Scytl – Secure Electronic Voting

STRICTLY CONFIDENTIAL

© Copyright 2018 – SCYTL SECURE ELECTRONIC VOTING, S.A. All rights reserved.

This Document is proprietary to SCYTL SECURE ELECTRONIC VOTING, S.A. (SCYTL) and is protected by the Spanish laws on copyright and by the applicable International Conventions.

The property of Scytl’s cryptographic mechanisms and protocols described in this Document are protected by patent applications.

No part of this Document may be: (i) communicated to the public, by any means including the right of making it available; (ii) distributed including but not limited to sale, rental or lending; (iii) reproduced whether direct or indirectly, temporary or permanently by any means and/or (iv) adapted, modified or otherwise transformed.

Notwithstanding the foregoing, the Document may be printed and/or downloaded.
Table of contents

1 Summary .............................................................................................................. 4

2 Appendix .............................................................................................................. 5
   2.1 EV Solution Intellectual Property Rights Notice (the Notice)........................... 5
       2.1.1 Definitions ................................................................................................. 5
       2.1.2 Copyright notice.......................................................................................... 6

3 Annex: “sVote Voting with Control Components Protocol
- Cryptographic proof of Privacy” ................................................................. 6
1 Summary

In 2013 the Federal Chancellery published a new regulation for the authorization of Internet voting systems (VEleS)\(^1\) that became enforced at the beginning of 2014. This VEleS regulation sets up a framework for the authorization of voting systems according to three different levels, which are directly linked to the amount of electorate that is able to vote through them. From the cantonal point of view, these levels limit the electorate up to 30\%, 50\% or 100\%. To reach the two higher levels, VEleS requires the voting system to pass a certification process based on the security and verifiability properties of such system. In this aim, the certification process includes an examination of the cryptographic protocol, to guarantee it is compliant with the ordinance security requirements of a specific level (abstract model assumptions), by means of verifying a cryptographic and symbolic proof of the voting system to be certified (Req. 5.1.1 of the VEleS Technical Annex\(^2\)).

For the 50\% electorate level, the voting system needs to provide cryptographic and symbolic proofs to demonstrate that the implemented cryptographic protocol provides \textbf{individual verifiability} under the reduced abstract model trust assumptions defined in section 4.1 of the VEleS Technical Annex.

However, for the 100\% electorate level, the system must provide proofs that demonstrate the protocol provides \textbf{complete verifiability} (including individual one) under the complete abstract model defined in section 4.3 of the VEleS Technical Annex. Furthermore, certification for the 100\% electorate level requires not only proving verifiability properties (as in the 50\% electorate level) but also voter privacy ones.

In 2017, Scytl sVote Voting Protocol was certified according to the 50\% electorate level as compliant with the individual verifiability requirements of the VEleS regulation. To this end, cryptographic\(^3\) and symbolic\(^4\) proofs of the individual verifiability properties were provided. However, to achieve the 100\% electoral level, Scytl sVote Voting Protocol has being revised\(^5\) in line to the complete verifiability requirements. Therefore, new cryptographic and symbolic proofs of complete verifiability, and cryptographic and symbolic proofs of voter privacy, have been generated to prove the complete verifiability requirements.

In this document, we are introducing the paper that provides the cryptographic proof of voter privacy of Scytl sVote Voting Protocol, according to the complete abstract model defined in the VEleS ordinance.

---

\(^1\) Swiss Federal Chancellery. Federal Chancellery Ordinance on Electronic Voting (VEleS) of 13 December 2013. (Status as of 1 July 2018)
\(^3\) Scytl Secure Electronic Voting. Swiss Online Voting System Cryptographic proof of Individual Verifiability. 2017
\(^4\) Véronique Cortier, David Galindo, Mathieu Turuan. Analysis of Cast-as-Intended Verifiability and Ballot Privacy Properties for Scytl's Swiss On-line Voting Protocol using ProVerif
2 Appendix

2.1 EV Solution Intellectual Property Rights Notice (the Notice)

Scytl sVote is part of a larger system called EV Solution, developed under the "Framework Agreement" entered into by and between Post CH Ltd (Swiss Post) and Scytl Secure Electronic Voting, S.A. (Scytl) on September 30th, 2015.

Parts of this EV Solution system and other relevant details are defined below.

