_0} as the Voter Choice Return Code Generation public key $\mathtt{K}_{h,\mathtt{id}}$ for the voter $\mathtt{id}$.
   - Then $\mathcal{B}$ programs the hash oracle HashAndSquare to output g_i whenever it receives $\mathrm{p}_i^{\mathtt{k}_{\mathtt{id}}}$ as the input for $\forall i \in (1, \ldots, n)$; in other words, $\mathcal{B}$ sets $(\mathtt{hpCC}_{\mathtt{id},1}, \ldots, \mathtt{hpCC}_{\mathtt{id},n}) = (g_1, \ldots, g_n)$.
   - After that, $\mathcal{B}$ sets the setup component's output to $\mathtt{c}_{\mathtt{pCC},\mathtt{id}} = (g_{n+1}, \ldots, g_{2\cdot n+1})$ and the result of the exponentiation done by the honest control component to $\mathtt{c}_{\mathtt{expPCC},j,\mathtt{id}} = (g_{n+1}^{s_{n+1}}, \ldots, g_{2\cdot n+1}^{s_{2\cdot n+1}})$.
   - Finally, $\mathcal{B}$ uses the values $g_1^{s_1}, \ldots, g_n^{s_n}$ instead of $(\mathtt{hpCC}_{\mathtt{id},1}^{\mathtt{k}_{h,\mathtt{id}}}, \ldots, \mathtt{hpCC}_{\mathtt{id},n}^{\mathtt{k}_{h,\mathtt{id}}})$ for generating the Return Codes Mapping Table **CMtable**.

   If $s_0 = s_1 = \cdots = s_{2\cdot n+1} = s = \mathtt{k}_{h,\mathtt{id}}$, then:

   - $\mathtt{K}_{h,\mathtt{id}} = g^s$
   - $\mathtt{c}_{\mathtt{expPCC},j,\mathtt{id}} = (g_{n+1}^s, \ldots, g_{2\cdot n+1}^s) = \mathtt{c}_{\mathtt{pCC},\mathtt{id}}^{\mathtt{k}_{h,\mathtt{id}}}$
   - $\left(\mathtt{hpCC}_{\mathtt{id},1}, \ldots, \mathtt{hpCC}_{\mathtt{id},n}\right)^s = \left(g_1, \ldots, g_n\right)^s = (g_1^s, \ldots, g_n^s) = (\mathtt{hpCC}_{\mathtt{id},1}^{\mathtt{k}_{h,\mathtt{id}}}, \ldots, \mathtt{hpCC}_{\mathtt{id},n}^{\mathtt{k}_{h,\mathtt{id}}})$

2. The Adversary $\mathcal{B}$ repeats the query $\mathtt{N_E}$ times to get $(\mathbf{c}_{\mathtt{pCC}}, \mathbf{c}_{\mathtt{ck}}, \mathbf{K}_j, \mathbf{Kc}_j, \mathbf{c}_{\mathtt{expPCC},j}, \mathbf{c}_{\mathtt{expCK},j})$ and sends the result to the Adversary $\mathcal{A}$.
   The result of the query corresponds either to the Challenger's output in rMTC.3 or to the one in rMTC.4.

3. The Adversary $\mathcal{A}$ replies with a coin β .

4. The Adversary $\mathcal{B}$ forwards β to the DDH game.

If the Adversary $\mathcal{A}$ can distinguish rMTC.3 and rMTC.4, then the Adversary $\mathcal{B}$ will win the DDH game. Moreover, since VerifyConfigPhase checks all verifiably generated group parameters, we can be sure that we deal with the $\mathbb{Q}_p$ group, where DDH is assumed to be hard (see section 9.1). Therefore, we conclude that the Adversary $\mathcal{A}$ cannot distinguish rMTC.3 and rMTC.4.

Conclusion: The game rMTC.4 does exactly what IdealSetupVoting does. Therefore, the games rMTC.4 and sMTC are identical and $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,mtc.4}} = 0$.
Collecting the bounds of the hops concludes the proof.

15 Preliminary Result: Vote Compliance

The following section demonstrates *vote compliance*, which forms a preliminary result for proving individual verifiability in section 16. The *vote compliance* property relates to the requirement from section 2.2.1 that a vote must be *cast in conformity with the system*. Informally, *vote compliance* demonstrates that the trustworthy control component only registers a ballot that conforms to a predetermined method of completing a ballot paper in a vote or election. For instance, the ballot should contain only one voting option corresponding to a candidate in an election with one seat.

15.1 Definition and Lemma

We formalize vote compliance by defining a *compliant ballot*:

Definition 14 (CompliantBallot) *Let $\mathbb{G}_q$ be an (ElGamal) cyclic group with generator g, $\mathtt{EL_{pk}} \in \mathbb{G}_q^{\delta}$ the election public key, $\mathbf{pk}_{\mathrm{CCR}} = (\mathtt{pk}_{\mathrm{CCR},1}, \ldots, \mathtt{pk}_{\mathrm{CCR},\varphi}) \in \mathbb{G}_q^{\varphi}$ the Choice Return Codes encryption public key $\mathbf{pk}_{\mathrm{CCR}}$ and let $(\mathtt{K_{id}}, \mathtt{k_{id}}) \in \mathbb{G}_q \times \mathbb{Z}_q$ be the verification card key pair for the verification card ID $\mathtt{vc_{id}}$. A ballot $\mathtt{b_{id}} = (\mathtt{vc_{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \pi_{\mathtt{Exp}}, \pi_{\mathtt{EqEnc}})$ with $\mathtt{E2} = (c_{2,0}, c_{2,1}, \ldots, c_{2,\psi})$, is* compliant *if for $\widetilde{\mathtt{E2}} \leftarrow \left(c_{2,0}, \prod_{i=1}^{\psi} c_{2,i}\right)$ it holds:*

(i) The ciphertext $\widetilde{\mathtt{E2}}$ encrypts $m^{\mathtt{k_{id}}}$, where m is the first plaintext element inside $\mathtt{E1} = (\gamma_1, \phi_{1,0}, \ldots, \phi_{1,\hat{\delta}-1})$:

$$\mathsf{Extract}((\gamma_1, \phi_{1,0}), \mathtt{EL_{pk}})^{\mathtt{k_{id}}} = \mathsf{Extract}(\widetilde{\mathtt{E2}}; \prod_{i=1}^{\psi} \mathtt{pk}_{\mathrm{CCR},i})$$

(ii) The ciphertexts $\widetilde{\mathtt{E1}}$ and $\widetilde{\mathtt{E2}}$ encrypt the same message:

$$\mathsf{Extract}(\widetilde{\mathtt{E1}}, \mathtt{EL_{pk}}) = \mathsf{Extract}(\widetilde{\mathtt{E2}}; \prod_{i=1}^{\psi} \mathtt{pk}_{\mathrm{CCR},i})$$

(iii) There are exactly ψ distinct partial Choice Return Codes encrypted in $\mathtt{E2}$:

$$\mathsf{Extract}\left((c_{2,0}, c_{2,k}); \mathtt{pk}_{\mathrm{CCR},k}\right) \neq \mathsf{Extract}\left((c_{2,0}, c_{2,t}); \mathtt{pk}_{\mathrm{CCR},t}\right) \ \forall k \neq t \text{ such that } 0 < k, t \leq \psi$$

(iv) Each i-th partial Choice Return Codes vector component $\mathtt{pCC_{id},}_i$ is correctly constructed:

$$\mathsf{Extract}\left((c_{2,0}, c_{2,i}); \mathtt{pk}_{\mathrm{CCR},i}\right) = \mathrm{p}_i^{\mathtt{k_{id}}} \ \forall i \text{ such that } 0 < i \leq \psi$$

and contains a valid combination of voting options as enforced by $\mathtt{correctnessID}_i$ (see section 3.4 of the system specification [37]).

(v) The first ciphertext element $(\gamma_1, \phi_{1,0})$ in $\mathtt{E1}$ encrypts a product of exactly ψ voting options:

$$\mathsf{Extract}((\gamma_1, \phi_{1,0}), \mathtt{EL_{pk}}) = \prod_{i=1}^{\psi} \mathrm{p}_i$$

The above definition ignores the write-in elements contained in $\texttt{E1} = (\gamma_1, \phi_{1,0}, \ldots, \phi_{1,\hat{\delta}-1})$. We consider write-ins as always cast *in conformity with the system*, which is in line with the Federal Chancellery's Ordinance [15] and the explanatory report [16]:

> [Explanatory Report, Sec 4.2.1]: In elections run using the first-past-the-post system, blank text fields ('write-in votes') are always considered to have been completed in conformity with the system.

MODELING VOTE COMPLIANCE. Adversaries against vote compliance attempt to bypass the honest control component's verifications, thereby forcing registration of a ballot that does *not* meet definition 14 of a compliant ballot. We analyze the Swiss Post Voting System up to the termination of the protocol SendVote since, at this point, the honest control component marks the ballot as *sent*. In Figure 23, we derive a game capturing adversaries against vote compliance. The advantage of $\mathcal{A}$ is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,vc}} = Pr\left[1 \leftarrow \mathsf{vc}_{\mathcal{A}}(\texttt{context})\right]$$

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

$$\mathsf{bad}_{\mathsf{vc}} = \begin{cases} \mathtt{b}_{\mathtt{id}} \text{ is not a compliant ballot (see Definition 14).} \\ \text{ballot is accepted by the honest control component i.e. } \mathtt{vc}_{\mathtt{id}} \in \mathsf{L}_{\mathtt{sentVotes},h}. \end{cases}$$

$\underline{\mathsf{vc}_{\mathcal{A}}(\texttt{context})}$

1. $\mathtt{id}, h \leftarrow \mathcal{A}$ // $\mathcal{A}$ specifies the target voter and the honest control component.
2. $(\mathbf{CMtable}, \mathsf{L}_{\mathtt{lVCC}}, \mathsf{L}_{\mathtt{pCC}}, \mathbf{K}, \mathbf{VCard}, \mathbf{VCks}, (\mathbf{sk}_{\mathrm{CCR}_h}, \mathbf{kc}_h, \mathbf{k}_h), \mathbf{pk}_{\mathrm{CCR}}, (\mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{K}_j, \mathbf{Kc}_j)_{j=1}^{m}) \leftarrow \mathsf{SetupVoting}$
3. $\left(\mathtt{EL}_{\mathtt{pk}}, \mathtt{EL}_{\mathtt{sk},h}, (\mathtt{EL}_{\mathtt{pk},j})_{j=1}^{m}, \mathtt{EB}_{\mathtt{sk}}, \mathtt{EB}_{\mathtt{pk}}\right) \leftarrow \mathsf{SetupTally}$
4. If VerifyConfigPhase fails, abort and output 0. // Verify the configuration phase—see section 11.4.
5. Give the following data to $\mathcal{A}$:

$$\left(\mathbf{VCard}, \mathbf{VCks}, \mathbf{CMtable}, \mathsf{L}_{\mathtt{lVCC}}, \mathsf{L}_{\mathtt{pCC}}, \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, (\mathtt{EL}_{\mathtt{pk},j}, \mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{K}_j, \mathbf{Kc}_j)_{j=1}^{m}, \mathbf{K}, \mathtt{EB}_{\mathtt{sk}}, \mathtt{EB}_{\mathtt{pk}}\right)$$

6. The Adversary $\mathcal{A}$ is allowed to query the oracles $\mathcal{O}$HonestPartialDecPCC and $\mathcal{O}$HonestLCCShare defined in section 13.3.
7. If $\mathsf{bad}_{\mathsf{vc}}$, output 1

Figure 23: Game modeling attacks against vote compliance. The protocols SetupVoting and SetupTally are executed in interaction with the Adversary $\mathcal{A}$. Public parameters context are implicit.

Lemma 2 *Let m be the number of the control components and ψ the number of selected voting options. In the random oracle model*

$$Adv_{\mathcal{A}}^{\mathsf{SPVS,vc}} \leq Adv_{\mathcal{B}_1}^{\mathsf{Exp,sSOUND}} + Adv_{\mathcal{B}_2}^{\mathsf{EqEnc,sSOUND}} + \psi \cdot (m-1) \cdot Adv_{\mathcal{B}_3}^{\mathsf{Exp,sSOUND}}$$

where m is the total number of control components, ψ is the number of allowed selections, $\mathcal{B}_1$ and $\mathcal{B}_3$ are adversaries against the simulation soundness property of the exponentiation Exp *proof system, and $\mathcal{B}_2$ is an adversary against the simulation soundness property of the plaintext equality proof system* EqEnc.

15.2 Proof of Lemma 2 (Vote Compliance)

Proof Outline

We show that the soundness of the NIZK proofs guarantees vote compliance. To this end, the Challenger adds extra checks in each game hop until we fulfill definition 14.

Game vc.1

Objective: In this game, the Challenger aborts if $\widetilde{\texttt{E1}}$ is not an encryption of $\rho^{\texttt{k}_{\texttt{id}}}$, where ρ is the first plaintext element encrypted in $\texttt{E1} = (\gamma_1, \phi_{1,0}, \ldots, \phi_{1,\hat{\delta}-1})$.

Adjustments: After the successful execution of the $\mathsf{VerifyBallotCCR_h}$ algorithm (see oracle $\mathcal{O}$HonestPartialDecPCC in Fig. 19), the Challenger manually checks that $\mathsf{Extract}(\widetilde{\texttt{E1}}, \texttt{EL}_{\texttt{pk}}) = (\mathsf{Extract}((\gamma_1, \phi_{1,0}); \texttt{EL}_{\texttt{pk}}))^{\texttt{k}_{\texttt{id}}}$. If the equality does not hold, the Challenger aborts.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,vc}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,vc.1}}| \leq \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{Exp,sSOUND}}$$

where $\mathcal{B}_1$ is an adversary against the simulation soundness property of the exponentiation proof system Exp.

Reasoning: The Adversary can notice the extra checks only if it can generate an exponentiation proof for an invalid statement without the honest control component noticing (i.e. $\mathsf{VerifyBallotCCR_h}$). However, the soundness of the exponentiation argument guarantees that this would happen only with negligible probability.

Reduction: In the reduction, $\mathcal{B}_1$ uses π_{Exp} and submits $\Big((g, (\gamma_1, \phi_{1,0})), (\texttt{K}_{\texttt{id}}, \widetilde{\texttt{E1}}), \pi_{\mathsf{Exp}}\Big)$ as the challenge statement/proof pair (see Figure 9).

Effect on the future games: This game hop restricts the adversary to generate a ballot with the correct relation between $\texttt{E1}$ and $\widetilde{\texttt{E1}}$.

Game vc.2

Objective: In this game, the Challenger aborts if the ciphertext $\widetilde{\texttt{E1}}$ encrypts a different message than $\widetilde{\texttt{E2}}$.

Adjustments: After the successful execution of the $\mathsf{VerifyBallotCCR_h}$ algorithm (see oracle $\mathcal{O}\mathsf{HonestPartialDecPCC}$ in Fig. 19), the Challenger manually checks that $\widetilde{\texttt{E1}}$ and $\widetilde{\texttt{E2}}$ encrypt the same message. First, the Challenger computes $\widetilde{\texttt{E2}} \leftarrow \left(c_{2,0}, \prod_{i=1}^{\psi} c_{2,i}\right)$. Then, the Challenger checks that $\mathsf{Extract}(\widetilde{\texttt{E1}}, \texttt{EL}_{\texttt{pk}}) = \mathsf{Extract}(\widetilde{\texttt{E2}}; \prod_{i=1}^{\psi} \texttt{pk}_{\text{CCR},i})$. If the equality does not hold, the Challenger aborts.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,vc.1}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,vc.2}}| \leq \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{EqEnc,sSOUND}}$$

where $\mathcal{B}_2$ is an adversary against the simulation soundness property of the plaintext equality proof system EqEnc.

Reasoning: The Adversary can notice the extra check only if it can generate a plaintext equality proof for an invalid statement without the honest control component noticing (i.e. $\mathsf{VerifyBallotCCR_h}$ returns $\top$ for a bogus proof). However, the soundness of the plaintext equality argument guarantees that this would happen only with negligible probability.

Reduction: In the reduction, $\mathcal{B}_2$ uses π_{EqEnc} and submits $((g, \texttt{EL}_{\texttt{pk}}, \prod_{i=1}^{\psi} \texttt{pk}_{\text{CCR},i}, \widetilde{\texttt{E1}}, \widetilde{\texttt{E2}}), \pi_{\mathsf{EqEnc}})$ as the challenge statement/proof pair (see Figure 9).

Effect on the future games: This game hop restricts the adversary to generate a ballot with a correct relation between $\widetilde{\texttt{E1}}$ and $\widetilde{\texttt{E2}}$.

Game vc.3

Objective: In this game, the Challenger aborts if the vector of partial Choice Return Codes $\mathbf{pCC}_{\texttt{id}}$—obtained after decryption— is not equal to the plaintext encrypted in $\texttt{E2}$.

Adjustments: After the successful execution of the $\mathsf{DecryptPCC_h}$ algorithm (see oracle $\mathcal{O}\mathsf{HonestPartialDecPCC}$ in Fig. 19), the Challenger manually checks that $\mathsf{Extract}(\texttt{E2}, \mathbf{pk}_{\text{CCR}}) = \mathbf{pCC}_{\texttt{id}}$, where $\mathbf{pCC}_{\texttt{id}}$ is the output of the $\mathsf{DecryptPCC_h}$ algorithm. If the equality does not hold, the Challenger aborts.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{vc}.2} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{vc}.3}| \leq \psi \cdot (m-1)\mathsf{Adv}_{\mathcal{B}_3}^{\mathsf{Exp},\mathsf{sSOUND}}$$

where m is the total number of control components, ψ is the number of allowed selections, $\mathcal{B}_3$ is an adversary against the simulation soundness property of the exponentiation proof system Exp.

Reasoning: The control components collectively decrypt $\mathtt{E2}$ to yield the partial Choice Return Codes $\mathbf{pCC}_{\mathtt{id}}$. Each adversarial control component sends a share $d_{j,i}$ together with an exponentiation proof for each selection i to the Challenger, showing that $d_{j,i} = c_{2,0}^{\mathsf{sk}_{\mathrm{CCR}_j,i}}$, where $\mathtt{E2} = (c_{2,0}, c_{2,1}, \ldots, c_{2,\psi})$. Since the honest control component internally keeps all received ballots, the Adversary cannot modify the ballot after it was verified by the honest control component in the algorithm $\mathsf{VerifyBallotCCR}_{\mathsf{h}}$ or trick the Challenger into using a ballot of another voter. Therefore, the only way in which the Adversary can alter the decryption of $\mathtt{E2}$ is by manipulating the shares $d_{j,i}$.

Therefore, the Adversary aims to send at least one (out of ψ per control component) $d_{j,i} \neq c_{2,0}^{\mathsf{sk}_{\mathrm{CCR}_j,i}}$ such that the algorithm $\mathsf{DecryptPCC}_{\mathsf{h}}$ does not detect the bogus share. However, since the algorithm $\mathsf{DecryptPCC}_{\mathsf{h}}$ verifies the exponentiation proofs, accepting a bogus share implies the generation of a valid exponentiation proof for an invalid statement, which amounts to breaking the exponentiation proof's soundness.

Reduction: In the reduction, $\mathcal{B}_3$ uses $\pi_{\mathsf{decPCC},j,i}$ and submits $((g, c_{2,0}, \mathsf{pk}_{\mathrm{CCR}_j,i}, d_{j,i}), \pi_{\mathsf{decPCC},j,i})$ as the challenge statement/proof pair (see Figure 9).

Wrapping up: Definition 14 states five conditions for a *compliant ballot*.

- The games $\mathsf{vc}.1$ and $\mathsf{vc}.2$ ensure the conditions (i) and (ii).
- Condition (iii) follows from game $\mathsf{vc}.3$ and the fact that the algorithm $\mathsf{CreateLCCShare}_{\mathsf{h}}$ checks that all partial Choice Return Codes are distinct.
- Condition (iv) is ensured by the $\mathsf{CreateLCCShare}_{\mathsf{h}}$ algorithms' check that each partial Choice Return Code has a matching entry in the partial Choice Return Codes allow list $\mathsf{L}_{\mathtt{pCC}}$.

  $$\mathtt{lpCC}_{\mathtt{id},i} \leftarrow \mathsf{RecursiveHash}(\mathtt{hpCC}_{\mathtt{id},i}^{\$}, \mathtt{vc}_{\mathtt{id}}, \mathsf{ee}, \mathsf{GetCorrectnessIdForSelectionIndex}(i)) \quad \forall i \in (1, \ldots, \psi)$$

  $$\text{Check that } \mathsf{Base64Encode}(\mathtt{lpCC}_{\mathtt{id},i}) \in \mathsf{L}_{\mathtt{pCC}} \quad \forall i \in (1, \ldots, \psi)$$

  The correctness ID included in the hash ensures that the vector of partial Choice Return Codes encodes a valid combination of voting options. Moreover, we can exclude the possibility of collisions in the random oracle model (see section 13.1).
- Condition (v) follows from conditions (i)-(iv).