2.1.1 Definitions

The following terms shall have the meanings specified below:

"EV Solution" means an online voting system consisting of the Scytl Standard Software (also referred to as Scytl sVote or Scytl Online Voting 2.0) in combination with the Swiss Post-Scytl Software, and all the associated middleware provided by Scytl as a bundle with the Scytl Standard Software and the Swiss Post-Scytl Software. Software below middleware (e.g. Linux OS and Windows OS and Oracle software) that are needed to run the EV Solution are not part of the EV Solution.

"Intellectual Property Rights" or "IPRs", for the purposes of this Notice and pursuant to the Framework Agreement, means copyright and patent rights (if any), know-how and trade secrets, performance rights and entitlements to such rights.

"Scytl Online Voting 2.0" is the brand name that was used to identify Scytl Standard Software in the market.

"Scytl Standard Software" means all software developed by Scytl for the EV Solution, whose architecture, specifications and capabilities are described in Scytl sVote documents, excluding Swiss Post-Scytl Software and software developed by Scytl independently to the EV Solution.

"Software" means software code (source code and object code), user interfaces and documentation (preparatory documentation and manuals) and including releases and patches etc.

"Scytl sVote" means the registered trademark proprietary to Scytl, that identifies Scytl Standard Software in the market.

"Swiss Post-Scytl Software" means the software developed for the EV Solution (excluding Scytl Standard Software) pursuant to the Framework Agreement. Swiss Post-Scytl Software comprises of the following:

i. Key Translation Module: A mapping service that translates external IDs to internal IDs for specific entities so that external systems can integrate with sVote.

ii. Swiss Post Integration Tools: A group of applications that allow the integration between Swiss Post's applications and sVote through file conversions.
iii. Swiss Post Voting Portal Frontend: Frontend application that guides the voters throughout all the voting steps enabling them to successfully cast a vote for a particular election.

2.1.2 Copyright notice

2.1.2.1 Scytl Standard Software
All intellectual property rights in the Scytl Standard Software are Scytl’s sole property. Scytl owns and shall retain all rights, title and interest in and to the Scytl Standard Software. Scytl Standard Software is licensed to Swiss Post under the terms and conditions described in the Framework Agreement.

2.1.2.2 Swiss Post-Scytl Software
All intellectual property rights in the Swiss Post-Scytl Software are the joint property of Scytl and Swiss Post (Joint IP).

2.1.2.3 EV Solution
All intellectual property rights in the EV Solution other than Joint IP will be owned by Scytl or by third parties as applicable.

3 Annex: “sVote Voting with Control Components Protocol - Cryptographic proof of Privacy”
sVote Voting with Control Components Protocol - Cryptographic proof of Privacy

R&S

November 23, 2018

Abstract

The purpose of this document is to provide a computational cryptographic analysis of the ballot privacy property of sVote with Control Components voting protocol under the trust assumption defined in The Swiss Confederation technical requirements for electronic vote casting [13].

Keywords: electronic voting protocols, binding election, ballot privacy.
Contents

1 Introduction 4
   1.1 Our contribution ................................. 6
   1.2 Proving the security properties of the protocol .......... 6
   1.3 Results ............................................ 7

2 Threat model and security goals 7
   2.1 Security Assumptions and Threat Model .................. 8
      2.1.1 Assumptions on parties .......................... 8
      2.1.2 Assumptions on communication channels .............. 8
      2.1.3 Additional assumptions on parties regarding voting secrecy 9
   2.2 Trust model in sVote .................................. 10
      2.2.1 Protocol participants in sVote ..................... 10
      2.2.2 Trust assumptions in sVote ....................... 11
   2.3 Correspondence between both security models ............... 11

3 Building blocks 12
   3.1 Encryption schemes .................................. 13
   3.2 Key derivation functions .............................. 13
   3.3 Representation of the voting options .................... 14
   3.4 Pseudo-random functions ................................ 14
   3.5 Signature schemes .................................... 14
   3.6 Non-interactive zero-knowledge proofs of knowledge ..... 15
   3.7 Verifiable mixnet .................................... 18

4 Protocol 18
   4.1 Abstractions and relation to the base protocol .......... 19
      4.1.1 System setup .................................... 19
      4.1.2 Election setup and voting phase .................... 21
      4.1.3 Mapping content to the base protocol ............... 22
   4.2 Codes value spaces .................................... 22
   4.3 Setup phase algorithms ................................ 23
      4.3.1 SetupElKey ..................................... 25
      4.3.2 Setup .......................................... 25
   4.4 Voting phase algorithms ................................ 28
      4.4.1 GetKey .......................................... 28
      4.4.2 CreateVote ...................................... 28
      4.4.3 ProcessVote ...................................... 29
      4.4.4 CreateCC ....................................... 30
      4.4.5 CreateRC ....................................... 31
      4.4.6 AuditCodes ...................................... 31
      4.4.7 Confirm .......................................... 32
      4.4.8 ProcessConfirm ................................... 32
   4.5 Tally phase algorithms ................................ 33
      4.5.1 Cleansing ........................................ 33
      4.5.2 MixDec1 ......................................... 33
      4.5.3 MixDec2 ......................................... 33
   4.6 Verify phase algorithm ................................ 34
      4.6.1 VerifyTally ....................................... 34
1 Introduction

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

Remote electronic voting was first introduced in Switzerland in three cantons: Geneva, Zurich and Neuchâtel [21]. The first binding trials were held in 2004. By 2019, 10 cantons will have offered the electronic voting channel to their electors. However, until recently the participation rate has been restricted to be up to 10% of the eligible voters. In 2011 the Federal Council of Switzerland started a task force for studying the security issues of electronic voting. As a result, the Federal Council published, in 2013, a report with the requirements for extending the use of the electronic voting systems to a larger part of the electorate. This framework [11], which became binding in January 2014, provides requirements of functionality, security, verifiability and testing/certification which allow the electronic voting systems to be extended to 30%, 50% or up to 100% of the electorate. More specifically, while current electronic voting systems may be allowed to be used for up to 30% of the electorate provided that they fulfil a certain set of functional and security requirements, systems to be used for up to 100% of the electorate are required to additionally provide verifiability features. Although the modality of electronic voting (DRE, remote, ...) is not specified in the report, it refers to electronic voting systems where the vote is cast electronically. In this paper, we will talk specifically of remote electronic voting systems.

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

Classically, cast-as-intended and recorded-as-cast verifiability are known as individually verifiable mechanisms, while counted-as-recorded is considered to be a universally verifiable method. The explanation is simple: only the voter knows that she had actually cast a vote, and the intended content. On the other hand, anybody should be able to verify the correct outcome of the election given the votes in the ballot box.
However, the trust model and verifiability requirements defined by the Federal Council differ from these well-known properties. Specifically, the Federal Council defines two types of verifiability in the regulation for e-voting:

**INDIVIDUAL VERIFIABILITY** is defined according to the following trust model:

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

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

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

**COMPLETE VERIFIABILITY** considers the following trust model:

- The server side of the voting platform is not trusted. Instead, there exists a group of so called control components which interact with it and which is trusted as a whole, under the assumption that at least one of them is reliable (each sole control component is not trusted).
- Same assumptions than for individual verifiability apply for voters, print office, client side, and channel between the client side and the server side.
- Given proofs generated by the system that will be verified by auditors, at least one of the auditors and her technical aids (software or hardware tools) are trusted to behave properly.

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

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

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

Besides its requirements, the different electorate extents for which the electronic voting system can be used define the level of certification to be passed. Specifically, systems to be used for more than the 50% of the electors have to provide both security and symbolic proofs which demonstrate that the system fulfills the claimed properties.

1.1 Our contribution

In this paper, we present a protocol which provides ballot privacy, according to the requirements of the Federal Council for systems to be used for up to 100% of the electorate. The protocol has the particularity of only allowing voters to cast one vote through the electronic channel and therefore gives provisions for ensuring that such vote is considered to be cast only in the case that it represents the voter intention, by means of a confirmation phase executed by the voter. After this step, the voter receives a confirmation from the server, which informs of the correct storage of the vote.

The protocol is an evolution of the so-called Norwegian voting protocol that was used in the Norwegian elections in 2011 and 2013. Importantly, it substantially improves the Norwegian scheme by not needing to rely on the strong assumption that two independent server-side entities do not collude to preserve voter privacy. Furthermore, the scheme also represents a great performance improvement of the voting client application compared with the original Puiggali-Guasch scheme, from which the Norwegian scheme was initially derived. Besides the presentation of the protocol, this paper also includes the definitions and the assumptions under which the security properties of the scheme are proven.

Particularly, the security proof of our scheme is focused on proving ballot privacy under the assumptions for complete verifiability given in Federal Council: voting client of an honest voter, one of control components and the print offices are trustworthy, also it is assumed that all voters receive their credentials via an untappable channel.

1.2 Proving the security properties of the protocol

The next methodology is followed in order to prove the security properties of the protocol by means of both cryptographic and symbolic proofs, according to the certification requirements of electronic voting systems to be approved by the Federal Council to be used by up to 100% of the electorate:

1. **Scheme definition:** includes the presentation of the protocol based on
algorithm and workflow descriptions (Section 4) along with the cryptographic primitives and building blocks used (Section 3).