Conclusion: After the game $\mathsf{vc.3}$, the Adversary is forced to submit a compliant ballot in order for the honest control component to accept it. Thus, the winning event cannot occur in this game, and therefore $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{vc.3}} = 0$.

Collecting the bounds of the hops concludes the proof.

16 Individual Verifiability

In this section, we analyze the individual verifiability property of the Swiss Post Voting System, which we defined in section 2.2.1.

Usually, by individual verifiability, we understand checks that only voters themselves can do. Section 1.2 details the checks that a voter should perform in our two-round return code scheme.

Combining the requirements and threat model for individual verifiability and the mechanisms of our return code scheme, no adversary should be able to force the trustworthy part of the system to do the following:

- accept a ballot of a non-eligible voter or discard a ballot from an eligible one;
- accept more than one ballot per voter;
- reveal information about the short Choice Return Codes $\mathbf{CC}_{\mathtt{id}}$ for options not present in the ballot or for an invalid ballot;
- reveal information about the short Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$ for an invalid confirmation attempt or an unconfirmed ballot;
- accept a ballot as confirmed which the voter has not confirmed by herself.

For simplicity, we formalize and prove the conditions separately. As such we have the following subsections:

- Ballot eligibility (section 16.1):
  we prove that an adversary cannot make an honest control component reject a compliant ballot of an eligible voter or accept a ballot of a non-eligible one;
- One ballot per voter (section 16.2):
  we prove that an adversary cannot make an honest control component accept multiple ballots from the same eligible voter;
- Sent-as-intended (section 16.3):
  we prove that the correct short Choice Return Codes $\mathbf{CC}_{\mathtt{id}}$ always correspond to the options inside the compliant ballot as recorded by the honest control component;
- Vote rejection (section 16.5):
  we prove that the correct short Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$ always corresponds to the ballot being marked as confirmed by the honest control component;
- Vote injection (section 16.7):
  we prove that the ballot cannot be confirmed without knowledge of the Ballot Casting Key $\mathtt{BCK}_{\mathtt{id}}$ (which is known only to the honest voter).

16.1 Ballot eligibility

Adversaries against the *ballot eligibility* property try to make an honest control component accept a ballot from a non-eligible voter or discard a ballot from an eligible one.

In the Swiss Post Voting System, the presence of an honest control component implies that only *compliant ballots* are accepted (see section 15). To form such a *compliant ballot* the Adversary must provide NIZK proofs using a verification card ID $\texttt{vc}_{\texttt{id}}$ and a verification card public key $\texttt{K}_{\texttt{id}}$. However, the honest control component knows all verification card IDs $\mathbf{vc}$ and verification card public keys $\mathbf{K}$ of all eligible voters and therefore would reject a ballot for a non-eligible voter. Finally, an honest control component would never reject a compliant ballot of an eligible voter.

16.2 One vote per voter

Adversaries against the *one vote per voter* property try to make an honest control component accept several votes from the same voter.

In the Swiss Post Voting System, the presence of an honest control component implies that only a single valid ballot per verification card ID $\texttt{vc}_{\texttt{id}}$, corresponding to a single voter, is accepted.

16.3 Sent-as-intended: Definition and Theorem

Adversaries against the *sent-as-intended* property try to register a ballot, where at least one voting option differs from the voting options selected by the voter. The honest voter and the honest control component do not detect the modification, although they execute their verification algorithms successfully.

As in the case of vote compliance, we analyze the Swiss Post Voting System up to the termination of the protocol SendVote, since it is the point when a ballot is marked as *sent* by the honest control component, and the honest voter verifies the short Choice Return Codes $\mathbf{CC}_{\texttt{id}}$.

We provide the adversary with all public keys and the election event context data. Moreover, the adversary knows all keys and data of untrustworthy components. We further strengthen the adversary by revealing the honest control component's CCR_h Choice Return Codes encryption secret key $\mathbf{sk}_{\text{CCR}_h}$, as our proof does not rely on the secrecy of this key. We allow the Adversary to deviate from the standard election control flow by granting access to multiple oracles: $\mathcal{O}$VoterGivesSVK, $\mathcal{O}$HonestPartialDecPCC, $\mathcal{O}$HonestLCCShare and $\mathcal{O}$VoterChecksCRC, as defined in section 13.3. Each oracle represents the actions of the honest control component or an honest voter.

In Figure 24, we present a game capturing adversaries against the sent-as-intended property of the Swiss Post Voting System. To win the game, the adversary must achieve the following conditions:

- the honest control component register the ballot,
- the ballot contains at least one voting option that differs from the voter's intention,

- the Adversary learns the the short Choice Return Code corresponding to the voter's intended voting option.

Section 2.2.1 requires a proof of individual verifiability for each *partial vote*. Therefore, we assume that the voter only checks a subset of voting options $\mathcal{S}_{\mathsf{checked}} = \{\hat{\mathrm{p}}_{\mathtt{id},1}, \ldots, \hat{\mathrm{p}}_{\mathtt{id},k}\}, k \leq \psi$
The advantage of $\mathcal{A}$ is defined as:

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

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

$$\mathsf{bad}_{\mathsf{sai}} = \begin{cases} \text{Ballot } \mathtt{b_{id}} = (\mathtt{vc_{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \pi_{\mathsf{Exp}}, \pi_{\mathsf{EqEnc}}) \text{ does not contain the voter's subset of selected voting options } \mathcal{S}_{\mathsf{checked}} \\ \quad \rho \leftarrow \mathsf{Extract}((\gamma_1, \phi_{1,0}), \mathtt{EL_{pk}}) \text{ s.t. } \exists \hat{\mathrm{p}}_{\mathtt{id},i} \in \mathcal{S}_{\mathsf{checked}}, \hat{\mathrm{p}}_{\mathtt{id},i} \nmid \rho \\ \text{Vote is marked as sent by the honest control component, i.e. } \mathtt{vc_{id}} \in \mathsf{L}_{\mathtt{sentVotes},h} \\ \text{Voter successfully verified the Choice Return Codes for } \mathcal{S}_{\mathsf{checked}}, \text{ i.e. } (\mathtt{id}, \mathsf{goodCRC}) \in \mathsf{L}_{\mathtt{VoterStatus}} \end{cases}$$

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

1. $\mathtt{id}, h \leftarrow \mathcal{A}$ // $\mathcal{A}$ specifies the target voter and the honest control component
2. $(\mathbf{CMtable}, \mathsf{L}_{\mathtt{lVCC}}, \mathsf{L}_{\mathtt{pCC}}, \mathbf{VCard}, \mathbf{VCks}, (\mathbf{sk}_{\mathrm{CCR}_h}, \mathbf{kc}_h, \mathbf{k}_h), \mathbf{pk}_{\mathrm{CCR}}, (\mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{K}_j, \mathbf{Kc}_j)_{j=1}^{m}, \mathbf{K}) \leftarrow \mathsf{SetupVoting}$
3. $\left(\mathtt{EL_{pk}}, \mathtt{EL_{sk},}_h, (\mathtt{EL_{pk},}_j)_{j=1}^{m}, \mathtt{EB_{sk}}, \mathtt{EB_{pk}}\right) \leftarrow \mathsf{SetupTally}$
4. If VerifyConfigPhase fails, abort and output 0. // Verify the configuration phase—see section 11.4.
5. Initialize an empty voter status list $\mathsf{L}_{\mathtt{VoterStatus}}$.
6. Set $\mathbf{VCard}^* \leftarrow \mathbf{VCard} \setminus \mathsf{VCard}_{\mathtt{id}}$ // Do not give the voting card of the target voter to $\mathcal{A}$.
7. Give the following data to $\mathcal{A}$:

$$\left(\mathbf{VCard}^*, \mathbf{VCks}, \mathbf{CMtable}, \mathsf{L}_{\mathtt{lVCC}}, \mathsf{L}_{\mathtt{pCC}}, \mathtt{EL_{pk}}, \mathbf{pk}_{\mathrm{CCR}}, (\mathtt{EL_{pk},}_j, \mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{K}_j, \mathbf{Kc}_j)_{j=1}^{m}, \mathbf{K}, \mathtt{EB_{sk}}, \mathtt{EB_{pk}}\right)$$

8. Give the following key to $\mathcal{A}$: // sent-as-intended does not rely on the secrecy of this key

$$(\mathbf{sk}_{\mathrm{CCR}_h})$$

. 9. The Adversary $\mathcal{A}$ is allowed to query the oracles $\mathcal{O}$VoterGivesSVK, $\mathcal{O}$HonestPartialDecPCC, $\mathcal{O}$HonestLCCShare and $\mathcal{O}$VoterChecksCRC defined in figure 19 during the protocol SendVote.
10. If $\mathsf{bad}_{\mathsf{sai}}$ output 1

Figure 24: Game modeling attacks against the sent-as-intended property of the Swiss Post Voting System. The protocols SetupVoting and SetupTally are executed in interaction with the adversary $\mathcal{A}$. The public parameters `context` are implicit.

Theorem 1 *In the random oracle model*

$$Adv_{\mathcal{A}}^{\mathsf{SPVS,sai}} \leq Adv_{\mathcal{B}_1}^{\mathsf{SPVS,vc}} + 2 \cdot \mathtt{N_E} \cdot Adv_{\mathcal{B}_2}^{\mathsf{Exp,NIZK}} + Adv_{\mathcal{B}_3}^{\mathsf{SPVS,mtc}} + \psi \cdot Adv_{\mathcal{B}_4}^{\mathsf{SEnc,INDCPA}} + \left(\frac{1}{|\mathcal{C}_{cc}| - \psi}\right)$$

where $\mathcal{B}_1$ is an adversary against the vote compliance property of the Swiss Post Voting System*, $\mathcal{B}_2$ is an adversary against the zero-knowledge property of the exponentiation proof system, $\mathcal{B}_3$ is an adversary*

against the Return Codes Mapping table **CMtable** *correctness property of the* Swiss Post Voting System*,* $\mathcal{B}_4$ *is an adversary against the semantic security of the symmetric encryption scheme,* $\mathcal{C}_{cc}$ *is the space of short Choice Return Codes,* $\mathtt{N_E}$ *is the number of voters and* ψ *is the number of voting options a voter can select.*

16.4 Sent-as-intended: Proof of Theorem 1

We prove that the Adversary cannot undetectably modify an honest voter's vote, i.e., the event $\mathsf{bad_{sai}}$ is unreachable.

We start with the original Swiss Post Voting System scheme and show that we can randomize all values related to the expected short Choice Return Codes without the Adversary noticing. Once all values are randomized, the Adversary is given no information about the expected short Choice Return Codes.

Step 8 of figure 24 strengthens the Adversary by exposing the CCR_h Choice Return Codes encryption secret key $\mathbf{sk}_{\text{CCR}_h}$, demonstrating that the sent-as-intended property does not rely on the secrecy of this key. The Adversary can verify the correctness of the values that depend on the CCR_h Choice Return Codes encryption secret key $\mathbf{sk}_{\text{CCR}_h}$, i.e., $\mathbf{d}_{h,\mathtt{id}}, \boldsymbol{\pi}_{\mathsf{decPCC},h}$.

Figure 25 depicts the values that are relevant for deriving the short Choice Return Codes $\mathbf{CC}_{\mathtt{id}}$.

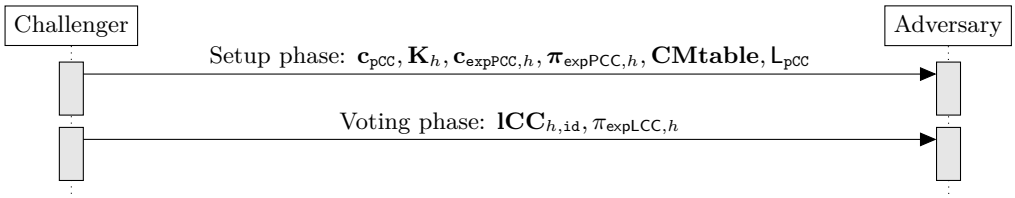

Figure 25: The part of the Challenger's output associated with retrieving the expected short Choice Return Codes $\mathbf{CC}_{\mathtt{id}}$ in the sent-as-intended game.

We gradually randomize all values to show that the Adversary cannot learn any information about the short Choice Return Codes and that the event $\mathsf{bad_{sai}}$ is unreachable. The oracle $\mathcal{O}$VoterChecksCRC will verify the Adversary's Choice Return Codes only once (see figure 19). Therefore, the Adversary has only one attempt to submit the correct short Choice Return Codes.

Game sai.1

Objective: In this game, the Challenger aborts if the Adversary submits a non-compliant ballot.

Adjustments: The Challenger repeats all adjustments from the vote compliance game hops vc.1 – vc.3.

Objective:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,sai.1}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,sai}}| = \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{SPVS,vc}}$$

where $\mathcal{B}_1$ is an adversary against the vote compliance property of the Swiss Post Voting System.

Reasoning: The Adversary notices the change only if the honest control component accepts a non-compliant ballot. Section 14.3 demonstrates that this happens only with negligible probability.

Game sai.2

Objective: In this game, the Challenger randomizes the honest control component's public keys, thereby preparing the switch to IdealSetupVoting.

Adjustments: Algorithms $\mathsf{CreateLCCShare_h}$ and $\mathsf{CreateLVCCShare_h}$ (executed in oracles $\mathcal{O}$HonestLCCShare and $\mathcal{O}$HonestCreateLVCCShare from Fig. 19 and 20) will use the canonical simulator $\mathcal{S}_{\mathsf{Exp}}$ to generate the proofs $\pi_{\mathsf{expLCC},h}$ and $\pi_{\mathsf{expLVCC},h}$ containing the Voter Choice Return Code Generation private key $\mathtt{k}_{h,\mathtt{id}}$ and the Voter Vote Cast Return Code Generation private key $\mathtt{kc}_{h,\mathtt{id}}$ of the honest control component.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,sai.1}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,sai.2}}| \leq 2 \cdot \mathtt{N_E} \cdot \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{Exp,NIZK}}$$

where $\mathtt{N_E}$ is the number of voters and $\mathcal{B}_2$ is an adversary against the zero-knowledge property of the Exp proof system.

Reasoning: The idealized protocol IdealSetupVoting randomizes the honest control component public keys $\mathtt{K}_{h,\mathtt{id}}$ and $\mathtt{Kc}_{h,\mathtt{id}}$. However, in the voting phase, those keys are still required for generating the proofs $\pi_{\mathsf{expLCC},h}$ and $\pi_{\mathsf{expLVCC},h}$ in the oracles $\mathcal{O}$HonestLCCShare and $\mathcal{O}$HonestCreateLVCCShare (see Fig. 19 and 20). Therefore, before running IdealSetupVoting, we first need to simulate the remaining proofs.

The Adversary notices the change only by distinguishing between the real and simulated proofs (up to $\mathtt{N_E}$ for each of the two proofs) of correct exponentiation.

Effect on Future Games: From this game onwards, we no longer require correct public keys $\mathtt{K}_{h,\mathtt{id}}$ and $\mathtt{Kc}_{h,\mathtt{id}}$ in the voting phase, hence we can switch to IdealSetupVoting.

Game sai.3

Objective: In this game, the Challenger runs IdealSetupVoting instead of SetupVoting.

Adjustments: IdealSetupVoting substitutes all exponentiations done by the honest control component with randomly sampled elements from $\mathbb{Q}_p$. Therefore, IdealSetupVoting replaces $\left(\mathtt{hpCC}_{\mathtt{id},i}\right)^{\mathtt{k}_{h,\mathtt{id}}}$ and $\left(\mathtt{hCK}_{\mathtt{id}}\right)^{\mathtt{kc}_{h,\mathtt{id}}}$ with $\{x_{i,\mathtt{id}}\}_{i=1}^{n}$ and $y_{\mathtt{id}}$, where all $x_{i,\mathtt{id}}$ and $y_{\mathtt{id}}$ are randomly sampled from $\mathbb{Q}_p$.

To ensure that the Choice Return Codes and the Vote Cast Return Codes of all voters are extractable, the Challenger needs to use the same random elements in the voting phase (i.e. adjust $\mathsf{CreateLCCShare_h}$ and $\mathsf{CreateLVCCShare_h}$ functions in $\mathcal{O}\mathsf{HonestLCCShare}$ and $\mathcal{O}\mathsf{HonestCreateLVCCShare}$ oracles from Fig. 19 and 20).

Concretely, the Challenger keeps an internal table where it stores inputs and outputs of the $\mathsf{IdealSetupVoting}$ protocol:

$$\left\{\left[\mathtt{hpCC}_{\mathtt{id},i} = \mathsf{HashAndSquare}(\mathrm{p}_i^{\mathtt{k_{id}}}); x_{i,\mathtt{id}}\right]\right\}_{i=1}^{n}, \left[\mathtt{hCK}_{\mathtt{id}} = \mathsf{HashAndSquare}(\mathtt{CK}_{\mathtt{id}}); y_{\mathtt{id}}\right]$$

Then, in the voting phase, the Challenger uses this internal table in the functions $\mathsf{CreateLCCShare_h}$ and $\mathsf{CreateLVCCShare_h}$ to reply to the queries. For example in $\mathsf{CreateLCCShare_h}$, whenever an honest CCR_h receives $\mathtt{pCC}_{\mathtt{id},x} = \mathrm{p}_x^{\mathtt{k_{id}}}$, it computes $\mathsf{HashAndSquare}(\mathtt{pCC}_{\mathtt{id},x})$ and checks if the resulting value is present in the internal table. If this is the case, it returns the corresponding $x_{x,\mathtt{id}}$, otherwise it randomly samples a value $x_{x,\mathtt{id}}$ from $\mathbb{Q}_p$ and adds the new pair $\left[\mathsf{HashAndSquare}(\mathtt{pCC}_{\mathtt{id},x}); x_{x,\mathtt{id}}\right]$ to the internal table. Equivalent logic applies to $\mathsf{CreateLVCCShare_h}$.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,sai.2}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,sai.3}}| = \mathsf{Adv}_{\mathcal{B}_3}^{\mathsf{SPVS,mtc}}$$

where $\mathcal{B}_3$ is an adversary against the Return Codes Mapping table **CMtable** correctness property of the Swiss Post Voting System.

Reasoning: The Adversary notices the change only by distinguishing between $\mathsf{SetupVoting}$ and $\mathsf{IdealSetupVoting}$, which was proven to be difficult in Lemma 1. After this hop, the view of the Adversary changes to the one in Fig. 26.

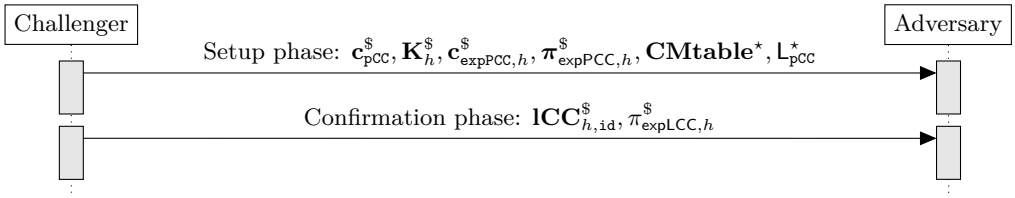

Figure 26: The part of the Challenger's output associated with retrieving the expected short Choice Return Codes $\mathbf{CC}_{\mathtt{id}}$ in the sent-as-intended game after the switch to $\mathsf{IdealSetupVoting}$. The randomly selected or simulated values are marked with \$ sign. The correct Return Codes Mapping table **CMtable** and the $\mathsf{L_{pCC}}$ allow list are marked with $\star$ sign.

Game sai.4

Objective: In this game, the Challenger randomizes the right-hand side of the Return Codes Mapping Table $\mathbf{CMtable}^{\star}$ containing the symmetric encryption of the expected short Choice Return Codes

$\mathbf{CC}_{\mathtt{id}}$. The Challenger limits the randomization to the subset of voting options that the voter will check ($\mathcal{S}_{\mathsf{checked}} = \{\hat{\mathrm{p}}_{\mathtt{id},1}, \ldots, \hat{\mathrm{p}}_{\mathtt{id},k}\}, k \leq \psi$).

Adjustments: For winning the sent-as-intended game, the Adversary must modify at least one voting option $\hat{\mathrm{p}}_{\mathtt{id},i} \in \mathcal{S}_{\mathsf{checked}}$. The Challenger adjusts the algorithm $\mathsf{GenCMTable}^{\$}$ to symmetrically encrypt a random message $\mathtt{xCC}_{\mathtt{id},i} \xleftarrow{\$} \mathcal{M}$ instead of the actual short Choice Return Code $\mathtt{CC}_{\mathtt{id},}$ corresponding to $\hat{\mathrm{p}}_{\mathtt{id},i}$. The Challenger performs this substitution for all voting options that the voter verifies: $\forall \hat{\mathrm{p}}_{\mathtt{id},i} \in \mathcal{S}_{\mathsf{checked}}$.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,sai.3}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,sai.4}}| \leq \psi \cdot \mathsf{Adv}_{\mathcal{B}_4}^{\mathsf{SEnc,INDCPA}}$$

where ψ is the number of allowed selections and $\mathcal{B}_4$ an adversary against the semantic security of the symmetric encryption scheme.

Reasoning: After the switch to $\mathsf{IdealSetupVoting}$, the honest control component's share $\mathtt{lCC}_{\mathtt{id},h}$ is sampled at random (see Fig. 21). In the Random Oracle Model (see section 13.1), the Adversary learns nothing about the Choice Return Code encryption symmetric key $\mathtt{skcc}^{\$}_{\mathtt{id},i}$ that is symmetrically encrypting the short Choice Return Code $\mathtt{CC}_{\mathtt{id},i}$.

$$\mathsf{Enc}_{\mathsf{s}}(\mathtt{CC}_{\mathtt{id},i}; \mathtt{skcc}^{\$}_{\mathtt{id},i})$$

Since the Adversary can query the oracle $\mathcal{O}\mathsf{HonestLCCShare}$ only once per voter (see Fig. 19), distinguishing the two games amounts to breaking the IND-CPA property of the symmetric encryption scheme. The factor of ψ in the bound follows from the fact that there are ψ short Choice Return Codes per voter.

Therefore we can directly apply the IND-CPA property of the symmetric encryption (up to n times) and change the plaintext from $\mathtt{CC}_{\mathtt{id},\hat{s}_i}$ to $\mathtt{xCC}_{\mathtt{id},\hat{s}_i}$ for each $\forall i \in \mathcal{S}_{\mathsf{checked}}$.

Wrapping up: In the $\mathsf{sai}.4$ game, the Adversary is not given any information about the short Choice Return Codes $\mathbf{CC}_{\mathtt{id}}$. All values related to the extraction of the expected short Choice Return Codes $\mathbf{CC}_{\mathtt{id}}$ are either selected at random (marked with a \$ sign) or an output of the random oracle (e.g. $\mathsf{L}^{\star}_{\mathtt{pCC}}$). Therefore, the Adversary learns no meaningful information, except that the short Choice Return Code for each voting option corresponding to $\hat{\mathrm{p}}_{\mathtt{id},i} \in \mathcal{S}_{\mathsf{checked}}$ differs from the ψ short Choice Return Codes that the Adversary can extract from the voter's ballot. Hence, the Adversary can only guess the short Choice Return Codes randomly. Given that the highest probability of a correct guess correctly is for a single Choice Return Code, we have that the probability to win the $\mathsf{sai}.4$ game is at most:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,sai.4}} \leq \left(\frac{1}{|\mathcal{C}_{cc}| - \psi}\right)$$

Collecting the bounds of the hops concludes the proof.

16.5 Vote rejection: Definition and Theorem

In this type of attacks, an honest voter sends and confirms her vote. The Adversary attempts to avoid the registration of the confirmation at the honest control component, resulting in the exclusion of the vote in the tally (the honest control component only mixes and decrypts confirmed votes). The honest voter and the honest control component do not detect the modification, although they follow their respective verification algorithms. In Figure 27, we define the game rac-rej, capturing adversaries against vote rejection. The advantage of the Adversary $\mathcal{A}$ is defined as:

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

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

$$\mathsf{bad}_{\mathsf{rej}} = \begin{cases} \text{vote is not recorded as confirmed by the honest control component i.e. } \mathtt{vc}_{\mathtt{id}} \notin \mathsf{L}_{\mathtt{confirmedVotes},h} \\ \text{voter has successfully verified } \mathtt{VCC}_{\mathtt{id}} \text{ i.e. } (\mathtt{id}, \mathsf{goodVCC}) \in \mathsf{L}_{\mathtt{VoterStatus}} \end{cases}$$

To model vote rejection, we allow the adversary to query all oracles of the SendVote and ConfirmVote procedures.

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

1. $\mathtt{id}, h \leftarrow \mathcal{A}$ // The Adversary $\mathcal{A}$ specifies the target voter and the honest control component.
2. $(\mathbf{CMtable}, \mathsf{L}_{\mathtt{lVCC}}, \mathsf{L}_{\mathtt{pCC}}, \mathbf{VCard}, \mathbf{VCks}, (\mathbf{sk}_{\mathrm{CCR}_h}, \mathbf{kc}_h, \mathbf{k}_h), \mathbf{pk}_{\mathrm{CCR}}, (\mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{K}_j, \mathbf{Kc}_j)_{j=1}^{m}, \mathbf{K}) \leftarrow \mathsf{SetupVoting}$
3. $\left(\mathtt{EL}_{\mathtt{pk}}, \mathtt{EL}_{\mathtt{sk},h}, (\mathtt{EL}_{\mathtt{pk},j})_{j=1}^{m}, \mathtt{EB}_{\mathtt{sk}}, \mathtt{EB}_{\mathtt{pk}}\right) \leftarrow \mathsf{SetupTally}$
4. If VerifyConfigPhase fails, abort and output 0. // Verify the configuration phase—see section 11.4.
5. Initialize an empty voter status list $\mathsf{L}_{\mathtt{VoterStatus}}$.
6. Set $\mathbf{VCard}^* \leftarrow \mathbf{VCard} \setminus \mathsf{VCard}_{\mathtt{id}}$ // Do not give the voting card of the target voter to the Adversary.
7. Give the following data to $\mathcal{A}$:

$$\left(\mathbf{VCard}^*, \mathbf{VCks}, \mathbf{CMtable}, \mathsf{L}_{\mathtt{lVCC}}, \mathsf{L}_{\mathtt{pCC}}, \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, (\mathtt{EL}_{\mathtt{pk},j}, \mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{K}_j, \mathbf{Kc}_j)_{j=1}^{m}, \mathbf{K}, \mathtt{EB}_{\mathtt{sk}}, \mathtt{EB}_{\mathtt{pk}}\right)$$

8. The Adversary is allowed to query the oracles $\mathcal{O}$VoterGivesSVK, $\mathcal{O}$HonestPartialDecPCC, $\mathcal{O}$HonestLCCShare and $\mathcal{O}$VoterChecksCRC of the SendVote protocol as defined in figure 19. Further, the Adversary is allowed to query the oracles $\mathcal{O}$VoterGivesBCK, $\mathcal{O}$HonestCreateLVCCShare, $\mathcal{O}$HonestVerifyLVCCShares and $\mathcal{O}$VoterChecksVCC of the ConfirmVote protocol as defined in figure 20.
9. If $\mathsf{bad}_{\mathsf{rej}}$ output 1

Figure 27: Game modeling attacks against the recorded as confirmed (rejection) property of the Swiss Post Voting System. The protocols SetupVoting and SetupTally are executed in interaction with the adversary $\mathcal{A}$. The public parameters context are implicit.

Theorem 2 *In the random oracle model,*

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

where $\mathtt{N_E}$ *is the number of voters,* $\mathcal{B}_1$ *is an adversary against the zero-knowledge property of the exponentiation proof system* Exp*, respectively,* $\mathcal{B}_2$ *is an adversary against the Return Codes Mapping table* **CMtable** *correctness property of the* Swiss Post Voting System*,* $\mathcal{B}_3$ *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.*

16.6 Vote rejection: Proof of Theorem 2

In this section, we prove that the Adversary cannot reject an honest voter's vote i.e. the $\mathsf{bad}_{\mathsf{rej}}$ event is unreachable.

In the proof, we start with the original Swiss Post Voting System scheme and show that we can randomize all values related to the expected short Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$. Once all values are randomized, the Adversary is given no information at all anymore about the expected $\mathtt{VCC}_{\mathtt{id}}$, and therefore has no other option than to guess it.

The Adversary can get information about the Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$ only during the Return Codes Mapping table **CMtable** generation or the vote confirmation; no other part of the configuration or voting phase deals with the $\mathtt{VCC}_{\mathtt{id}}$ value. Figure 28 depicts the data that depends on the Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$.

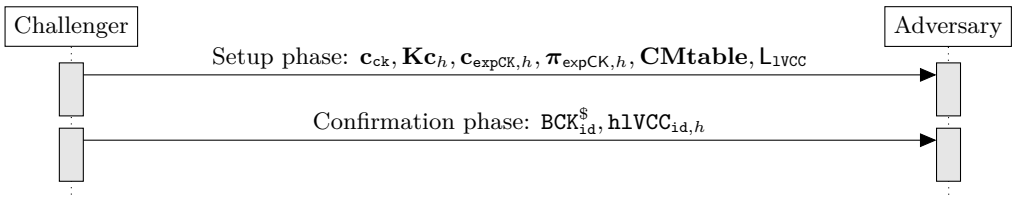

Figure 28: The part of the Challenger's output associated with retrieving the expected short Vote Cast Return Code in the vote rejection game. The randomly selected or simulated values are marked with a $ sign.

Whenever the oracle $\mathcal{O}$HonestVerifyLVCCShares returns $\mathtt{lVCC}_{\mathtt{id},j}$, it marks the ballot with $\mathtt{vc}_{\mathtt{id}}$ as confirmed, i.e., $\mathtt{vc}_{\mathtt{id}} \in \mathsf{L}_{\mathtt{confirmedVotes},h}$ (see Fig. 20). This implies that the Adversary loses the vote rejection game, as one of the winning event $\mathsf{bad}_{\mathsf{rej}}$ conditions requires $\mathtt{vc}_{\mathtt{id}} \notin \mathsf{L}_{\mathtt{confirmedVotes},h}$.

Game rac-rej.1

Objective: In this game, the Challenger randomizes the honest control component's public keys, thereby preparing the switch to IdealSetupVoting.

Adjustments: Algorithms $\mathsf{CreateLCCShare}_{\mathsf{h}}$ and $\mathsf{CreateLVCCShare}_{\mathsf{h}}$ (executed in oracles $\mathcal{O}$HonestLCCShare and $\mathcal{O}$HonestCreateLVCCShare from Fig. 19 and 20) will use the canonical simulator $\mathcal{S}_{\mathsf{Exp}}$ to generate the

proofs $\pi_{\mathsf{expLCC},h}$ and $\pi_{\mathsf{expLVCC},h}$ containing the Voter Choice Return Code Generation private key $\mathtt{k}_{h,\mathtt{id}}$ and the Voter Vote Cast Return Code Generation private key $\mathtt{kc}_{h,\mathtt{id}}$ of the honest control component.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{rac\text{-}rej}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{rac\text{-}rej}.1}| \leq \mathtt{N_E} \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{Exp},\mathsf{NIZK}}$$

where $\mathtt{N_E}$ is the number of voters and $\mathcal{B}_1$ is an adversary against the zero-knowledge property of the Exp proof system.

Reasoning: The idealized protocol IdealSetupVoting randomizes the honest control component public keys $\mathtt{K}_{h,\mathtt{id}}$ and $\mathtt{Kc}_{h,\mathtt{id}}$. However, in the voting phase, those keys are still required for generating the proofs $\pi_{\mathsf{expLCC},h}$ and $\pi_{\mathsf{expLVCC},h}$ in the oracles $\mathcal{O}$HonestLCCShare and $\mathcal{O}$HonestCreateLVCCShare (see Fig. 19 and 20). Therefore, before running IdealSetupVoting, we first need to simulate the remaining proofs.

The Adversary notices the change only by distinguishing between the real and simulated proofs (up to $\mathtt{N_E}$ for each of the two proofs) of correct exponentiation.

Effect on Future Games: From this game onwards, we no longer require correct public keys $\mathtt{K}_{h,\mathtt{id}}$ and $\mathtt{Kc}_{h,\mathtt{id}}$ in the voting phase, hence we can switch to IdealSetupVoting.

Game rac-rej.2

Objective: In this game, the Challenger runs IdealSetupVoting instead of SetupVoting.

Adjustments: IdealSetupVoting substitutes all exponentiations done by the honest control component with randomly sampled elements from $\mathbb{Q}_p$. Therefore, IdealSetupVoting replaces $\left(\mathtt{hpCC}_{\mathtt{id},i}\right)^{\mathtt{k}_{h,\mathtt{id}}}$ and $\left(\mathtt{hCK}_{\mathtt{id}}\right)^{\mathtt{kc}_{h,\mathtt{id}}}$ with $\{x_{i,\mathtt{id}}\}_{i=1}^{n}$ and $y_{\mathtt{id}}$, where all $x_{i,\mathtt{id}}$ and $y_{\mathtt{id}}$ are randomly sampled from $\mathbb{Q}_p$.

To ensure that the Choice Return Codes and the Vote Cast Return Code of all voters are extractable, the Challenger needs to use the same random elements in the voting phase (i.e. adjust $\mathsf{CreateLCCShare}_{\mathsf{h}}$ and $\mathsf{CreateLVCCShare}_{\mathsf{h}}$ functions in $\mathcal{O}$HonestLCCShare and $\mathcal{O}$HonestCreateLVCCShare oracles from Fig. 19 and 20).

Concretely, the Challenger keeps an internal table where it stores inputs and outputs of the IdealSetupVoting protocol:

$$\left\{\left[\mathtt{hpCC}_{\mathtt{id},i} = \mathsf{HashAndSquare}(\mathrm{p}_i^{\mathtt{k}_{\mathtt{id}}}); x_{i,\mathtt{id}}\right]\right\}_{i=1}^{n}, \left[\mathtt{hCK}_{\mathtt{id}} = \mathsf{HashAndSquare}(\mathtt{CK}_{\mathtt{id}}); y_{\mathtt{id}}\right]$$

Then, in the voting phase, the Challenger uses this internal table in the functions $\mathsf{CreateLCCShare}_{\mathsf{h}}$ and $\mathsf{CreateLVCCShare}_{\mathsf{h}}$ to reply to the queries. For example in $\mathsf{CreateLCCShare}_{\mathsf{h}}$, whenever an honest CCR_h receives $\mathtt{pCC}_{\mathtt{id},x} = \mathrm{p}_x^{\mathtt{k}_{\mathtt{id}}}$, it computes $\mathsf{HashAndSquare}(\mathtt{pCC}_{\mathtt{id},x})$ and checks if the resulting value is present in the internal table. If this is the case, it returns the corresponding $x_{x,\mathtt{id}}$, otherwise it randomly

samples a value $x_{x,\mathtt{id}}$ from $\mathbb{Q}_p$ and adds the new pair $\left[\mathsf{HashAndSquare}(\mathtt{pCC}_{\mathtt{id},x}); x_{x,\mathtt{id}}\right]$ to the internal table. Equivalent logic applies to $\mathsf{CreateLVCCShare}_{\mathsf{h}}$.

Claim:

$$|\mathsf{Adv}^{\mathsf{SPVS,rac\text{-}rej.1}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{SPVS,rac\text{-}rej.2}}_{\mathcal{A}}| = \mathsf{Adv}^{\mathsf{SPVS,mtc}}_{\mathcal{B}_2}$$

where $\mathcal{B}_2$ is an adversary against the Return Codes Mapping table **CMtable** correctness property of the Swiss Post Voting System.

Reasoning: The Adversary notices the change only by distinguishing between SetupVoting and IdealSetupVoting, which was proven to be difficult in Lemma 1. After this hop, the view of the Adversary changes to the one in Fig. 29.

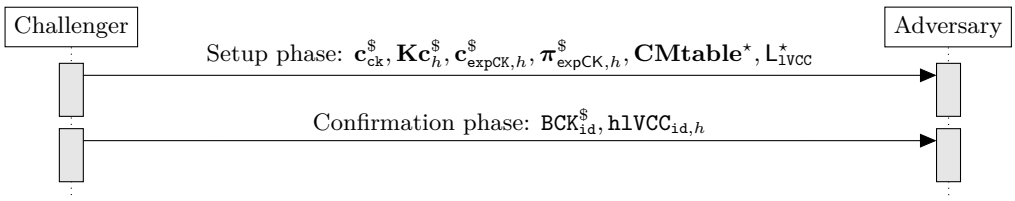

Figure 29: The part of the Challenger's output associated with retrieving the expected short Vote Cast Return Code after the switch to IdealSetupVoting. The randomly selected or simulated values are marked with a $ sign. The correct Return Codes Mapping table **CMtable** and the $\mathsf{L}_{\mathtt{lVCC}}$ allow list are marked with a $\star$ sign.

Game rac-rej.3

Objective: In this game, the Challenger randomizes the right-hand side of the Return Codes Mapping Table $\mathbf{CMtable}^{\star}$ containing the symmetric encryption of the expected short Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$.

Adjustments: The Challenger adjusts the algorithm $\mathsf{GenCMTable}^{\$}$ to symmetrically encrypt a random message $\mathtt{xVCC}_{\mathtt{id}} \xleftarrow{\$} \mathcal{M}$ instead of the actual short Vote Cast Return Code $\mathtt{VCC}_{\mathtt{id}}$. However, the Challenger uses $\mathtt{VCC}_{\mathtt{id}}$ for printing the honest voter's voting cards.

Claim:

$$|\mathsf{Adv}^{\mathsf{SPVS,rac\text{-}rej.2}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{SPVS,rac\text{-}rej.3}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{SEnc,INDCPA}}_{\mathcal{B}_3}$$

where $\mathcal{B}_3$ an adversary against the semantic security of the symmetric encryption scheme.

Reasoning: After the switch to IdealSetupVoting, the honest control component's share $\texttt{lVCC}_{\texttt{id},h}$ is sampled at random (see Fig. 21). In the Random Oracle Model (see section 13.1), the Adversary learns nothing about the Vote Cast Return Code encryption symmetric key $\texttt{skvcc}^{\$}_{\texttt{id}}$ that is symmetrically encrypting the short Vote Cast Return Code $\texttt{VCC}_{\texttt{id}}$.

$$\mathsf{Enc}_{\mathsf{s}}(\texttt{VCC}_{\texttt{id}}; \texttt{skvcc}^{\$}_{\texttt{id}})$$

Whenever the oracle $\mathcal{O}$HonestVerifyLVCCShares returns $\texttt{lVCC}_{\texttt{id},j}$, it marks the ballot with $\texttt{vc}_{\texttt{id}}$ as confirmed (see Fig. 20). Therefore, the Adversary cannot obtain all shares $\{\mathsf{VerifyLVCCHash_j}\}_{j=1}^{m}$ and learn the Vote Cast Return Code encryption symmetric key $\texttt{skvcc}_{\texttt{id}}$ to decrypt the short Vote Cast Return Code$\texttt{VCC}_{\texttt{id}}$. This means that, even with the knowledge of the Return Codes Mapping table **CMtable**, the Adversary has no information related to $\texttt{skvcc}_{\texttt{id}}$. Therefore we can directly apply the IND-CPA property of the symmetric encryption and change the plaintext from $\texttt{VCC}_{\texttt{id}}$ to $\texttt{xVCC}_{\texttt{id}}$.

After this hop, the view of the Adversary changes to the one in Fig. 30.

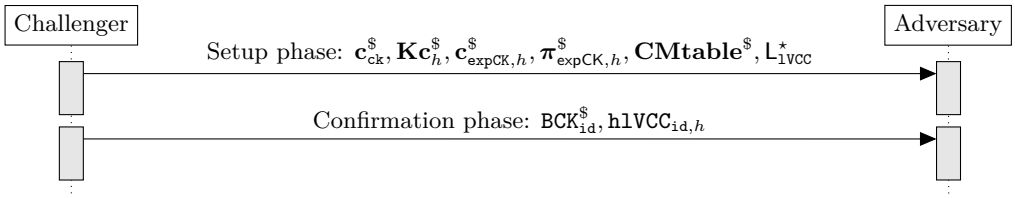

Figure 30: The part of the Challenger's output associated with retrieving the expected short Vote Cast Return Code after randomizing the Return Codes Mapping table **CMtable**. The randomly selected or simulated values are marked with a $ sign.

Wrapping up: In the rac-rej.3 game the Adversary is not given any information about the short Vote Cast Return Code $\texttt{VCC}_{\texttt{id}}$. All values related to the expected short Vote Cast Return Code are either selected at random (marked with a $ sign) or an output of the random oracle (e.g. $\texttt{hlVCC}_{\texttt{id},h}, \mathsf{L}^{\star}_{\texttt{lVCC}}$). Therefore, the Adversary cannot extract any meaningful information and can only guess the Vote Cast Return Code $\texttt{VCC}_{\texttt{id}}$ randomly. Further, the adversary only has a single attempt, as the honest voters abort if they see an invalid $\texttt{VCC}_{\texttt{id}}$. Thus,

$$\mathsf{Adv}^{\mathsf{SPVS,rac\text{-}rej.3}}_{\mathcal{A}} \leq \frac{1}{|\mathcal{C}_{vcc}|}$$

Collecting the bounds of the hops concludes the proof.