2. Security proof: consists in the demonstration of the fulfillment of the security properties defined in Section 5 by the protocol instantiation from Section 4. Specifically, the property of ballot privacy is analyzed.

3. Symbolic proof: consists of a particular representation of the protocol and the properties to prove, in such a way that logic clauses can be used to check that the protocol fulfills ballot privacy requirements. For the scope of this project, the representation of protocol and properties will be done in a machine-readable language, in order to use specific software (ProVerif) for the verification. The preparation and representation of the symbolic proof is provided in another deliverable.

4. Relation to the implementation design: Section 4.1 presents simplifications and abstractions on the protocol model to easily check the adequacy of the protocol model against the implementation design (closer to the code itself).

1.3 Results
The protocol has been proven to provide ballot privacy (Section 4), edge cases are discussed in Section 6. Given the format of the choice return codes, it is important to take into account some considerations: a vote cannot contain repeated voting options, and the number of voting options inside has to be fixed. In case the voter can perform a variable number of selections, the maximum will be included in the vote, filling those not selected by the voter with blank options (which are all different). This means that the voter will receive choice codes for both the options she selected and the voting options she left blank. Finally, only one vote is allowed per voter, and this has also effect on the security of the cast-as-intended verification mechanism, as it is shown in the analysis. These considerations are all enforced by the protocol implementation.

2 Threat model and security goals
In this section, we derive precise, formal security goals from the informal description of the model for complete verifiability given by the Chancellery. Our interpretation of the threat model is supported by quotes from and references to relevant excerpts of the Chancellery’s requirements [13]. The extraction of precise properties from the legislations informally stated goals is an important step for justifying that the model used throughout the security proofs of sVoted are indeed the same up to the differences in notations.

To precisely see this, we review the Chancellery’s requirements for complete verifiability in Section 2.1. Then, we introduce the model used in sVoted in Section 2.2. Last, in Section 2.3 we provide concrete mappings between the system components and the communication channels in both models.
2.1 Security Assumptions and Threat Model

2.1.1 Assumptions on parties

According to section 4.3 of the Chancellery’s requirements [13], control components, auditors and auditors’ technical aid are referred as additional system components. Similarly, section 4.3 defines additional communication channels referring to system components listed in section 4.1. Based on that and also the fact that complete verifiability in practice uses the same provisions as for individual verifiability we treat the complete abstract model defined in section 4.3 as the extension of the reduced abstract model defined in section 4.1.

Thus, we assume, that the full list of system components consists of components mentioned in section 4.1 plus additional components from section 4.3. Also, we assume that the full list of possible communication channels consists of those defined in 4.1 plus additional communication channels from section 4.3. Using the same logic, we defined trusted elements as the combination of trusted components from section 4.1 (if section 4.3 do not state otherwise) and trusted assumption of section 4.3.

The full list of the system components of the complete abstract model is defined in Table 1. Please note, that the term ‘system component’ is introduced by the legislation and consists of System itself as well as Voters, Print office etc.

<table>
<thead>
<tr>
<th>System components</th>
<th>Trust assumption</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voters</td>
<td>significant proportion of voters are non-trustworthy</td>
<td>4.1</td>
</tr>
<tr>
<td>User platform</td>
<td>untrustworthy for individual</td>
<td>4.1</td>
</tr>
<tr>
<td></td>
<td>and complete verifiability</td>
<td></td>
</tr>
<tr>
<td>Trusted technical aids</td>
<td>trustworthy</td>
<td>4.1</td>
</tr>
<tr>
<td>for voters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System (server-side)</td>
<td>untrustworthy</td>
<td>4.1</td>
</tr>
<tr>
<td>Print office</td>
<td>trustworthy</td>
<td>4.1</td>
</tr>
<tr>
<td>Control Components</td>
<td>trustworthy only as the whole</td>
<td>4.3</td>
</tr>
<tr>
<td>Auditors</td>
<td>at least one is trustworthy</td>
<td>4.3</td>
</tr>
<tr>
<td>Auditor’s technical aid</td>
<td>at least one honest auditor has a trustworthy aid</td>
<td>4.3</td>
</tr>
</tbody>
</table>

Table 1: Assumption on parties of the complete abstract model defined in VEleS [13]

2.1.2 Assumptions on communication channels

Chancellery’s reduced abstract model (section 4.1) defines as trustworthy the technical aids, the system, and the print office and also all channels except User platform ↔ system and System ↔ print office. The complete abstract model (section 4.3) regards the system and only the system as untrustworthy, introducing a set of control components trusted as the whole instead. Also section 4.3 states “Of the additional communications channels, only those between the
auditors and their technical aids may be deemed trustworthy.”