16.7 Vote injection: Definition and Theorem

In this type of attack, an honest voter sends a vote (executing the SendVote protocol), but then aborts the process and does *not* confirm the vote. Therefore, the voter expects that the vote is *not* included in the tally since the honest control component only tallies confirmed votes. However, the adversary forces

the registration of the non-confirmed vote resulting in inserting the vote in final tally. In Figure 31, we define game rac-inj capturing adversaries against vote injection. The advantage of $\mathcal{A}$ is defined as:

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

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

$$\mathsf{bad}_{\mathsf{inj}} = \begin{cases} \text{voter did not confirm the vote} \\ \text{vote is recorded as confirmed by the honest control component i.e. } \mathtt{vc}_{\mathtt{id}} \in \mathsf{L}_{\mathtt{confirmedVotes},h} \end{cases}$$

To model the vote injection, we allow the adversary to query all oracles of SendVote and ConfirmVote procedures, except for $\mathcal{O}$VoterChecksVCC and $\mathcal{O}$VoterGivesBCK (as a voter unwilling to confirm the vote does not participate in the confirmation phase).

rac-inj$_{\mathcal{A}}$(context)

1. $\mathtt{id}, h \leftarrow \mathcal{A}$ // The Adversary $\mathcal{A}$ specifies the target voter and the honest control component.
2. $(\mathbf{CMtable}, \mathsf{L}_{\mathtt{lVCC}}, \mathsf{L}_{\mathtt{pCC}}, \mathbf{VCard}, \mathbf{VCks}, (\mathbf{sk}_{\mathrm{CCR}_h}, \mathbf{kc}_h, \mathbf{k}_h), \mathbf{pk}_{\mathrm{CCR}}, (\mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{K}_j, \mathbf{Kc}_j)_{j=1}^{m}, \mathbf{K}) \leftarrow$ SetupVoting
3. $(\mathtt{EL}_{\mathtt{pk}}, \mathtt{EL}_{\mathtt{sk},h}, (\mathtt{EL}_{\mathtt{pk},j})_{j=1}^{m}, \mathtt{EB}_{\mathtt{sk}}, \mathtt{EB}_{\mathtt{pk}}) \leftarrow$ SetupTally
4. If VerifyConfigPhase fails, abort and output 0. // Verify the configuration phase—see section 11.4.
5. Initialize an empty voter status list $\mathsf{L}_{\mathtt{VoterStatus}}$.
6. Set $\mathbf{VCard}^* \leftarrow \mathbf{VCard} \backslash \mathrm{VCard}_{\mathtt{id}}$ // Do not give the voting card of the target voter to the Adversary.
7. Give the following data to the Adversary:

$$\left(\mathbf{VCard}^*, \mathbf{VCks}, \mathbf{CMtable}, \mathsf{L}_{\mathtt{lVCC}}, \mathsf{L}_{\mathtt{pCC}}, \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, (\mathtt{EL}_{\mathtt{pk},j}, \mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{K}_j, \mathbf{Kc}_j)_{j=1}^{m}, \mathbf{K}, \mathtt{EB}_{\mathtt{sk}}, \mathtt{EB}_{\mathtt{pk}}\right)$$

8. The Adversary is allowed to query SendVote oracles $\mathcal{O}$VoterGivesSVK, $\mathcal{O}$HonestPartialDecPCC, $\mathcal{O}$HonestLCCShare and $\mathcal{O}$VoterChecksCRC as defined in figure 19. Further, the Adversary is allowed to query ConfirmVote oracles $\mathcal{O}$HonestCreateLVCCShare and $\mathcal{O}$HonestVerifyLVCCShares as defined in figure 20 (excluding $\mathcal{O}$VoterGivesBCK and $\mathcal{O}$VoterChecksVCC, as per our setting the voter is not willing to confirm the vote).
9. If $\mathsf{bad}_{\mathsf{inj}}$ output 1

Figure 31: Game modeling attacks against recorded as confirmed (injection) property of the Swiss Post Voting System. Protocols SetupVoting and SetupTally are executed in interaction with the adversary $\mathcal{A}$. The public parameters context are implicit.

Theorem 3 *In the random oracle model*

$$Adv_{\mathcal{A}}^{\mathsf{SPVS,rac\text{-}inj}} \leq 2 \cdot \mathtt{N_E} \cdot Adv_{\mathcal{B}_1}^{\mathsf{Exp,NIZK}} + Adv_{\mathcal{B}_2}^{\mathsf{SPVS,mtc}} + \mathtt{maxConfAttempts} \cdot (\frac{1}{|\mathcal{C}_{bck}|})$$

where $\mathtt{N_E}$ *is the number of voters,* $\mathcal{B}_1$ *is an adversary against simulation sound property of the Exponentiation proof system* Exp*,* $\mathcal{B}_2$ *is an adversary against the Return Codes Mapping table* **CMtable** *correctness property of the* Swiss Post Voting System*,* $\mathcal{C}_{bck}$ *is the space of Ballot Casting Keys values, and* maxConfAttempts *is the maximum number of attempts to submit the Confirmation Key.*

16.8 Vote injection: Proof of Theorem 3

In this section, we prove that the Adversary cannot inject a vote i.e. $\mathsf{bad}_{\mathsf{inj}}$ event is unreachable.

In the proof, we start with the original Swiss Post Voting System scheme and show that we can randomize all values related to the expected Ballot Casting Key $\mathtt{BCK}_{\mathtt{id}}$. Once we reach this state, the Adversary has no information to extract the expected $\mathtt{BCK}_{\mathtt{id}}$, and therefore has no better strategy than to guess it.

Recall that the ballot is marked as confirmed only if the oracle $\mathcal{O}$HonestVerifyLVCCShares does not abort (see Fig. 20). It implies that $\mathtt{hhlVCC}_{\mathtt{id}} \in \mathsf{L}_{\mathtt{lVCC}}$ (see function $\mathsf{VerifyLVCCHash}_{\mathsf{j}}$ for details, section 5.2.3 of [37]). In the random oracle model collisions do not happen, hence verification passes only when the honest control component receives the expected $\mathtt{CK}_{\mathtt{id}}$. Therefore, the ballot will be marked as confirmed only if the Adversary uses the correct $\mathtt{CK}_{\mathtt{id}}$ i.e gets the expected $\mathtt{BCK}_{\mathtt{id}}$.

The Adversary can get information about $\mathtt{BCK}_{\mathtt{id}}$ only during the Return Codes Mapping table **CMtable** generation; no other part of the configuration or voting phases deals with the $\mathtt{BCK}_{\mathtt{id}}$ value. Thus, for winning, the Adversary can use only the data from the Fig. 32, which includes the following: the Setup Component outputs ($\mathbf{c}_{\mathtt{ck}}$, **CMtable**) and the honest control component's contribution during the setup phase ($\mathbf{Kc}_h, \mathbf{c}_{\mathtt{expCK},h}, \boldsymbol{\pi}_{\mathsf{expCK},h}$).

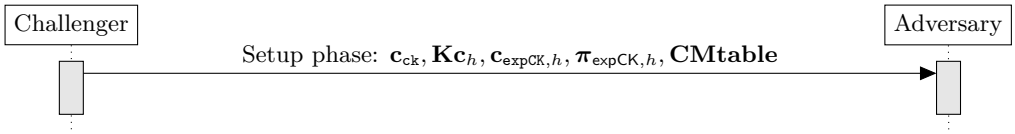

Figure 32: The part of the Challenger's output associated with retrieving the expected short Vote Cast Return Code in the vote injection game. The randomly selected or simulated values are marked with a $ sign.

Game rac-inj.1

Objective: In this game, the Challenger randomizes the honest control component's public keys, thereby preparing the switch to IdealSetupVoting.

Adjustments: Algorithms $\mathsf{CreateLCCShare}_{\mathsf{h}}$ and $\mathsf{CreateLVCCShare}_{\mathsf{h}}$ (executed in oracles $\mathcal{O}$HonestLCCShare and $\mathcal{O}$HonestCreateLVCCShare from Fig. 19 and 20) will use the canonical simulator $\mathcal{S}_{\mathsf{Exp}}$ to generate the proofs $\pi_{\mathsf{expLCC},h}$ and $\pi_{\mathsf{expLVCC},h}$ containing the Voter Choice Return Code Generation private key $\mathtt{k}_{h,\mathtt{id}}$ and the Voter Vote Cast Return Code Generation private key $\mathtt{kc}_{h,\mathtt{id}}$ of the honest control component.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,rac\text{-}inj}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,rac\text{-}inj.1}}| \leq 2 \cdot \mathtt{N}_{\mathtt{E}} \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{Exp,NIZK}}$$

where $\mathtt{N_E}$ is the number of voters and $\mathcal{B}_1$ is an adversary against the zero-knowledge property of the Exp proof system.

Reasoning: The idealized protocol IdealSetupVoting randomizes the honest control component public keys $\mathtt{K}_{h,\mathtt{id}}$ and $\mathtt{Kc}_{h,\mathtt{id}}$. However, in the voting phase, those keys are still required for generating the proofs $\pi_{\mathsf{expLCC},h}$ and $\pi_{\mathsf{expLVCC},h}$ in the oracles $\mathcal{O}$HonestLCCShare and $\mathcal{O}$HonestCreateLVCCShare (see Fig. 19 and 20). Therefore, before running IdealSetupVoting, we first need to simulate the remaining proofs.

The Adversary notices the change only by distinguishing between the real and simulated proofs (up to $\mathtt{N_E}$ for each of the two proofs) of correct exponentiation.

Effect on Future Games: From this game onwards, we no longer require correct public keys $\mathtt{K}_{h,\mathtt{id}}$ and $\mathtt{Kc}_{h,\mathtt{id}}$ in the voting phase, hence we can switch to the IdealSetupVoting.

Game rac-inj.2

Objective: In this game, the Challenger runs IdealSetupVoting instead of SetupVoting.

Adjustments: IdealSetupVoting substitutes all exponentiations done by the honest control component with randomly sampled elements from $\mathbb{Q}_p$. Therefore, IdealSetupVoting replaces $\left(\mathtt{hpCC}_{\mathtt{id},i}\right)^{\mathtt{k}_{h,\mathtt{id}}}$ and $\left(\mathtt{hCK}_{\mathtt{id}}\right)^{\mathtt{kc}_{h,\mathtt{id}}}$ with $\{x_{i,\mathtt{id}}\}_{i=1}^{n}$ and $y_{\mathtt{id}}$, where all $x_{i,\mathtt{id}}$ and $y_{\mathtt{id}}$ are randomly sampled from $\mathbb{Q}_p$.

To ensure that the Choice Return Codes and the Vote Cast Return Code of all voters are extractable, the Challenger needs to use the same random elements in the voting phase (i.e. adjust $\mathsf{CreateLCCShare}_{\mathsf{h}}$ and $\mathsf{CreateLVCCShare}_{\mathsf{h}}$ functions in $\mathcal{O}$HonestLCCShare and $\mathcal{O}$HonestCreateLVCCShare oracles from Fig. 19 and 20).

Concretely, the Challenger keeps an internal table where it stores inputs and outputs of the IdealSetupVoting protocol:

$$\left\{\left[\mathtt{hpCC}_{\mathtt{id},i} = \mathsf{HashAndSquare}(\mathrm{p}_i^{\mathtt{k}_{\mathtt{id}}}); x_{i,\mathtt{id}}\right]\right\}_{i=1}^{n}, \left[\mathtt{hCK}_{\mathtt{id}} = \mathsf{HashAndSquare}(\mathtt{CK}_{\mathtt{id}}); y_{\mathtt{id}}\right]$$

Then, in the voting phase, the Challenger uses this internal table in the functions $\mathsf{CreateLCCShare}_{\mathsf{h}}$ and $\mathsf{CreateLVCCShare}_{\mathsf{h}}$ to reply to the queries. For example in $\mathsf{CreateLCCShare}_{\mathsf{h}}$, whenever an honest CCR_h receives $\mathtt{pCC}_{\mathtt{id},x} = \mathrm{p}_x^{\mathtt{k}_{\mathtt{id}}}$, it computes $\mathsf{HashAndSquare}(\mathtt{pCC}_{\mathtt{id},x})$ and checks if the resulting value is present in the internal table. If this is the case, it returns the corresponding $x_{x,\mathtt{id}}$, otherwise it randomly samples a value $x_{x,\mathtt{id}}$ from $\mathbb{Q}_p$ and adds the new pair $\left[\mathsf{HashAndSquare}(\mathtt{pCC}_{\mathtt{id},x}); x_{x,\mathtt{id}}\right]$ to the internal table. Equivalent logic applies to $\mathsf{CreateLVCCShare}_{\mathsf{h}}$.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,rac\text{-}inj.1}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,rac\text{-}inj.2}}| = \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{SPVS,mtc}}$$

where $\mathcal{B}_2$ is an adversary against the Return Codes Mapping table **CMtable** correctness property of the Swiss Post Voting System.

Reasoning: The Adversary notices the change only by distinguishing between SetupVoting and IdealSetupVoting, which was proven to be difficult in Lemma 1. After this hop, the view of the Adversary changes to the one in Fig. 33.

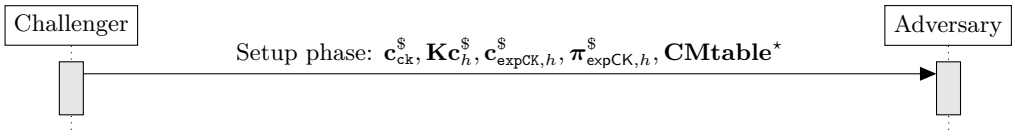

Figure 33: The part of the Challenger's output associated with retrieving the expected short Vote Cast Return Code after the switch to IdealSetupVoting. The randomly selected or simulated values are marked with a $ sign.

We observe that after the switch to IdealSetupVoting, the honest control component's share $\mathtt{lVCC}_{\mathtt{id},h}$ is sampled at random (see Fig. 21). Thus, in the Return Codes Mapping table **CMtable**, the symmetric key $\mathtt{skvcc}_{\mathtt{id}}$ that encrypts $\mathtt{VCC}_{\mathtt{id}}$ is independent from $\mathtt{BCK}_{\mathtt{id}}$. This means that, even with the knowledge of the Return Codes Mapping table **CMtable**, the Adversary has no information related to $\mathtt{skvcc}_{\mathtt{id}}$.

Wrapping up In the rac-inj.2 game the Adversary is not given any information about the Ballot Casting Key $\mathtt{BCK}_{\mathtt{id}}$. All values related to the expected $\mathtt{BCK}_{\mathtt{id}}$ are either selected at random (all values marked with the $ sign) or are an output of the hash oracle ($\mathtt{hlVCC}_{\mathtt{id},h}$). Therefore, the Adversary cannot extract any meaningful information and can only guess the value $\mathtt{BCK}_{\mathtt{id}}$ randomly. Moreover, since the honest control component will respond only to the limited number of queries, the Adversary has `maxConfAttempts` attempts. Thus,

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{rac\text{-}inj.2}} \leq \mathtt{maxConfAttempts} \cdot \left(\frac{1}{\mathcal{C}_{bck}}\right)$$

where `maxConfAttempts` is the maximum number of attempts the adversary is allowed to submit the confirmation key $\mathtt{CK}_{\mathtt{id}}$.

Collecting the bounds of the hops concludes the proof.

17 Universal Verifiability

Section 2.2.2 explains that *universal verifiability* allows auditors to ascertain that the Swiss Post Voting System correctly determined the result.

This section focuses on the correctness of the tally algorithms; we cover other aspects relevant for the correctness of the election result—such as the correct registration of votes, *ballot eligibility*, or *one vote per voter*—in our proof of individual verifiability in section 16.

17.1 Definition and Theorem

To formalize the correctness of the tally phase, we define the property *correct tally* and prove it under the assumption that all control components are dishonest.

We define *correct tally* with a game correctTally comparing the election result corresponding to the real execution of the tally phase with the election result obtained by the Challenger. Since we assume that all control components and the tally control component are dishonest, the Adversary is allowed to produce the entire election transcript. The Adversary wins if the claimed result passes the auditor's VerifyTally algorithm, but is different from the one calculated by the Challenger using the plaintext extractor (see section 13.2).

Figure 34 defines game correctTally capturing adversaries against *correct tally*. The advantage of an adversary $\mathcal{A}$ is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{correctTally}} = Pr[1 \leftarrow \mathsf{correctTally}_{\mathcal{A}}^{\mathsf{SPVS}}(\mathtt{context})]$$

$\underline{\mathsf{correctTally}_{\mathcal{A}}^{\mathsf{SPVS}}(\mathtt{context})}$:

1. $L = (\{\mathtt{vc_{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \pi_{\mathsf{Exp}}, \pi_{\mathsf{EqEnc}}\}_{\mathtt{id} \in \mathcal{ID}}) \leftarrow \mathcal{A}$ // The Adversary $\mathcal{A}$ prepares a list of ballots
2. $(\{\mathbf{c}_{\mathsf{mix},j}, \pi_{\mathsf{mix},j}, \mathbf{c}_{\mathsf{Dec},j}, \boldsymbol{\pi}_{\mathsf{dec},j}\}_{j=1}^{m}) \leftarrow \mathcal{A}$ // The Adversary $\mathcal{A}$ prepares the results of the online control components' mixing and decrypting algorithms
3. $(\mathbf{c}_{\mathsf{mix},m+1}, \pi_{\mathsf{mix},m+1}, \mathbf{m}, \boldsymbol{\pi}_{\mathsf{dec},m+1}, L_{\mathsf{votes}}) \leftarrow \mathcal{A}$ // The Adversary $\mathcal{A}$ prepares the results of the Tally control component's mixing, decrypting, and processing algorithms
4. If $\mathsf{VerifyTally}(L, \{\mathbf{c}_{\mathsf{mix},j}, \pi_{\mathsf{mix},j}, \mathbf{c}_{\mathsf{Dec},j}, \boldsymbol{\pi}_{\mathsf{dec},j}\}_{j=1}^{m}, \mathbf{c}_{\mathsf{mix},m+1}, \pi_{\mathsf{mix},m+1}, \mathbf{m}, \boldsymbol{\pi}_{\mathsf{dec},m+1}, L_{\mathsf{votes}})$ fails, abort and output 0
5. Else $\mathbf{c}_{\mathsf{init}} \leftarrow \mathsf{GetMixnetInitialCiphertexts}(\hat{\delta}, L, \mathbf{EL}_{\mathsf{pk}})$ // Get the mixnet initial ciphertexts from the list of ballots L
6. $\mathbf{m}^* \leftarrow \mathsf{Extract}(\mathbf{c}_{\mathsf{init}}, \mathbf{EL}_{\mathsf{pk}})$ // Extract the plaintexts from the mixnet initial ciphertexts
7. Set $L_{\mathsf{votes}}^* \leftarrow \mathsf{ProcessPlaintexts}(\mathbf{m}^*, \psi, \hat{\delta})$ // The context includes the primes mapping table pTable
8. If $L_{\mathsf{votes}}^* \neq L_{\mathsf{votes}}$ up to a permutation, output 1. Else, output 0

Figure 34: Game modeling attacks against correct tally. The public parameters `context` are implicit.

Theorem 4 *In the random oracle model*

$$Adv_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{correctTally}} \leq (m+1) \cdot Adv_{\mathcal{B}_1}^{\mathsf{mix},\mathsf{sSOUND}} + (m+1) \cdot \hat{\mathtt{N}}_{\mathtt{C}} \cdot Adv_{\mathcal{B}_2}^{\mathsf{dec},\mathsf{sSOUND}}$$

where m is the total number of control components, $\mathcal{B}_1$ is an adversary against the soundness of the Shuffle proof system, $\hat{\mathtt{N}}_\mathtt{C}$ is the number of mixed votes, $\mathcal{B}_2$ an adversary against the soundness of the decryption proof system.

17.2 Proof of Theorem 4 (Correct Tally)

In this section we show that the soundness of the shuffle and decryption NIZK proofs allows us to claim that the Swiss Post Voting System enjoys tally correctness. To this end, the Challenger manually checks the statements of the shuffle and decryption proofs. The Adversary wins if the algorithm VerifyTally returns $\top$ yet the election result L_{votes} is incorrect. For the purpose of this proof, we consider the Tally control component as the m+1-th control component and designate the electoral board public key $\mathtt{EB}_{\mathtt{pk}}$ as $\mathtt{EL}_{\mathtt{pk},m+1}$

Game correctTally.1

Objective: In this game, the Challenger aborts if the result of the shuffle $\mathbf{c}_{\mathsf{mix},j}$ is not a re-encryption and permutation of the control component's input $\mathbf{c}_{\mathsf{Dec},j-1}$ for $1 \leq j \leq m+1$.

Adjustments: After the successful execution of the VerifyTally algorithm, the Challenger manually checks that the lists $\mathbf{c}_{\mathsf{Dec},j-1}$ and $\mathbf{c}_{\mathsf{mix},j}$ contain identical plaintexts up to a permutation. First, the Challenger combines the remaining election public keys $\overline{\mathtt{EL}}_{\mathtt{pk},j} \leftarrow \mathsf{CombinePublicKeys}(\mathtt{EL}_{\mathtt{pk},j}, \ldots, \mathtt{EL}_{\mathtt{pk},m+1})$ for each $1 \leq j \leq m+1$ (note that $\overline{\mathtt{EL}}_{\mathtt{pk},1} = \mathtt{EL}_{\mathtt{pk}}$). Then, the Challenger checks that:

$$\mathsf{Extract}(\mathbf{c}_{\mathsf{Dec},j-1}, \overline{\mathtt{EL}}_{\mathtt{pk},j}) = \mathsf{Extract}(\mathbf{c}'_{\mathsf{mix},j}, \overline{\mathtt{EL}}_{\mathtt{pk},j})$$

where $\mathbf{c}'_{\mathsf{mix},j}$ is a permutation π_j of $\mathbf{c}_{\mathsf{mix},j}$, for each $1 \leq j < m+1$. If for some j the permutation π_j does not exist, the Challenger aborts the game.

Claim:

$$|\mathsf{Adv}^{\mathsf{SPVS},\mathsf{correctTally}.1}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{SPVS},\mathsf{correctTally}}_{\mathcal{A}}| \leq (m+1) \cdot \mathsf{Adv}^{\mathsf{mix},\mathsf{sSOUND}}_{\mathcal{B}_1}$$

where $m+1$ is the total number of control components and $\mathcal{B}_1$ is an adversary against the soundness of the shuffle proof (see section 8).

Reasoning: The Adversary can notice the extra check only by generating a shuffle proof for an invalid statement without the auditors noticing; in that case, the algorithm VerifyTally returns $\top$ when verifying a bogus shuffle proof. However, the soundness of the shuffle argument guarantees that this would happen only with negligible probability (see section 8).

Game correctTally.2

Objective: In this game, the Challenger aborts if the result of the (partial) decryption $\mathbf{c}_{\mathsf{Dec},j}$ does not contain the same plaintexts as the shuffled ciphertexts $\mathbf{c}_{\mathsf{mix},j}$ for $1 \leq j \leq m+1$.

Adjustments: After the successful execution of the VerifyTally algorithm, the Challenger manually checks that the lists $\mathbf{c}_{\mathsf{mix},j}$ and $\mathbf{c}_{\mathsf{Dec},j}$ (for $1 \leq j \leq m$) and $\mathbf{c}_{\mathsf{mix},m+1}$ and $\mathbf{m}$ contain identical plaintexts. First, the Challenger reconstructs the remaining election public keys:

$$\overline{\mathtt{EL}}_{\mathtt{pk},j} \leftarrow \mathsf{CombinePublicKeys}(\mathtt{EL}_{\mathtt{pk},j}, \ldots, \mathtt{EL}_{\mathtt{pk},m+1})$$

for each $1 \leq j \leq m+1$. Then, the Challenger checks that $\mathsf{Extract}(\mathbf{c}_{i,j}, \overline{\mathtt{EL}}_{\mathtt{pk},j}) = \mathsf{Extract}(\mathbf{c}'_{i,j}, \overline{\mathtt{EL}}_{\mathtt{pk},j+1})$, where $\mathbf{c}_{i,j} \in \mathbf{c}_{\mathsf{mix},j}$ and $\mathbf{c}'_{i,j} \in \mathbf{c}_{\mathsf{Dec},j}$. If for some pair (i,j) the equality does not hold, the Challenger aborts the game.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{correctTally}.2} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{correctTally}.1}| \leq (m+1) \cdot \hat{\mathtt{N}}_{\mathtt{C}} \cdot \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{dec},\mathsf{sSOUND}}$$

where $m+1$ is the total number of control components, $\hat{\mathtt{N}}_{\mathtt{C}}$ is the number of mixed votes, and $\mathcal{B}_2$ is an adversary against the soundness of the decryption proof system.

Reasoning: The Adversary can notice the extra check only by generating a decryption proof for an invalid statement without the auditors noticing; in that case, the algorithm VerifyTally returns $\top$ when verifying a bogus decryption proof. However, the soundness of the decryption proof guarantees that this would happen only with negligible probability (see Def. 3).

Wrapping up: After correctTally.2 the adversary can no longer modify the content of the votes during the shuffle or (partial) decryption. Therefore, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{correctTally}.2} = 0$.
Collecting the bounds of the hops concludes the proof.

18 Vote Privacy

18.1 Oracles for Modeling Vote Privacy

Our computational proof of vote privacy grants the Adversary access to multiple oracles. These oracles represent the honest parties' actions during the SendVote, ConfirmVote, and Tally protocol[7] and allow the Adversary to deviate from the standard election control flow (as explained in section 1.2).
We re-use the following oracles from section 13.3:

- $\mathcal{O}$HonestPartialDecPCC
- $\mathcal{O}$HonestLCCShare
- $\mathcal{O}$VoterChecksCRC
- $\mathcal{O}$HonestCreateLVCCShare
- $\mathcal{O}$HonestVerifyLVCCShares
- $\mathcal{O}$VoterChecksVCC

However, we assume a trustworthy voting client for vote privacy (see section 2.2.3); hence, we replace the following oracles:

- $\mathcal{O}$HonestVote replaces $\mathcal{O}$VoterGivesSVK.
- $\mathcal{O}$ConfirmBallot replaces $\mathcal{O}$VoterGivesBCK.

Moreover, we require a computational proof of privacy under the assumption that all m online control components are malicious (see section 2.2.3). For privacy, we thus consider the *tally control component*—together with a trustworthy electoral board member—an additional control component and we assume that just one out of $m + 1$ control components is trustworthy. Therefore, we introduce the oracle $\mathcal{O}$HonestMixDecOnlinePrivacy to represent the honest online control component during the tally phase and $\mathcal{O}$HonestMixDecOfflinePrivacy for the honest Tally control component.
Figure 35 and 36 detail the vote privacy-specific oracles.

[7] Section 13.3 explains why we omit oracles for the configuration phase.

$\mathcal{O}$HonestVote($\mathtt{id}, \hat{\mathtt{p}}_{\mathtt{id},0}, \hat{\mathtt{p}}_{\mathtt{id},1}$): // send an encrypted vote on behalf of an honest voter.

1. If the voter is not honest, i.e., $\mathtt{id} \notin \mathcal{ID}_h$ or already voted i.e. $(\mathtt{id}, \mathsf{voted}) \in \mathsf{L}_{\mathtt{VoterStatus}}$, abort and output $\perp$.
2. $\mathtt{k}_{\mathtt{id}} \leftarrow \mathsf{GetKey}(\mathtt{SVK}_{\mathtt{id}}, \mathtt{VCks}_{\mathtt{id}}, \mathtt{vc}_{\mathtt{id}})$ // obtain the verification card secret key.
   GetKey implicitly checks that the context containing $\mathtt{vc}_{\mathtt{id}}$, $\mathtt{EL}_{\mathtt{pk}}$, $\mathbf{pk}_{\mathrm{CCR}}$ and $\hat{\mathtt{p}}_{\mathtt{id},0}, \hat{\mathtt{p}}_{\mathtt{id},1}$ is correct.
3. $\mathtt{b}_\beta \leftarrow \mathsf{CreateVote}\left(\mathtt{vc}_{\mathtt{id}}, \hat{\mathtt{p}}_{\mathtt{id},\beta}, \mathbf{w}_{\mathtt{id}}, \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, \mathtt{k}_{\mathtt{id}}\right)$
   the ballot for the voting option $\hat{\mathtt{p}}_{\mathtt{id},\beta}$ with optional write-ins $\mathbf{w}_{\mathtt{id}}$ selected at random.
4. Set $\mathcal{S}_{\mathsf{checked}} = \{\hat{\mathtt{p}}_{\mathtt{id},\beta}\}$ // so the voter will verify the corresponding Choice Return Code
5. Add the entry $(\mathtt{id}, \mathsf{voted})$ to $\mathsf{L}_{\mathtt{VoterStatus}}$ and $(\mathtt{vc}_{\mathtt{id}}, \hat{\mathtt{p}}_{\mathtt{id},0}, \hat{\mathtt{p}}_{\mathtt{id},1})$ to $\mathsf{L}_{\mathtt{honest}}$.
6. Output $\mathtt{b}_\beta$.

$\mathcal{O}$ConfirmBallot($\mathtt{id}$): // confirm the honest voter's ballot.

1. If $(\mathtt{id}, \mathsf{goodCRC}) \notin \mathsf{L}_{\mathtt{VoterStatus}}$, abort and output $\perp$.
2. Else, set the voter's entry in $\mathsf{L}_{\mathtt{VoterStatus}}$ to $(\mathtt{id}, \mathsf{confirmed})$.
3. $\mathtt{CK}_{\mathtt{id}} \leftarrow \mathsf{CreateConfirmMessage}(\mathtt{BCK}_{\mathtt{id}}, \mathtt{k}_{\mathtt{id}})$
4. Output the Confirmation Key $\mathtt{CK}_{\mathtt{id}}$.

Figure 35: Oracles given to the adversary $\mathcal{A}$ in experiments $\mathsf{Exp}^{\mathsf{priv},\beta}_{\mathcal{A},\mathcal{V}}(\mathtt{context})$ (see Figure 37) modeling the voting phase. The Challenger parametrizes the oracle $\mathcal{O}$HonestVote with the coin $\beta \in \{0, 1\}$.

$\mathcal{O}\mathsf{HonestMixDecOnlinePrivacy}\left(\mathsf{bb}, \{\mathbf{c}_{\mathsf{mix},j}, \mathbf{c}_{\mathsf{Dec},j}, \pi_{\mathsf{mix},j}, \boldsymbol{\pi}_{\mathsf{dec},j}\}_{j=1}^{h-1}\right)$:

1. For each $\mathtt{vc}_{\mathtt{id}} \in \mathsf{L}_{\mathtt{confirmedVotes},h}$, check that $\mathtt{vc}_{\mathtt{id}} \in \mathsf{L}_{\mathtt{honest}}$ and add the corresponding encoded voting options to the lists of honest votes $L_0 \leftarrow L_0 \cup (\hat{\mathrm{p}}_{\mathtt{id},0})$ and $L_1 \leftarrow L_1 \cup (\hat{\mathrm{p}}_{\mathtt{id},1})$.
2. Ensure that the lists L_0 and L_1 are identical up to a permutation:
   $\mathsf{Order}(L_0) = \mathsf{Order}(L_1)$, otherwise abort and output $\perp$.
   This condition ensures that the tally result does not provide any information about the coin β.
3. $\mathbf{c}_{\mathsf{init},h} \leftarrow \mathsf{GetMixnetInitialCiphertexts}_h(\hat{\delta}, \mathbf{vcMap}_h, \mathtt{EL}_{\mathtt{pk}})$ // The context includes the ballot box ID bb.
4. If $\mathsf{VerifyMixDecOnline}_h(\mathbf{c}_{\mathsf{init},h}, \{\mathbf{c}_{\mathsf{mix},j}, \pi_{\mathsf{mix},j}, \mathbf{c}_{\mathsf{Dec},j}, \boldsymbol{\pi}_{\mathsf{dec},j}\}_{j=1}^{h-1}, \mathtt{EL}_{\mathtt{pk}}, \{\mathtt{EL}_{\mathtt{pk},j}\}_{j=1}^{m}, \mathtt{EB}_{\mathtt{pk}})$ fails, abort and output $\perp$. If the honest control component is the first to mix $h = 1$, we omit this step.
5. $(\mathbf{c}_{\mathsf{mix},h}, \pi_{\mathsf{mix},h}, \mathbf{c}_{\mathsf{Dec},h}, \boldsymbol{\pi}_{\mathsf{dec},h}) \leftarrow \mathsf{MixDecOnline}_h(\mathbf{c}_{\mathsf{Dec},h-1}, \{\mathtt{EL}_{\mathtt{pk},j}\}_{j=1}^{m}, \mathtt{EB}_{\mathtt{pk}}, \mathtt{EL}_{\mathtt{sk},h})$
6. Output $(\mathbf{c}_{\mathsf{mix},h}, \pi_{\mathsf{mix},h}, \mathbf{c}_{\mathsf{Dec},h}, \boldsymbol{\pi}_{\mathsf{dec},h})$

$\mathcal{O}\mathsf{HonestMixDecOfflinePrivacy}\left(\mathsf{bb}, \{\mathbf{vc}_j, \mathbf{E1}_j, \widetilde{\mathbf{E1}}_j, \mathbf{E2}_j, \boldsymbol{\pi}_{\mathsf{Exp},j}, \boldsymbol{\pi}_{\mathsf{EqEnc},j}\}_{j=1}^{m}, \{\mathbf{c}_{\mathsf{mix},j}, \mathbf{c}_{\mathsf{Dec},j}, \pi_{\mathsf{mix},j}, \boldsymbol{\pi}_{\mathsf{dec},j}\}_{j=1}^{m}\right)$:

1. If the encrypted confirmed votes $\{\mathbf{vc}_j, \mathbf{E1}_j, \widetilde{\mathbf{E1}}_j, \mathbf{E2}_j, \boldsymbol{\pi}_{\mathsf{Exp},j}, \boldsymbol{\pi}_{\mathsf{EqEnc},j}\}_{j=1}^{m}$ are not consistent between all control components, abort and output $\perp$.
2. For each $\mathtt{vc}_{\mathtt{id}} \in \mathbf{vc}_1$, check that $\mathtt{vc}_{\mathtt{id}} \in \mathsf{L}_{\mathtt{honest}}$ and add the corresponding encoded voting options to the lists of honest votes $L_0 \leftarrow L_0 \cup (\hat{\mathrm{p}}_{\mathtt{id},0})$ and $L_1 \leftarrow L_1 \cup (\hat{\mathrm{p}}_{\mathtt{id},1})$.
3. Ensure that the lists L_0 and L_1 are identical up to a permutation:
   $\mathsf{Order}(L_0) = \mathsf{Order}(L_1)$, otherwise abort and output $\perp$.
   This condition ensures that the tally result does not provide any information about the coin β.
4. If $\mathsf{VerifyVotingClientProofs}(\mathbf{vc}_1, \mathbf{E1}_1, \widetilde{\mathbf{E1}}_1, \mathbf{E2}_1, \boldsymbol{\pi}_{\mathsf{Exp},1}, \boldsymbol{\pi}_{\mathsf{EqEnc},1}, \mathbf{KMap}, \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}})$ fails, abort and output $\perp$.
5. $\mathbf{c}_{\mathsf{init},1} \leftarrow \mathsf{GetMixnetInitialCiphertexts}_1(\hat{\delta}, \mathbf{vcMap}_1, \mathtt{EL}_{\mathtt{pk}})$
6. If $\mathsf{VerifyMixDecOffline}(\mathbf{c}_{\mathsf{init},1}, \{\mathbf{c}_{\mathsf{mix},j}, \pi_{\mathsf{mix},j}, \mathbf{c}_{\mathsf{Dec},j}, \boldsymbol{\pi}_{\mathsf{dec},j}\}_{j=1}^{m}, \mathtt{EL}_{\mathtt{pk}}, \{\mathtt{EL}_{\mathtt{pk},j}\}_{j=1}^{m}, \mathtt{EB}_{\mathtt{pk}})$ fails, abort and output $\perp$.
7. $(\mathbf{c}_{\mathsf{mix},m+1}, \pi_{\mathsf{mix},m+1}, \mathbf{m}, \boldsymbol{\pi}_{\mathsf{dec},m+1}) \leftarrow \mathsf{MixDecOffline}(\mathbf{c}_{\mathsf{Dec},m}, \mathtt{PW}_1, \ldots, \mathtt{PW}_{k-1})$
8. $L_{\mathsf{votes}} \leftarrow \mathsf{ProcessPlaintexts}(\mathbf{m}^*, \psi, \hat{\delta})$ // The context includes the primes mapping table pTable.
9. Output $(L_{\mathsf{votes}}, \mathbf{c}_{\mathsf{mix},m+1}, \pi_{\mathsf{mix},m+1}, \mathbf{m}, \boldsymbol{\pi}_{\mathsf{dec},m+1})$.

Figure 36: Oracle given to the adversary $\mathcal{A}$ in the experiment $\mathsf{Exp}^{\mathsf{priv},\beta}_{\mathcal{A},\mathcal{V}}(\mathtt{context})$ (see Figure 37) during the tally phase. The adversary can query once either one of the oracles.

18.2 Definition and Theorem

In the literature, multiple formal definitions of vote privacy exist [8]. We choose Benaloh's definition [7] since it is simple and sufficient for the Swiss use case (Swiss elections do *not* have a complex counting function such as ranked-choice voting).

Benaloh's definition formalizes vote privacy with an experiment where the challenger picks a coin $\beta \in \{0, 1\}$, and the adversary provides the challenger with two possible voting options $\hat{\mathrm{p}}_{\mathtt{id},0}, \hat{\mathrm{p}}_{\mathtt{id},1}$. Depending on the coin β, the challenger picks the first or the second option that was provided. At some point, the adversary asks to see the tally result and provides a guess b for the coin β. The adversary wins the experiment, if the guess b is correct.

This formalization excludes trivial privacy breaches where all voters vote for the same option or the ballot box contains just one vote since, by definition, the tally result must not provide any information about individual votes. One can justify this condition because a ballot box usually contains a sufficient number of votes from honest voters. Nevertheless, the point is more delicate under the trust assumptions in section 2.2.3, where all online control components are untrustworthy. In that scenario, an adversary could breach vote privacy by deleting a sufficient number of honest voters' votes. However, the Federal Chancellery's Ordinance allows us to exclude these types of attacks. For convenience, we repeat the relevant explanations:

> [VEleS Annex 2.7.2]: There is no obligation to prevent attacks that limit the number of tallied votes to the degree that all partial votes for a question, list or candidate are the same.

> [Explanatory Report, Sec 4.2.1]: In cases where Number 2.9.3.3 applies, as a result of the exclusion in Number 2.7.2, [...], it would be permissible to assume that a control component correctly registers and subsequently does not delete a sufficient number of votes from trustworthy voters as sent by the trustworthy user platform. Alternatively, it would be permissible to assume that the secrecy of the vote is not endangered if not all votes cast are tallied, but simply an arbitrary subset.

Hence, we conclude that Benaloh's definition is an appropriate formalization for capturing the vote privacy requirements defined in section 2.2.3.

Figure 37 defines two Benaloh-type experiments $\mathsf{Exp}^{\mathsf{priv},0}_{\mathcal{A},\mathcal{V}}(\texttt{context})$ and $\mathsf{Exp}^{\mathsf{priv},1}_{\mathcal{A},\mathcal{V}}(\texttt{context})$; vote privacy is achieved whenever the experiments are indistinguishable, i.e., $\mathsf{Adv}^{\mathsf{SPVS},\mathsf{priv}}_{\mathcal{A}} \approx 0$.

$\underline{\mathsf{Exp}^{\mathsf{priv},\beta}_{\mathcal{A},\mathcal{V}}(\texttt{context})}$:

Configuration Phase

1. $\mathcal{ID}_h, h \leftarrow \mathcal{A}$ // The Adversary $\mathcal{A}$ specifies the honest voters and the honest control component
2. $(\mathbf{CMtable}, \mathsf{L}_{\texttt{lVCC}}, \mathsf{L}_{\texttt{pCC}}, \mathbf{VCard}, \mathbf{VCks}, (\mathbf{sk}_{\mathrm{CCR}_h}, \mathbf{kc}_h, \mathbf{k}_h), \mathbf{pk}_{\mathrm{CCR}}, (\mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{K}_j, \mathbf{Kc}_j)_{j=1}^{m}, \mathbf{K}) \leftarrow \mathsf{SetupVoting}$
3. $\left(\mathtt{EL}_{\mathtt{pk}}, \mathtt{EL}_{\mathtt{sk},h}, (\mathtt{EL}_{\mathtt{pk},j})_{j=1}^{m}, \mathtt{EB}_{\mathtt{sk}}, \mathtt{EB}_{\mathtt{pk}}\right) \leftarrow \mathsf{SetupTally}$
4. If $\mathsf{VerifyConfigPhase}$ fails, abort and output 0. // Verify the configuration phase—see section 11.4.
5. Initialize an empty voter status list $\mathsf{L}_{\texttt{VoterStatus}}$ and an empty honest ballots list $\mathsf{L}_{\texttt{honest}}$.
6. Set $\mathbf{VCard}^* \leftarrow (\mathbf{VCard})_{\mathtt{id} \notin \mathcal{ID}_h}$ // Do not give the voting card of honest voters to the Adversary $\mathcal{A}$.
7. Give the following data to $\mathcal{A}$:

$$\left(\mathbf{VCard}^*, \mathbf{VCks}, \mathbf{CMtable}, \mathsf{L}_{\texttt{lVCC}}, \mathsf{L}_{\texttt{pCC}}, \mathtt{EL}_{\mathtt{pk}}, \mathbf{pk}_{\mathrm{CCR}}, (\mathtt{EL}_{\mathtt{pk},j}, \mathbf{pk}_{\mathrm{CCR}_j}, \mathbf{K}_j, \mathbf{Kc}_j)_{j=1}^{m}, \mathbf{K}, \mathtt{EB}_{\mathtt{pk}}\right)$$

8. If $h \neq m+1$, give the following keys to $\mathcal{A}$: // privacy does not rely on the secrecy of those keys

$$\left(\mathbf{sk}_{\mathrm{CCR}_h}, \mathbf{kc}_h, \mathbf{k}_h\right)$$

Voting Phase

The Adversary is allowed to query the oracles $\mathcal{O}\mathsf{HonestPartialDecPCC}, \mathcal{O}\mathsf{HonestLCCShare}, \mathcal{O}\mathsf{VoterChecksCRC}$ from figure 19, $\mathcal{O}\mathsf{HonestCreateLVCCShare}, \mathcal{O}\mathsf{HonestVerifyLVCCShares}, \mathcal{O}\mathsf{VoterChecksVCC}$ from figure 20, and $\mathcal{O}\mathsf{HonestVote}, \mathcal{O}\mathsf{ConfirmBallot}$ from figure 35. The Challenger parametrizes the oracle $\mathcal{O}\mathsf{HonestVote}$ with the coin β. The oracles and invoked algorithms impose certain conditions and restrictions; for instance, the adversary cannot send multiple votes for the same voter or confirm a vote that was not previously sent.