The list of the all possible communication channels is presented in Table 2.

<table>
<thead>
<tr>
<th>Communication channel</th>
<th>Trust assumption</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voters ↔ user platform</td>
<td>trustworthy</td>
<td>4.1</td>
</tr>
<tr>
<td>Voters ↔ trustworthy technical aids</td>
<td>trustworthy</td>
<td>4.1</td>
</tr>
<tr>
<td>Trustworthy technical aids ↔ user platform</td>
<td>trustworthy</td>
<td>4.1</td>
</tr>
<tr>
<td>User platform ↔ system</td>
<td>untrustworthy</td>
<td>4.1</td>
</tr>
<tr>
<td>System ↔ print office</td>
<td>untrustworthy</td>
<td>4.1</td>
</tr>
<tr>
<td>Print office → voter</td>
<td>trustworthy</td>
<td>4.1</td>
</tr>
<tr>
<td>Control component ↔ system</td>
<td>untrustworthy</td>
<td>4.3</td>
</tr>
<tr>
<td>System ↔ auditor’s technical aids</td>
<td>untrustworthy</td>
<td>4.3</td>
</tr>
<tr>
<td>Auditors’ technical aid ↔ auditors</td>
<td>trustworthy</td>
<td>4.3</td>
</tr>
<tr>
<td>Bidirectional channels for communication between control components</td>
<td>untrustworthy</td>
<td>4.3</td>
</tr>
</tbody>
</table>

Table 2: Assumption on communication channels of the complete abstract model defined in VEleS [13]

2.1.3 Additional assumptions on parties regarding voting secrecy

According to [13, Section 4.3], voting secrecy do not account for scenarios where the attacker corrupts user platform: “Under the trust assumptions for complete verifiability of the protocol, the attacker is unable to breach voting secrecy or to obtain early provisional results without changing the voters or their user platforms maliciously.”

Please note that, according to 4.4.8 [13], vote secrecy should be preserved only for trustworthy voters: “It must be ensured that the voting secrecy of trustworthy voters cannot be breached without maliciously changing their user platform through the server-sided manipulation of the application.”

Additionally, the sections 4.3 and supplementary provision 4.4.8 and 4.4.9 imply that for privacy the user platform of trustworthy voters is considered to be trustworthy. Provision 4.4.8 states the following “Voters should therefore be able, using a trustworthy platform, to satisfy themselves that the application is sending their vote in encrypted form with the correct key.”. Provision 4.4.9 is “it must be ensured that the server-sided system cannot find out the content of a vote cast in cooperation with an untrustworthy voter.”

---

1This channel may only be regarded as trustworthy if the information has been sent by Swiss Post (section 4.2.9 page 23)
2In the French version the word corrupting is used: “compte tenu des hypothèses de confiance qui ont été formulées à propos de la vérifiabilité complète du protocole, l’attaquant ne peut ni violer le secret du vote, ni établir des résultats partiels de manière anticipé sans corrompre les électeurs ou leurs plates-formes utilisateurs respectives.”
2.2 Trust model in sVote

2.2.1 Protocol participants in sVote

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

- **Voter**: they participate in the election by choosing their preferred options.

- **Voting Client**: is the device used by the voter to cast their vote given the voting options selected by the voters.

- **Voting Server**: it receives, processes and stores the votes cast by the voters in the ballot box BB.

- **Control Components** are separated in two groups, one is participating in choice return codes the other in mixing:
  
  **Choice Return Codes Control Components (CCR’s)**: they collaborate with the Print office indirectly (via the Voting Server) in the setup phase, and directly with the Voting Server in the voting phase, to compute the so-called long Choice Return Codes.

  **Mixing Control Components (CCM’s)**: they perform the mixing and partial decryption of the ciphertexts in the ballot box.

- **Print Office**: It is responsible for generating, printing and delivering the voting cards to the voters as well as for generating the required election keys.

- **Election Administrators**: they are responsible for generating the election configuration, verifying it, computing the results and publishing them. In the Protocol Specification this entity is divided into Administration Board and Administration Portal based on their ability to perform cryptographic operations, however for the privacy proof we do not distinguish between those two.