Tally Phase

The Adversary $\mathcal{A}$ is allowed to query one of the following oracles exactly once for each ballot box ID bb: $\mathcal{O}\mathsf{HonestMixDecOnlinePrivacy}$ (if $h \neq m+1$) or $\mathcal{O}\mathsf{HonestMixDecOfflinePrivacy}$ (if $h = m+1$). The oracles ensure that the tally result does not leak the coin β. Figure 36 defines both oracles.

9. $b \leftarrow \mathcal{A}$ // The Adversary $\mathcal{A}$ outputs a decision
10. Return b

Figure 37: Benaloh's experiment for vote privacy $\mathsf{Exp}^{\mathsf{priv},\beta}_{\mathcal{A},\mathcal{V}}(\texttt{context})$ defined for $\beta \in \{0,1\}$. The protocols $\mathsf{SetupVoting}$ and $\mathsf{SetupTally}$ are executed in interaction with the adversary $\mathcal{A}$. The public parameters $\texttt{context}$ are implicit.

Definition 15 (Privacy) *Consider the* Swiss Post Voting System *described in section 10 for a set $\mathcal{ID}$ of voter identities. We say that the protocol has vote privacy if:*

$$Adv^{\mathsf{SPVS},\mathsf{priv}}_{\mathcal{A}} = \left| Pr\left[1 \leftarrow [\mathsf{Exp}^{\mathsf{priv},0}_{\mathcal{A},\mathcal{V}}(\texttt{context})\right] - Pr\left[1 \leftarrow [\mathsf{Exp}^{\mathsf{priv},1}_{\mathcal{A},\mathcal{V}}(\texttt{context})\right] \right| \approx 0$$

Theorem 5 *In the random oracle model*

$$\begin{aligned}Adv_{\mathcal{A}}^{\mathsf{SPVS,priv}} \leq\ & \mathtt{N_{bb}} \cdot m \cdot (Adv_{\mathcal{B}_1}^{\mathsf{mix,sound}} + \hat{\mathtt{N}}_{\mathtt{C}} \cdot Adv_{\mathcal{B}_2}^{\mathsf{dec,sSOUND}}) + (\mathtt{N_E} - \mathtt{N}_{\mathtt{E}h}) \cdot (Adv_{\mathcal{B}_3}^{\mathsf{Exp,sSOUND}} + Adv_{\mathcal{B}_4}^{\mathsf{EqEnc,sSOUND}}) + \\ & 2 \cdot \mathtt{N_E} \cdot m \cdot \epsilon_{ext} + 2 \cdot \mathtt{N}_{\mathtt{E}h} \cdot Adv_{\mathcal{B}_5}^{\mathsf{ElGamal,INDCPA}} + Adv_{\mathcal{B}_6}^{\mathsf{sch,NIZK}} + \mathtt{N_{bb}} \cdot (\hat{\mathtt{N}}_{\mathtt{C}} \cdot Adv_{\mathcal{B}_7}^{\mathsf{dec,NIZK}} + Adv_{\mathcal{B}_8}^{\mathsf{mix,NIZK}}) + \\ & \mathtt{N}_{\mathtt{E}h} \cdot (\frac{1}{2^{\lambda_{\mathtt{KSkey}}}} + Adv_{\mathcal{B}_{11}}^{\mathsf{SEnc,INDCPA}} + Adv_{\mathcal{B}_{12}}^{\mathbb{Q}_p,\mathsf{ESGSP}} + Adv_{\mathcal{B}_{13}}^{\mathsf{ElGamal,INDCPA}})\end{aligned}$$

where $\mathtt{N_{bb}}$ *is a number of ballot boxes used in the election,* $\hat{\mathtt{N}}_{\mathtt{C}}$ *is the number of mixed votes,* m *is the number of online control components,* $\mathtt{N_E}$ *is the number of voters,* $\mathtt{N}_{\mathtt{E}h}$ *is the number of honest voters,* $\mathcal{B}_1$ *is an adversary against the soundness of the shuffle argument explained in Section 8,* $\mathcal{B}_2$ *is an adversary against the soundness of the decryption proof system,* $\mathcal{B}_3$ *is an adversary against the simulation soundness property of the exponentiation proof system* Exp*,* $\mathcal{B}_4$ *is an adversary against the simulation soundness property of the plaintext equality proof system* EqEnc*,* ϵ_{ext} *is given by the weak simulation extractability property of the Exponentiation proof system* Exp*,* $\mathcal{B}_5$ *is an adversary against the IND-CPA property of the ElGamal encryption scheme,* $\mathcal{B}_6$ *is an adversary against the zero-knowledge property of the Schnorr proof system,* $\mathcal{B}_7$ *is an adversary against the zero-knowledge property of the Decryption proof system* dec*,* $\mathcal{B}_8$ *is an adversary against the zero-knowledge property of the shuffle argument* mix*,* $\mathcal{B}_9$ *is an adversary against the zero-knowledge property of the exponentiation proof system* Exp*,* $\mathcal{B}_{10}$ *is an adversary against the zero-knowledge properties of the plaintext equality* EqEnc *proof system,* $\lambda_{\mathtt{KSkey}}$ *is the bit strength of the keystore symmetric encryption key* $\mathtt{KSkey}_{\mathtt{id}}$*,* $\mathcal{B}_{11}$ *is an adversary against the IND-CPA property of the symmetric encryption scheme,* $\mathcal{B}_{12}$ *is an adversary against the* ESGSP *problem (see Definition 9.2),* $\mathcal{B}_{13}$ *is an adversary against the IND-CPA property of the distributed ElGamal encryption scheme [14].*

18.3 Proof of Theorem 5 (Vote Privacy)

In this section, we prove that the experiments $\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{priv},0}(\mathtt{context})$ and $\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{priv},1}(\mathtt{context})$ are indistinguishable. Therefore, the Swiss Post Voting System enjoys vote privacy as defined in section 2.2.3.

Proof Outline

We start with the original Swiss Post Voting System, then, with gradual changes, arrive at a point where one can switch between the experiments $\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{priv},0}(\mathtt{context})$ and $\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{priv},1}(\mathtt{context})$ without the Adversary noticing.

Step 8 in experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{priv},\beta}(\mathtt{context})$ (see Figure 37) strengthens the Adversary by exposing some of the trusted control component's secret keys, demonstrating that vote privacy does not rely on the secrecy of those keys. We reveal the following keys to the Adversary:

1. The CCR_h Choice Return Codes encryption secret key $\mathbf{sk}_{\mathrm{CCR}_h}$,
2. The vector of Voter Vote Cast Return Code Generation secret keys $\mathbf{kc}_h$,

3. The vector of Voter Choice Return Code Generation secret key $\mathbf{k}_h$.

During the Swiss Post Voting System's execution, the Adversary queries the Challenger and receives replies. Figure 38 shows the Challenger's output that is not related to the exposed keys.

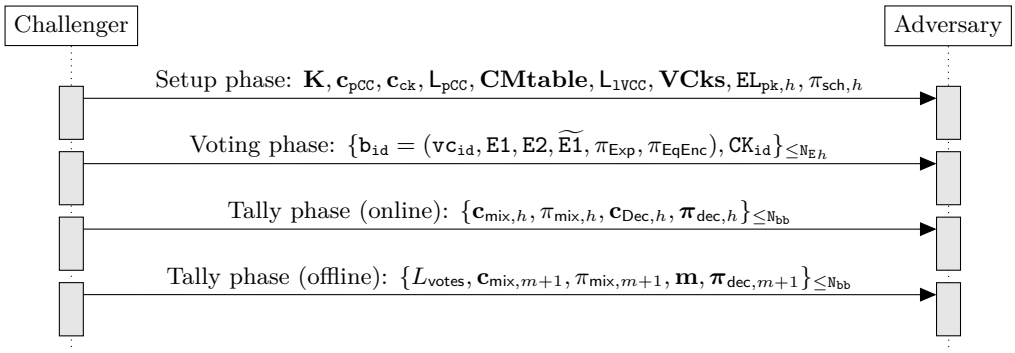

Figure 38: The Challenger's output associated with the honest parties' secret keys in the vote privacy game $\mathsf{Exp}^{\mathsf{priv},\beta}_{\mathcal{A},\mathcal{V}}(\mathtt{context})$.

In the first part of the proof, hops priv.1 - priv.2, we show that the Adversary cannot alter the tally result using the corrupt control components. Then, in hop priv.3, we demonstrate that the Adversary cannot mark or copy the honest voter's ballot. After that, in priv.4, we show that the Adversary cannot alter the Return Codes Mapping table **CMtable** and make some Choice Return Codes unextractable.

In the second part, we proceed with the randomization of the Challenger's outputs, except for the vote encryption $\mathtt{E1}$ and the election public key share $\mathtt{EL}_{\mathtt{pk},h}$ (if one online control component is honest) or $\mathtt{EB}_{\mathtt{sk}}$ (if the offline tally control component is honest). Finally, we rely on the IND-CPA property of the ElGamal encryption scheme to prove that we can safely switch between $\mathsf{Exp}^{\mathsf{priv},0}_{\mathcal{A},\mathcal{V}}(\mathtt{context})$ and $\mathsf{Exp}^{\mathsf{priv},1}_{\mathcal{A},\mathcal{V}}(\mathtt{context})$.

After this point, the Adversary can no longer tell the experiments apart; thus, his only chance to win is to output a guess b for the coin β. The probability of correctly guessing is $\frac{1}{2}$ and is independent of the coin β. Therefore, $\mathsf{Adv}^{\mathsf{SPVS},\mathsf{priv}}_{\mathcal{A}} \approx 0$, which is what we wanted to prove.

Game priv.1

Objective: In this game, the Challenger aborts if $\mathbf{c}_{\mathtt{mix},h-1}$ is not the result of a series of correct shuffles of the initial mixnet ciphertexts $\mathbf{c}_{\mathtt{init},h}$ performed by the control components before the honest control component h. For the purpose of this game hop, we consider the Tally control component as the m+1-th control component and designate the electoral board public key $\mathtt{EB}_{\mathtt{pk}}$ as $\mathtt{EL}_{\mathtt{pk},m+1}$. Section 13.2 describes the plaintext extractor used in this game hop.

Adjustments: After the successful execution of the $\mathsf{VerifyMixDecOnline}_h$ or VerifyMixDecOffline algorithm, the Challenger manually checks that the lists $\mathbf{c}_{\mathsf{Dec},j-1}$ and $\mathbf{c}_{\mathsf{mix},j}$ contain identical plaintexts up to a permutation for each $j < h$, and $\mathbf{c}_{\mathsf{Dec},0} = \mathbf{c}_{\mathsf{init},h}$. First, the Challenger combines the remaining election public keys $\overline{\mathtt{EL}}_{\mathtt{pk},j} \leftarrow \mathsf{CombinePublicKeys}(\mathtt{EL}_{\mathtt{pk},j}, \ldots, \mathtt{EL}_{\mathtt{pk},m+1})$ for each $1 \leq j \leq m+1$ (note that $\overline{\mathtt{EL}}_{\mathtt{pk},1} = \mathtt{EL}_{\mathtt{pk}}$). Then, the Challenger checks that:

$$\mathsf{Extract}(\mathbf{c}_{\mathsf{Dec},j-1}, \overline{\mathtt{EL}}_{\mathtt{pk},j}) = \mathsf{Extract}(\mathbf{c}'_{\mathsf{mix},j}, \overline{\mathtt{EL}}_{\mathtt{pk},j})$$

where $\mathbf{c}'_{\mathsf{mix},j}$ is a permutation π_j of $\mathbf{c}_{\mathsf{mix},j}$, for each $1 \leq j < h$. If for some j the permutation π_j does not exist, the Challenger aborts the game.

Claim:

$$|\mathsf{Adv}^{\mathsf{SPVS,priv.1}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{SPVS,priv}}_{\mathcal{A}}| \leq \mathtt{N}_{\mathtt{bb}} \cdot m \cdot \mathsf{Adv}^{\mathsf{mix,sound}}_{\mathcal{B}_1}$$

where $\mathtt{N}_{\mathtt{bb}}$ is a number of ballot boxes used in the election event, m is the total number of control components and $\mathcal{B}_1$ is an adversary against the soundness of the shuffle argument (see section 8).

Reasoning: Before mixing and decrypting the votes of a specific ballot box (up to $\mathtt{N}_{\mathtt{bb}}$ per election), the honest control component verifies the shuffle proofs of the preceding control components in the algorithms $\mathsf{VerifyMixDecOnline}_h$ or VerifyMixDecOffline respectively. Thus, the honest control component has to verify $\mathtt{N}_{\mathtt{bb}} \cdot (h-1)$ shuffle proofs. At most, if the Tally control component is the only trustworthy control component, the challenger must manually check $\mathtt{N}_{\mathtt{bb}} \cdot m$ shuffle proofs.

The Adversary can notice the extra check only by generating a shuffle proof for an invalid statement without the honest control component noticing; in that case, the algorithms $\mathsf{VerifyMixDecOnline}_h$ or VerifyMixDecOffline return $\top$ when verifying a bogus shuffle proof. However, the soundness of the shuffle argument guarantees that this would happen only with negligible probability (see section 8).

Game priv.2

Objective: In this game, the Challenger aborts if the input to the honest control component $\mathbf{c}_{\mathsf{Dec},h-1}$ is not the result of a series of correct partial decryptions performed by the control components preceding the honest control component h. For the purpose of this game hop, we consider the Tally control component as the m+1-th control component and designate the electoral board public key $\mathtt{EB}_{\mathtt{pk}}$ as $\mathtt{EL}_{\mathtt{pk},m+1}$. Section 13.2 describes the plaintext extractor used in this game hop.

Adjustments: After successful execution of $\mathsf{VerifyMixDecOnline}_h$ or VerifyMixDecOffline, the Challenger manually check that the lists $\mathbf{c}_{\mathsf{mix},j}$ and $\mathbf{c}_{\mathsf{Dec},j}$ contain identical plaintexts for each $j < h$. First, the Challenger reconstructs the remaining election public keys:

$$\overline{\mathtt{EL}}_{\mathtt{pk},j} \leftarrow \mathsf{CombinePublicKeys}(\mathtt{EL}_{\mathtt{pk},j}, \ldots, \mathtt{EL}_{\mathtt{pk},m+1})$$

for each $1 \leq j < h$. Then, the Challenger checks that $\mathsf{Extract}(\mathbf{c}_{i,j}, \overline{\mathtt{EL}}_{\mathtt{pk},j}) = \mathsf{Extract}(\mathbf{c}'_{i,j}, \overline{\mathtt{EL}}_{\mathtt{pk},j+1})$, where $\mathbf{c}_{i,j} \in \mathbf{c}_{\mathsf{mix},j}$ and $\mathbf{c}'_{i,j} \in \mathbf{c}_{\mathsf{Dec},j}$. If for some pair (i,j) the equality does not hold, the Challenger aborts the game.

Claim:

$$|\mathsf{Adv}^{\mathsf{SPVS,priv.2}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{SPVS,priv.1}}_{\mathcal{A}}| \leq \mathtt{N}_{\mathtt{bb}} \cdot \hat{\mathtt{N}}_{\mathtt{C}} \cdot m \cdot \mathsf{Adv}^{\mathsf{dec,sSOUND}}_{\mathcal{B}_2}$$

where $\mathtt{N}_{\mathtt{bb}}$ is a number of ballot boxes used in the election, $\hat{\mathtt{N}}_{\mathtt{C}}$ is the number of mixed votes, m is the number of online control components and $\mathcal{B}_2$ is an adversary against the soundness of the decryption proof system.

Reasoning: Before mixing and decrypting the votes of a specific ballot box (up to $\mathtt{N}_{\mathtt{bb}}$ per election), the honest control component verifies the decryption proofs of the preceding control components in the algorithms $\mathsf{VerifyMixDecOnline}_h$ or VerifyMixDecOffline respectively. Thus, the honest control component has to verify $\mathtt{N}_{\mathtt{bb}} \cdot \hat{\mathtt{N}}_{\mathtt{C}} \cdot (h-1)$ decryption proofs. At most, if the Tally control component is the only trustworthy control component, the challenger must manually check $\mathtt{N}_{\mathtt{bb}} \times \hat{\mathtt{N}}_{\mathtt{C}} \times (h-1)$ decryption proofs.

The Adversary can notice the extra check only by generating a decryption proof for an invalid statement without the auditors noticing; in that case, the algorithms $\mathsf{VerifyMixDecOnline}_h$ or VerifyMixDecOffline return $\top$ when verifying a bogus decryption proof. However, the soundness of the decryption proof guarantees that this would happen only with negligible probability (see Def. 3).

Effect on the future games: From this game on, the corrupted control components must shuffle and decrypt the votes in the tally phase correctly.

Game priv.3

Objective: In this game, the Challenger aborts when receiving a ballot $\mathtt{b}_{\mathtt{id}} = (\mathtt{vc}_{\mathtt{id}}, \mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \pi_{\mathsf{Exp}}, \pi_{\mathsf{EqEnc}})$ with valid proofs but incorrect relations between the ciphertexts $(\gamma_1, \phi_{1,0})$, $\widetilde{\mathtt{E1}}$, and $\mathtt{E2}$ (i.e., $\widetilde{\mathtt{E1}} \neq (\gamma_1, \phi_{1,0})^{\mathtt{k}_{\mathtt{id}}}$ or $\widetilde{\mathtt{E2}}$ and $\widetilde{\mathtt{E1}}$ do not encrypt the same message). Moreover, the Challenger aborts when the NIZK proofs $\pi_{\mathsf{Exp}}, \pi_{\mathsf{EqEnc}}$ do not verify.

Adjustments: After the successful execution of $\mathsf{VerifyBallotCCR}_{\mathtt{h}}$ (in case an online control component is honest) or VerifyVotingClientProofs (in case the offline Tally control component is honest), the Challenger manually checks that $\widetilde{\mathtt{E1}} = (\gamma_1, \phi_{1,0})^{\mathtt{k}_{\mathtt{id}}}$ and $\widetilde{\mathtt{E2}}$ and $\widetilde{\mathtt{E1}}$ encrypt the same message. First, the Challenger parses $\mathtt{E2}$ as $(c_{2,0}, c_{2,1}, \ldots, c_{2,\psi})$ and computes $\widetilde{\mathtt{E2}} \leftarrow \left(c_{2,0}, \prod_{i=1}^{\psi} c_{2,i}\right)$. Then, the Challenger checks that $\widetilde{\mathtt{E1}} = (\gamma_1, \phi_{1,0})^{\mathtt{k}_{\mathtt{id}}}$ and $\mathsf{Extract}(\widetilde{\mathtt{E1}}, \mathtt{EL}_{\mathtt{pk}}) = \mathsf{Extract}(\widetilde{\mathtt{E2}}, \prod_{i=1}^{\psi} \mathtt{pk}_{\mathrm{CCR},i})$. If one of the condition fails, the Challenger aborts the game.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{priv}.3} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{priv}.2}| \leq (\mathtt{N_E} - \mathtt{N_{E}}_h) \cdot (\mathsf{Adv}_{\mathcal{B}_3}^{\mathsf{Exp},\mathsf{sSOUND}} + \mathsf{Adv}_{\mathcal{B}_4}^{\mathsf{EqEnc},\mathsf{sSOUND}})$$

where $\mathtt{N_E}$ is the number of voters, $\mathtt{N_{E}}_h$ is the number of honest voters, $\mathcal{B}_3$ is an adversary against the simulation soundness property of the exponentiation proof system Exp, $\mathcal{B}_4$ is an adversary against the simulation soundness property of the plaintext equality proof system EqEnc.

Reasoning: The Adversary can notice the Challenger's extra checks only when providing altered ballots (up to $(\mathtt{N_E} - \mathtt{N_{E}}_h)$) without the honest control component noticing. However, the soundness of the ballot proofs guarantees that this happens only with negligible probability. An altered ballot in this context means that either the ballot proofs are invalid, the inputs to the ballot proofs were modified, or the verification card secret key $\mathtt{k_{id}}$, the election public key $\mathtt{EL_{pk}}$ or one of the CCR_j Choice Return Codes encryption public key $\mathbf{pk}_{\mathrm{CCR}_j}$ are incorrect, or $\widetilde{\mathtt{E1}} \neq (\gamma_1, \phi_{1,0})^{\mathtt{k_{id}}}$, or $\widetilde{\mathtt{E2}}$ and $\widetilde{\mathtt{E1}}$ encrypt different messages.

The honest online control component verifies the correctness of the ballot prior to accepting it (see oracle $\mathcal{O}\mathsf{HonestPartialDecPCC}$ in figure 19) while the honest Tally control component verifies it in the algorithm $\mathsf{VerifyVotingClientProofs}$ (see the oracle $\mathcal{O}\mathsf{HonestMixDecOfflinePrivacy}$ in figure 36). Both verifications use the public keys $\mathtt{EL_{pk}}$, $\mathtt{K_{id}}$, and $\{\mathbf{pk}_{\mathrm{CCR}_j}\}_{j=1}^{m}$ received from the trustworthy setup component; therefore, the honest component would never accept a ballot proof computed using different public keys.

Effect on Future Games: The verification of the ballot proofs prevents ballot-copying or the acceptance of votes from non-eligible voters. The ballot proofs include auxiliary data $\mathbf{i}_{\mathtt{aux}}$ containing the verification card ID $\mathtt{vc_{id}}$ and a unique verification card public key $\mathtt{K_{id}}$. The honest control component accepts only one ballot per $\mathtt{vc_{id}}$ and requires a valid verification card public key $\mathtt{K_{id}}$.

Moreover, the auxiliary data $\mathbf{i}_{\mathtt{aux}}$ contains the ciphertext elements of $\mathtt{E1}$ embedding the write-ins: $(\phi_{1,1}, \ldots, \phi_{1,\hat{\delta}})$. Therefore, the adversary cannot mark the ballots by modifying $(\phi_{1,1}, \ldots, \phi_{1,\hat{\delta}})$; otherwise, the ballot proofs would fail to validate.

Game priv.4

Objective: In this game, the Challenger ignores the responses from all control components; instead, it extracts the secret keys from the NIZK proofs and uses them to construct the Return Codes Mapping table **CMtable** and the allow lists $\mathsf{L_{pCC}}$ and $\mathsf{L_{lVCC}}$.

Adjustments: For constructing the **CMtable** the Challenger needs to know $\left\{\mathtt{hpCC}_{\mathtt{id},i}^{\sum_{j=1}^{m}\mathtt{k}_{j,\mathtt{id}}}\right\}_{i=1}^{n}$ and $\mathtt{hCK}_{\mathtt{id}}^{\sum_{j=1}^{m}\mathtt{kc}_{j,\mathtt{id}}}$ for all honest voters $\mathtt{id} \in \mathcal{ID}_h$ i.e. the values that the setup component should obtain after combining and decrypting the control component responses—assuming that all control components computed them correctly. To obtain those values, the Challenger needs the secret keys of the adversarial control components for all voters $\mathtt{id} \in \mathcal{ID}_h$ i.e. $\{\mathtt{k}_{j,\mathtt{id}}\}_{j=1,j\neq h}^{m}$ and $\{\mathtt{kc}_{j,\mathtt{id}}\}_{j=1,j\neq h}^{m}$.

Therefore, the Challenger will use the extractor $\mathcal{E}_{\mathcal{A}}$ (see Definition 8) of the exponentiation proof system on each of the NIZK proofs corresponding to honest voters $\boldsymbol{\pi}_{\mathsf{expPCC},j}$ ($\mathtt{N_E}h$ proofs per component) and $\boldsymbol{\pi}_{\mathsf{expCK},j}$ ($\mathtt{N_E}h$ proofs per component) done by the adversarial control components (up to m). As a result, the Challenger obtains the witnesses $\left\{\mathtt{k}_{j,\mathtt{id}}, \mathtt{kc}_{j,\mathtt{id}}\right\}_{j=1,j\neq h}^{m}$.

After that, the following adjustments are made:

- The algorithm CombineEncLongCodeShares (see section 4.1.5 of [37]) will use the extracted keys and the honest control component's private keys $\mathtt{k}_{h,\mathtt{id}}$ and $\mathtt{kc}_{h,\mathtt{id}}$ to compute $\mathtt{lVCC}^{\star}_{\mathtt{id},j} \leftarrow \mathtt{hCK}_{\mathtt{id}}^{\mathtt{kc}_{j,\mathtt{id}}} \quad \forall j \in (1,\ldots,m) \quad \forall \mathtt{id} \in \mathcal{ID}$. Then the algorithm will use the computed $\mathtt{lVCC}^{\star}_{\mathtt{id},j}$ to generate $\mathtt{pVCC}^{\star}_{\mathtt{id}}$ and the allow list $\mathsf{L}^{\star}_{\mathtt{lVCC}}$.
- The algorithm GenCMTable (see section 4.1.6 of [37]) will use the extracted keys and the honest control component's private keys $\mathtt{k}_{h,\mathtt{id}}$ and $\mathtt{kc}_{h,\mathtt{id}}$ to compute the values $(\mathtt{pCC}^{\star}_{\mathtt{id},1},\ldots,\mathtt{pCC}^{\star}_{\mathtt{id},n}) \leftarrow \left\{\mathtt{hpCC}_{\mathtt{id},i}^{\sum_{j=1}^{m}\mathtt{k}_{j,\mathtt{id}}}\right\}_{i=1}^{n}$. Then it will use the computed values $(\mathtt{pCC}^{\star}_{\mathtt{id},1},\ldots,\mathtt{pCC}^{\star}_{\mathtt{id},n})$ and $\mathtt{pVCC}^{\star}_{\mathtt{id}}$ (from the previous function) to construct the correct Return Codes Mapping table $\mathbf{CMtable}^{\star}$.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{priv}.4} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS},\mathsf{priv}.3}| \leq 2 \cdot \mathtt{N_E} \cdot m \cdot \epsilon_{ext}$$

where $\mathtt{N_E}$ is the number of voters, m is the total number of control components and ϵ_{ext} is given by the weak simulation extractability property of the Exponentiation proof system Exp.

Reasoning: If the verification of the control components' NIZK proofs is not successful, then the game aborts (see Figure 37). Therefore, the Adversary can only win if VerifyConfigPhase is successful, which implies that all NIZK proofs verify.
The Adversary can notice the change only if some of the adversarial NIZK proofs $\{\boldsymbol{\pi}_{\mathsf{expPCC},j}, \boldsymbol{\pi}_{\mathsf{expCK},j}\}_{j=1,j\neq h}^{m}$ verify successfully but $\mathcal{E}_{\mathcal{A}}$ could not extract the corresponding witness. However, the probability of not extracting the witness from a valid proof is bounded by the weak simulation extractability property of the Exponentiation proof system Exp.

Game priv.5

Objective: In this game the Challenger, playing on behalf of the Setup Component, outputs random elements instead of ciphertexts.

Adjustments: The algorithm GenVerDat samples $\mathbf{c}_{\mathtt{pCC}}, \mathbf{c}_{\mathtt{ck}}$ at random, in particular independently from values $\mathtt{hpCC}_{\mathtt{id},1},\ldots,\mathtt{hpCC}_{\mathtt{id},n},\mathtt{hCK}_{\mathtt{id}}$ for all honest voters.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,priv.5}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,priv.4}}| \leq 2 \cdot \mathtt{N_E}_h \cdot \mathsf{Adv}_{\mathcal{B}_5}^{\mathsf{ElGamal,INDCPA}}$$

where $\mathtt{N_E}$ is the number of voters and $\mathcal{B}_5$ is an adversary against the IND-CPA property of the ElGamal encryption scheme.

Reasoning: The Adversary can notice the change only if it can distinguish between randomly sampled group elements and ElGamal ciphertexts; which amounts to winning in the IND-CPA game.

Reduction: Technically, the reduction unfolds in two steps: One for the ciphertexts $\mathbf{c}_{\mathtt{pCC}}$ and one for $\mathbf{c}_{\mathtt{ck}}$; hence the factor of 2 in the bound. We describe only one of these reductions; the other can be done by analogy.

Suppose the Adversary $\mathcal{A}$ can detect the change from priv.4 to priv.5: i.e., it outputs 0 if there was no change and 1 otherwise. Then we can construct a new Adversary $\mathcal{B}$ that breaks the IND-CPA property of ElGamal:

1. The Adversary $\mathcal{B}$ sends to the IND-CPA game two messages $\mathbf{m}_0 = \left\{\mathsf{HashAndSquare}(\mathrm{p}_i^{\mathtt{kid}})\right\}_{i=1}^{n}$ and $\mathbf{m}_1 = \{R_i\}_{i=1}^{n}$, where $R_i \xleftarrow{\$} \mathbb{Q}_p$.
2. The IND-CPA game replies with $\mathbf{c}_\beta$, which is either an encryption of $\mathbf{m}_0$ (same as in priv.4) or an encryption of random messages i.e. a vector of random elements (same as in priv.5).
3. The Adversary $\mathcal{B}$ repeats the query $\mathtt{N_E}_h$ times to get $\mathbf{c}_{\mathtt{pCC}}^{\beta}$, which is either identical to the one from priv.4 or priv.5, and sends $\mathbf{c}_{\mathtt{pCC}}^{\beta}$ to the Adversary $\mathcal{A}$.
4. The Adversary $\mathcal{A}$ replies with a coin β
5. The Adversary $\mathcal{B}$ forwards β to the IND-CPA game.

Adversary $\mathcal{B}$ wins in at least the cases where Adversary $\mathcal{A}$ wins, therefore $\mathsf{Adv}_{\mathcal{B}} \geq \mathsf{Adv}_{\mathcal{A}}$. However, as $\mathcal{B}$ attempts to break IND-CPA security of multi-recipient ElGamal, it must also hold that $\mathsf{Adv}_{\mathcal{B}} \leq \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{ElGamal,INDCPA}}$ for any Adversary $\mathcal{B}'$. We conclude that $\mathsf{Adv}_{\mathcal{A}} \leq \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{ElGamal,INDCPA}}$.

Effect on the future games: After this hop, the view of the Adversary changes to the one in Figure 39.

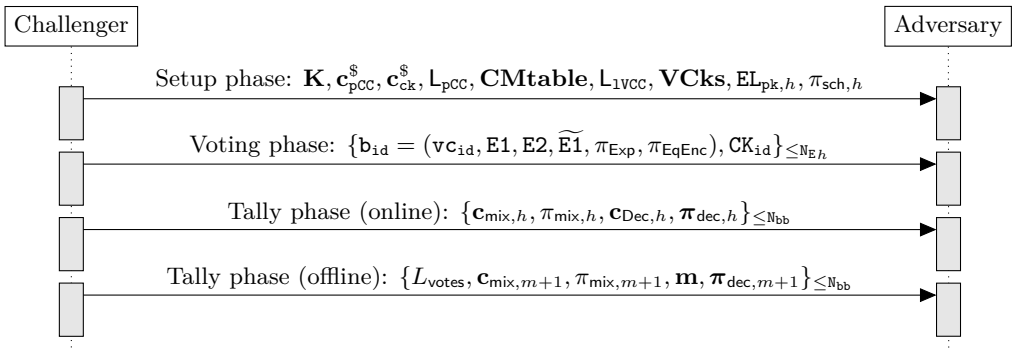

Figure 39: The Challenger's outputs associated with the honest parties' secret values after randomizing the values of SetupVoting phase. The randomly selected or simulated values are marked with a $ sign.

Game priv.6

Objective: In this game, the Challenger simulates the Schnorr proof done by the honest online control component during the SetupTally process. If no online control component is honest, this hop can be skipped since the trustworthy setup component generates the electoral board key pair $(\mathtt{EB}_{\mathtt{pk}}, \mathtt{EB}_{\mathtt{sk}})$ for the Tally control component without generating a Schnorr proof.

Adjustments: Algorithm $\mathsf{SetupTallyCCM}_h$ will use the canonical simulators $\mathcal{S}_{\mathsf{sch}}$ to generate the Schnorr proof $\pi_{\mathsf{sch},h}$ containing the CCM_h election secret key $\mathtt{EL}_{\mathtt{sk},h}$.

Claim:

$$|\mathsf{Adv}^{\mathsf{SPVS,priv.6}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{SPVS,priv.5}}_{\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{sch,NIZK}}_{\mathcal{B}_6}$$

where $\mathcal{B}_6$ is an adversary against the zero-knowledge property of the Schnorr proof system.

Reasoning: The Adversary notices the change only if it can distinguish between the real and simulated Schnorr proofs. However, this probability is bounded by the zero-knowledge property of the Schnorr proof system (see section 7).

Game priv.7

Objective: In this game, the Challenger simulates all shuffle and decryption proofs done by the honest control component.

Adjustments: The algorithm $\mathsf{MixDecOnline}_h$ (in oracle $\mathcal{O}\mathsf{HonestMixDecOnlinePrivacy}$ from Figure 36) and $\mathsf{MixDecOffline}$ (in oracle $\mathcal{O}\mathsf{HonestMixDecOfflinePrivacy}$ from Figure 36) will use the canonical simulators $\mathcal{S}_{\mathsf{dec}}$ (see section 7) and $\mathcal{S}_{\mathsf{Mix}}$ (see section 8.5) to generate all decryption $\boldsymbol{\pi}_{\mathsf{dec},h}$ and shuffle $\pi_{\mathsf{mix},h}$ proofs.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,priv.7}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,priv.6}}| \leq \mathtt{N}_{\mathtt{bb}} \cdot (\hat{\mathtt{N}}_{\mathtt{C}} \cdot \mathsf{Adv}_{\mathcal{B}_7}^{\mathsf{dec,NIZK}} + \mathsf{Adv}_{\mathcal{B}_8}^{\mathsf{mix,NIZK}})$$

where $\mathtt{N}_{\mathtt{bb}}$ is the number of ballot boxes used in the election, $\hat{\mathtt{N}}_{\mathtt{C}}$ is the number of mixed votes, $\mathcal{B}_7$ is an adversary against the zero-knowledge property of the Decryption proof system dec and $\mathcal{B}_8$ is an adversary against the zero-knowledge property of the shuffle argument mix.

Reasoning: The Adversary notices the change only if it can distinguish between the real and simulated shuffle and decryption proofs (up to $\mathtt{N}_{\mathtt{bb}}$ per election). However, this probability is bounded by the zero-knowledge property of the Decryption proof system (see section 7) and the verifiable mixnet (see section 8.5).

Game priv.8

Objective: In this game, the Challenger simulates the honest voting client's ballot NIZK proofs.

Adjustments: The algorithm $\mathsf{CreateVote}$ (in oracle $\mathcal{O}\mathsf{HonestVote}$ from Figure 35) uses the canonical simulators $\mathcal{S}_{\mathsf{Exp}}$ and $\mathcal{S}_{\mathsf{EqEnc}}$ to simulate π_{Exp} and π_{EqEnc} for all honest voters.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,priv.8}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,priv.7}}| \leq \mathtt{N}_{\mathtt{E}h} \cdot (\mathsf{Adv}_{\mathcal{B}_9}^{\mathsf{Exp,NIZK}} + \mathsf{Adv}_{\mathcal{B}_{10}}^{\mathsf{EqEnc,NIZK}})$$

where $\mathtt{N}_{\mathtt{E}h}$ is the number of honest voters, $\mathcal{B}_9$ and $\mathcal{B}_{10}$ are adversaries against the zero-knowledge properties of the Exp and EqEnc proof systems.

Reasoning: The Adversary notices the change only if it can distinguish between the real and simulated exponentiation and plaintext equality proofs (up to $\mathtt{N}_{\mathtt{E}h}$). However, this probability is bounded by the zero-knowledge property of the Exponentiation and Plaintext Equality proof system (see section 7).

Effect on the future games: After this hop, all NIZK proofs generated by the Challenger are simulated, and the view of the Adversary changes to the one in Figure 40.

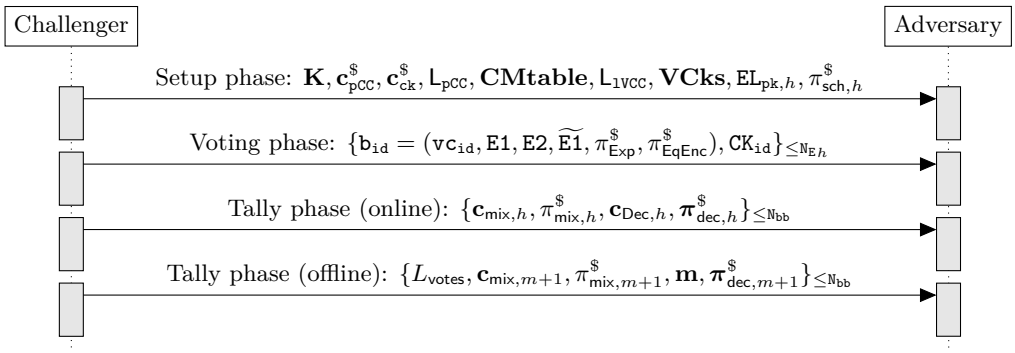

Figure 40: All Challenger's output associated with honest parties' private values after simulating all NIZK proofs. The randomly selected or simulated values are marked with a $ sign.

Game priv.9

Objective: In this game, the Challenger uses a random key to create the Verification Card Keystore $\mathtt{VCks}_{\mathtt{id}}$ instead of deriving it from the Start Voting Key $\mathtt{SVK}_{\mathtt{id}}$ for all honest voters.

Adjustments: To support this change, we make the following adjustments for all honest voters:

- The algorithm GenCredDat selects a random key $K^{\$}$ from $\mathcal{K}$ and uses it to create the Verification Card Keystore $\mathtt{VCks}_{\mathtt{id}}$: $\mathtt{VCks}_{\mathtt{id}} \leftarrow \mathsf{Enc}_{\mathsf{s}}(\mathtt{k}_{\mathtt{id}}; K^{\$})$.
- The oracle $\mathcal{O}$HonestVote (see Figure 35) does not use the Start Voting Key $\mathtt{SVK}_{\mathtt{id}}$ to derive the Keystore symmetric encryption key $\mathtt{KSkey}_{\mathtt{id}}$ and instead uses $K^{\$}$ directly to open the Verification Card Keystore $\mathtt{VCks}_{\mathtt{id}}$.

Claim:

$$|\mathsf{Adv}^{\mathsf{SPVS},\mathsf{priv}.9}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{SPVS},\mathsf{priv}.8}_{\mathcal{A}}| \leq \mathtt{N}_{\mathtt{E}h} \cdot \frac{1}{2^{\lambda_{\mathtt{KSkey}}}}$$

where $\mathtt{N}_{\mathtt{E}h}$ is the number of honest voters, and $\lambda_{\mathtt{KSkey}}$ is the bit strength of the Keystore symmetric encryption key $\mathtt{KSkey}_{\mathtt{id}}$.

Reasoning: Before this game, the Challenger derived the Keystore symmetric encryption key $\mathtt{KSkey}_{\mathtt{id}}$ by applying a memory-hard key derivation function to the Start Voting Key $\mathtt{SVK}_{\mathtt{id}}$. The Keystore symmetric encryption key $\mathtt{KSkey}_{\mathtt{id}}$ opens the Verification Card Keystore $\mathtt{VCks}_{\mathtt{id}}$. The Adversary notices the change only by distinguishing the derived Keystore symmetric encryption key $\mathtt{KSkey}_{\mathtt{id}}$ from a truly random key $K^{\$}$ for one of the $\mathtt{N}_{\mathtt{E}h}$ voters. $\lambda_{\mathtt{KSkey}}$ designates the bit strength of the Keystore symmetric encryption key $\mathtt{KSkey}_{\mathtt{id}}$, which depends on the Start Voting Key's code space $|\mathcal{C}_{svk}|$ and the parameters of the memory-hard key derivation function (see section 19.2).

Game priv.10

Objective: In this game, the Challenger publishes a random value instead of the actual Verification Card Keystore $\texttt{VCks}_{\texttt{id}}$ for all honest voters.

Adjustments: To support this change, we make the following adjustments for all honest voters:

- The algorithm GenCredDat randomly samples a ciphertext $\texttt{VCks}^{\$}_{\texttt{id}}$ from the symmetric ciphertext space $\mathcal{C}$.
- The oracle $\mathcal{O}$HonestVote (see Figure 35) does not execute GetKey and instead direcly uses its knowledge of the verification card secret key $\texttt{k}_{\texttt{id}}$ to generate a ballot.

Claim:

$$|\mathsf{Adv}^{\mathsf{SPVS,priv.10}}_{\mathcal{A}} - \mathsf{Adv}^{\mathsf{SPVS,priv.9}}_{\mathcal{A}}| \leq \mathtt{N}_{\mathtt{E}h} \cdot \mathsf{Adv}^{\mathsf{SEnc,INDCPA}}_{\mathcal{B}_{11}}$$

where $\mathcal{B}_{11}$ is an adversary against the IND-CPA property of the symmetric encryption scheme.

Reasoning: After the last hop, all honest voters' keys are symmetrically encrypted with random keys. Therefore, the Adversary can notice the change only if it can distinguish between ciphertexts and random elements, which amounts to winning the IND-CPA game.

Effect on the future games: After this hop, the view of the Adversary changes to the one in Figure 41.

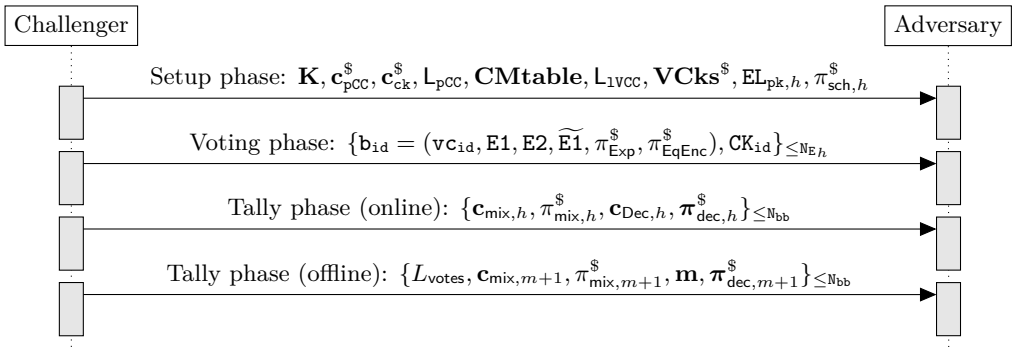

Figure 41: All Challenger's output associated with honest parties' private values before applying ESGSP. The randomly selected or simulated values are marked with a $ sign.

Game priv.11

Objective: In this game, the Challenger randomizes the values that result from the voting client's exponentiation to the verification card secret key $\texttt{k}_{\texttt{id}}$ for all honest voters.

Adjustments: To randomize the values $\mathtt{K_{id}} = g^{\mathtt{k_{id}}}$, $\mathtt{E2} = \mathsf{GetCiphertext}\,(\mathbf{pk}_{\mathrm{CCR}}, r', \mathbf{pCC}_{\mathtt{id}})$, $\mathbf{pCC}_{\mathtt{id}} = (\mathrm{p}_1^{\mathtt{k_{id}}}, \ldots, \mathrm{p}_{\psi}^{\mathtt{k_{id}}})$, $\widetilde{\mathtt{E1}} = (\gamma_1, \phi_{1,0})^{\mathtt{k_{id}}}$, $\mathtt{CK_{id}} = \mathsf{HashAndSquare}(\mathtt{BCK_{id}})^{\mathtt{k_{id}}}$ that rely on the verification card secret key $\mathtt{k_{id}}$, we make the following adjustments for all honest voters:

- The algorithm $\mathsf{GenVerDat}$ samples the verification card public key $\mathtt{K}_{\mathtt{id}}^{\$}$, the vector of partial Choice Return Codes $\mathbf{pCC}_{\mathtt{id}}^{\$}$, and the Confirmation Key $\mathtt{CK}_{\mathtt{id}}^{\$}$ at random for all honest voters. Then it proceeds with the generation of the ciphertexts $\mathbf{c}_{\mathtt{pCC}}, \mathbf{c}_{\mathtt{ck}}$ and the partial Choice Return Codes allow list $\mathsf{L}_{\mathtt{pCC}}$ using the sampled random values instead of the computed ones. The Challenger adds $(\mathtt{id}, \mathbf{pCC}_{\mathtt{id}}^{\$}, \mathtt{CK}_{\mathtt{id}}^{\$})$ to $\mathsf{L}_{\mathtt{sampled}}$.
- The algorithm $\mathsf{CreateVote}$ (in oracle $\mathcal{O}\mathsf{HonestVote}$ from Figure 35) samples the ciphertext $\widetilde{\mathtt{E1}}^{\$}$ at random. Then it uses the value $\mathbf{pCC}_{\mathtt{id}}^{\$}$ from $\mathsf{L}_{\mathtt{sampled}}$ to produce $\mathtt{E2}^{\$} \leftarrow \mathsf{GetCiphertext}\left(\mathbf{pk}_{\mathrm{CCR}}, r', \mathbf{pCC}_{\mathtt{id}}^{\$}\right)$.
- The algorithm $\mathsf{CreateConfirmMessage}$ ($\mathcal{O}\mathsf{ConfirmBallot}$ from Figure 35) outputs the Confirmation Key $\mathtt{CK}_{\mathtt{id}}^{\$}$ from $\mathsf{L}_{\mathtt{sampled}}$.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,priv.11}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,priv.10}}| \leq \mathtt{N}_{\mathtt{E}h} \cdot \mathsf{Adv}_{\mathcal{B}_{12}}^{\mathbb{Q}_p,\mathsf{ESGSP}}$$

where $\mathcal{B}_{12}$ is an adversary against the ESGSP problem (see Definition 9.2).

Reasoning: The Adversary can query the oracle $\mathcal{O}\mathsf{HonestVote}$ only once for a given voter $\mathtt{id}$. Therefore, the Adversary can distinguish the two games only by distinguishing between the honestly computed values $\mathtt{K_{id}}, \widetilde{\mathtt{E1}}, \mathtt{E2}, \mathbf{pCC}_{\mathtt{id}}, \mathtt{CK_{id}}$ and randomly sampled elements from $\mathbb{Q}_p$, which amounts to breaking the ESGSP problem (see Definition 9.2).

Reduction: Suppose the Adversary $\mathcal{A}$ can detect the change from priv.10 to priv.11: i.e., $\mathcal{A}$ outputs 0 if there was no change and 1 otherwise. Then we can construct a new Adversary $\mathcal{B}$ that breaks the ESGSP assumption in $\mathbb{Q}_p$:

1. The Adversary $\mathcal{B}$ runs the ESGSP game for the group $\mathbb{Q}_p$ with a random generator $\mathtt{E1}$ and receives back a tuple of $2 \cdot (n+3)$ elements:

   $$(\mathtt{E1}, x, g_1, \ell_1, \ldots, \ell_{n+1}, \mathtt{E1}^{s_0}, x^{s_1}, g_1^{s_2}, \ell_1^{s_3}, \ldots, \ell_{n+1}^{s_{n+2}})$$

   where $s_0 = s_1 = \cdots = s_{n+3} = s$ or not.

   Now, the Adversary $\mathcal{B}$ uses the tuple to generate a view for the Adversary $\mathcal{A}$:

   - It uses x^{s_1} as the public key $\mathtt{K_{id}}$ for the voter $\mathtt{id}$. x equals g since x is the smallest integer $x > 1$ that is a generator of $\mathbb{Q}_p$.

- Then it programs the hash oracle $\mathsf{HashAndSquare}$ to output g_1 whenever it receives $\mathtt{BCK_{id}}$ as input.
- After that, it makes the Setup Component to set $\mathbf{pCC}_{\mathtt{id}}$ to $\ell_1^{s_3}, \ldots, \ell_n^{s_{n+2}}$, where $\ell_1, \ldots, \ell_{n+1}$ are the first small primes that are group elements $\mathbb{Q}_p$, and the value $\mathtt{CK_{id}}$ to $g_1^{s_1}$.
- Finally, it sets $\widetilde{\mathtt{E1}} = \mathtt{E1}^{s_0}$.

2. The Adversary $\mathcal{B}$ repeats the query $\mathtt{N_E}$ times to get $(\mathbf{K}, \{\mathtt{E1}, \mathtt{E2}, \widetilde{\mathtt{E1}}, \mathtt{CK_{id}}\}_{\leq \mathtt{N_{E_h}}})$. Then it sends $\mathbf{K}$ to the Adversary $\mathcal{A}$ and uses the rest of the data to reply to the $\mathcal{A}$'s queries.

   If $s_0 = s_1 = \cdots = s_{n+2} = s$, then:

   - $\mathtt{K_{id}} = x^s = g^s$
   - $\widetilde{\mathtt{E1}} = \mathtt{E1}^s$
   - $\mathtt{CK_{id}} = g_1^s = \mathsf{HashAndSquare}(\mathtt{BCK_{id}})^s$
   - $\mathbf{pCC}_{\mathtt{id}} = (\ell_1^s, \ldots, \ell_n^s)$

   which is identical to the Challenger's output in priv.10. Otherwise, the view is identical to the Challenger's output in priv.11.

3. The Adversary $\mathcal{A}$ replies with a coin β
4. The Adversary $\mathcal{B}$ forwards β to the ESGSP game.

Adversary $\mathcal{B}$ wins in at least the cases where Adversary $\mathcal{A}$ wins, therefore $\mathsf{Adv}_{\mathcal{B}} \geq \mathsf{Adv}_{\mathcal{A}}$. However, as $\mathcal{B}$ attempts to break ESGSP problem in $\mathbb{Q}_p$, it must also hold that $\mathsf{Adv}_{\mathcal{B}} \leq \mathsf{Adv}_{\mathcal{B}'}^{\mathbb{Q}_p, \mathsf{ESGSP}}$ for any Adversary $\mathcal{B}'$. We conclude that $\mathsf{Adv}_{\mathcal{A}} \leq \mathsf{Adv}_{\mathcal{B}'}^{\mathbb{Q}_p, \mathsf{ESGSP}}$.

Effect on the future games: We arrived at the point where all values output by the Challenger are changed to random elements, apart from the encrypted vote $\mathtt{E1}$ and the election public key share $\mathtt{EL_{pk,h}}$. The ciphertext $\mathtt{E1}$ contains the message $\mathrm{p}_{\mathtt{id}}^{\beta}$, the only value which still depends on the coin β. The view of the Adversary changes to the one in Figure 42.

Note, that the part of the Return Codes Mapping table **CMtable** that correspond to honest voters is also marked as independent from the Challenger's private keys (i.e. $\mathbf{sk}_{\mathtt{setup}}, \mathtt{EL_{sk,h}}, \{\mathtt{k_{id}}, \mathtt{SVK_{id}}, \mathbf{CC}_{\mathtt{id}}, \mathtt{BCK_{id}}, \mathtt{VCC_{id}}\}_{\mathtt{id} \in \mathcal{ID}_h}$). To see why, recall that the part of $\mathbf{CMtable}^{\$}$ that corresponds to a voter $\mathtt{id}$ is normally constructed based on values $\mathtt{CK_{id}} = \mathsf{HashAndSquare}(\mathtt{BCK_{id}})^{\mathtt{k_{id}}}$ and $\mathbf{pCC}_{\mathtt{id}} = (\mathrm{p}_1^{\mathtt{k_{id}}}, \ldots, \mathrm{p}_{\psi}^{\mathtt{k_{id}}})$, which depend on the $\mathtt{k_{id}}$. However, after the hop priv.11, those values are changed to random elements. Therefore, the resulting $\mathbf{CMtable}^{\$}$ is independent from the Challenger's private keys and selections.

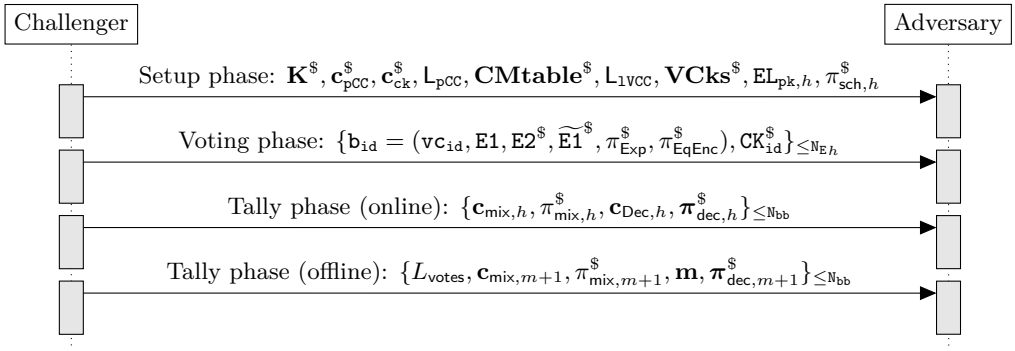

Figure 42: All Challenger's output associated with honest parties' private values before applying ESGSP. The randomly selected or simulated values are marked with a $ sign.

Game priv.12

Objective: The Challenger changes the adversarial view from β to $1 - \beta$.

Adjustments: The Challenger changes $\mathrm{p}_{\mathtt{id}}^{\beta}$ to $\mathrm{p}_{\mathtt{id}}^{1-\beta}$ for all honest voters $\mathtt{id} \in \mathtt{N}_{\mathtt{E}h}$ in the $\mathcal{O}$HonestVote (see Figure 35) oracle.

Claim:

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,priv.12}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,priv.11}}| \leq \mathtt{N}_{\mathtt{E}h} \cdot \mathsf{Adv}_{\mathcal{B}_{13}}^{\mathsf{ElGamal,INDCPA}}$$

where $\mathcal{B}_{13}$ is an adversary against the IND-CPA property of the distributed ElGamal encryption scheme [14].

Reasoning: The Adversary cannot control all control components, therefore it cannot learn the election secret key $\mathtt{EL}_{\mathtt{sk}}$. Moreover, the Adversary does not know the encryption randomness of the honest voters' ballots. Thus, it can detect the change only by distinguishing the ciphertexts for $\mathrm{p}_{\mathtt{id}}^{\beta}$ and $\mathrm{p}_{\mathtt{id}}^{1-\beta}$; which amounts to breaking the semantic security (IND-CPA) of the ElGamal encryption scheme (see section 3).

Wrapping up: In priv.12, the Adversary can no longer distinguish $\beta = 0$ from $\beta = 1$ resulting in $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SPVS,priv.12}} \approx 0$. Collecting the bounds of the hops concludes the proof.

Part V

Parameters and Conclusion

19 Choice of Parameters

19.1 Mathematical Groups, Cryptographic Parameters, and Security Level

We instantiate the ElGamal encryption and Pedersen commitment scheme over the group of quadratic residues $\mathbb{Q}_p \subset \mathbb{Z}_p$. The cryptographic primitives specification defines the concrete choice of parameters for different security levels and describes the verifiable selection of the group parameters [36].

19.2 Voters' codes

Section 1.2 explains the role of the codes figuring on the voter's printed voting card. The length of these codes requires a careful balance between usability and security. Therefore, we justify the voters' codes length separately to the system's security level (see section 19.1). We distinguish between Return Codes (Choice Return Codes and Vote Cast Return Codes), the Ballot Casting Key, and the Start Voting Key. The *Return Codes* space should prevent the adversary from correctly guessing them. Therefore, we define the deterrence factor ϵ as the honest party's probability to detect an adversary's attempt to cheat. To avoid confusion between Choice Return Codes and the Vote Cast Return Codes, we use longer codes for the Vote Cast Return Code.

Code/Key	**Space**	**Deterrence factor**
short Choice Return Code $\texttt{CC}_{\texttt{id},i}$	4-digit numbers. Space $\mathcal{C}_{cc}$ has 10^4 values.	$\epsilon = 1 - 2^{\log_2(10^4)}$ $\geq 1 - 2^{-13}$
short Vote Cast Return Code $\texttt{VCC}_{\texttt{id}}$	8-digit numbers. Space $\mathcal{C}_{vcc}$ has 10^8 values.	$\epsilon = 1 - 2^{\log_2(10^8)}$ $\geq 1 - 2^{-26}$

Table 8: Size of the Return Codes

The *Ballot Casting Key* allows the voter to confirm her vote. We allow the voter five confirmation attempts. Furthermore, for usability reasons, the Ballot Casting Key's length must differ from the Return Codes' size. The *Start Voting Key* opens the Verification Card Keystore $\texttt{VCks}_{\texttt{id}}$. Therefore, we must use a size that is manageable for voters while preventing brute-force attacks. Moreover, we apply a memory-hard key derivation function Argon2id (see section 6) for additional security. The section *Usability* in the system specification defines the length and alphabet of the Start Voting Key $\texttt{SVK}_{\texttt{id}}$ and Ballot Casting Key $\texttt{BCK}_{\texttt{id}}$ as well as the parametrization of Argon2id [37].

20 Conclusion

We demonstrate that the Swiss Post Voting System provides complete verifiability and voting secrecy even in the presence of a powerful adversary controlling significant parts of the client and server-side system. Our work includes a detailed analysis of the Federal Chancellery's ordinance to extract precise security goals, specifications of the protocol's cryptographic building blocks, a computational model, and a game-hopping based security analysis. We conclude that the Swiss Post Voting System provides a reasonably strong assurance that the cryptographic protocol fulfills the security objectives of the Federal Chancellery's ordinance.

Acknowledgments

Swiss Post is thankful to all security researchers for their contributions and the opportunity to improve the system's security guarantees. In particular, we want to thank the following experts for their reviews or suggestions reported on our Gitlab repository. We list them here in alphabetical order:

- Aleksander Essex (Western University Canada)
- Pierrick Gaudry, Véronique Cortier, Alexandre Debant (CNRS/LORIA)
- Rolf Haenni, Reto Koenig, Philipp Locher, Eric Dubuis (Bern University of Applied Sciences)
- Thomas Edmund Haines (Australian National University)
- Pascal Junod (modulo p SA)
- Florian Moser (ETH Zürich)
- Olivier Pereira (Université catholique Louvain)
- Carsten Schürmann
- Vanessa Teague (Thinking Cybersecurity)
- Jan Willemson (Cybernetica)

Scytl's previous work, which is the basis of this document, received funding from the European Commission under the auspices of the PROMETHEUS Project, Horizon 2020 Innovation Action (Grant Agreement No. 780701).

References

[1] 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.

[2] 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.

[3] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[4] 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.

[5] 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.

[6] 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.

[7] Josh Benaloh. Verifiable secret-ballot elections. PhD thesis, Yale University, 1987ß.

[8] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 499–516. IEEE Computer Society, 2015.

[9] 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.

[10] 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.

[11] 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.

[12] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: new generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302. IEEE, 2016.

[13] Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. From 5-pass mq-based identification to mq-based signatures. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – Asiacrypt 2016*, volume 10032 of *Lecture Notes in Computer Science*, pages 135–165. Springer-Verlag Berlin Heidelberg, 2016.

[14] 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.

[15] Die Schweizerische Bundeskanzlei (BK). Federal Chancellery Ordinance on Electronic Voting (OEV), Provisional Version of 25 May 2022.

[16] Die Schweizerische Bundeskanzlei (BK). Partial revision of the Ordinance on Political Rights and total revision of the Federal Chancellery Ordinance on Electronic Voting (Redesign of Trials). Explanatory report for its entry into force on 1 July 2022.

[17] 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.

[18] 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.

[19] Eidgenössisches Departement für auswärtige Angelegenheiten EDA. Direct democracy. `https://www.eda.admin.ch/aboutswitzerland/en/home/politik/uebersicht/direkte-demokratie.html/`.

[20] 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.

[21] gfs.bern. Vorsichtige Offenheit im Bereich digitale Partizipation - Schlussbericht, 2020.

[22] Kristian Gjøsteen. The Norwegian Internet Voting Protocol. Cryptology ePrint Archive, Paper 2013/473, 2013. `https://eprint.iacr.org/2013/473`.

[23] Oded Goldreich. *Foundations of cryptography: volume 1, basic tools*. Cambridge university press, 2007.

[24] Rolf Haenni, Reto E. Koenig, Philipp Locher, and Eric Dubuis. CHVote Protocol Specification, Version 3.2. Cryptology ePrint Archive, Report 2017/325, 2020. `https://eprint.iacr.org/2017/325`.

[25] Thomas Haines, Rageev Goré, and Bhavesh Sharma. Did you mix me? formally verifying verifiable mix nets in electronic voting.

[26] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Third Edition*. CRC Press, 2020.

[27] Eike Kiltz, Julian Loss, and Jiaxin Pan. Tightly-secure signatures from five-move identification protocols. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 68–94, Cham, 2017. Springer International Publishing.

[28] Hugo Krawczyk and Pasi Eronen. HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869, Internet Engineering Task Force (IETF), 2010.

[29] 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.

[30] 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.

[31] 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.

[32] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[33] 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.

[34] Phillip Rogaway. Evaluation of some blockcipher modes of operation. In *Technical report - CRYPREC*, 2011.

[35] Ben Smyth. A foundation for secret, verifiable elections. *IACR Cryptology ePrint Archive*, 2018:225, 2018.

[36] Swiss Post. Cryptographic Primitives of the Swiss Post Voting System. Pseudo-Code Specification. Version 1.0.0. `https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives`, 2022.

[37] Swiss Post. Specification of the Swiss Post Voting System. Version 1.0.0. `https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/master/System/System_Specification.pdf`, 2022.

[38] Swiss Post. Swiss Post Voting System. Verifier Specification. Version 1.0.0. `https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/master/System/Verifier_Specification.pdf`, 2022.

[39] Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In *International Conference on Cryptology in Africa*, pages 100–113. Springer, 2010.

[40] 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.

[41] 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.