- **Global Bulletin board**: is the entity used to store all the information generated during the election to verify the entire process. It stores election configuration, votes, confirmations and keeps track of all the actions performed by each entity. The Bulletin Board is implemented as a distributed system, that includes: election configuration (maintained by Print office), Secure Logger (maintained by CCRs) and Ballot Box (maintained by Voting Server). In this document we refer to Secure Logger as CCR’s logs.

- **Electoral Board**: This entity owns a key pair whose private key is shared among the Board members and is used to partially decrypt the votes in the last Control Component execution.

---

3For generating cryptographic material, Print Office runs a software called Secure Data Manager (SDM). This software is executed in a controlled, offline environment on the canton’s premises. All operations on the SDM are subject to very strict 4-eyes principles and are executed on laptops with special access rights and hardened laptops.
- **Auditors**: they are responsible for verifying the integrity of the procedures run in the counting phase. They are a crucial part of ensuring the verifiability properties as set up by the Chancellery’s requirements, so auditors can be leveraged to detect misbehavior.

- **Verifier**: is the component used to verify the correctness of the entire election process, the integrity of the data processed through different voting system components, and that these processes are accurate and fair.

### 2.2.2 Trust assumptions in sVote

Privacy is proven under assumption required for complete verifiability using the following trust model:

- The **Voting Server** is not trusted. Instead, there exists two groups of so-called control components **CCM’s** and **CCR’s** which interact with **Voting Server** directly and indirectly with **Print office** (via Voting Server). Each group of control components is trusted as a whole, under the assumption that at least one of them is reliable. However, each sole control component is not trusted.

- Credential delivery channel (postal channel between **Print office** and voters) is considered to be trustworthy.

- **Print office** is trusted.

- The **Voting Client** of honest voters is considered to be trusted for privacy, and not trusted for individual and universal verifiability.

- Initial election configuration (number and names of the candidates, number of voters, number of allowed options etc) generated by **Election Administrators** is assumed to be correct as the **Print office** has no means for verifying this information.

- The communication channel between the client side and the server side is not trusted.

- A part of the voters may not be trustworthy.

- **Electoral Board** is treated as set of control component and therefore is trusted as whole, i.e. at least one Electoral Board member is assumed to be trustworthy.

- At least one of the auditors and her technical aids (software or hardware tools) are trusted to behave properly.

### 2.3 Correspondence between both security models

Now we can align the security model defined by the Chancellery (see Section 2.1) and the security model of sVote defined in Section 2.2. A mapping for the different protocol participants is given in Table 3. Similarly, a mapping regarding the communication channels is given in Table 4.
<table>
<thead>
<tr>
<th>sVote’s system component</th>
<th>Chancellery’s system component</th>
<th>Trust assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voters</td>
<td>Voters</td>
<td>significant proportion of voters are non-trustworthy</td>
</tr>
<tr>
<td>Voting Client</td>
<td>User platform</td>
<td>untrustworthy for individual and complete verifiability trustworthy for privacy</td>
</tr>
<tr>
<td>Voting Card</td>
<td>Trusted technical aids for voters</td>
<td>trustworthy</td>
</tr>
<tr>
<td>Voting Server</td>
<td>System (server-side)</td>
<td>untrustworthy</td>
</tr>
<tr>
<td>Print office</td>
<td>Print office</td>
<td>trustworthy</td>
</tr>
<tr>
<td>CCM</td>
<td>Control Components</td>
<td>trustworthy only as the whole</td>
</tr>
<tr>
<td>CCR</td>
<td>Auditors</td>
<td>at least one is trustworthy</td>
</tr>
<tr>
<td>Verifier</td>
<td>Auditor’s technical aid</td>
<td>at least one honest auditor has a trustworthy aid</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>sVote communication channels</th>
<th>Chancellery’s communication channel</th>
<th>Trust assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voters ↔ Voting Client</td>
<td>Voters ↔ user platform</td>
<td>trustworthy</td>
</tr>
<tr>
<td>Voters ↔ Voting Cards</td>
<td>Voters ↔ Trustworthy technical aids ↔ User platform</td>
<td>trustworthy</td>
</tr>
<tr>
<td>no channel exists</td>
<td>Trustworthy technical aids ↔ User platform</td>
<td>trustworthy</td>
</tr>
<tr>
<td>Voting Client ↔ Voting Server</td>
<td>User platform ↔ system</td>
<td>untrustworthy</td>
</tr>
<tr>
<td>Voting Server ↔ Print office</td>
<td>System ↔ print office</td>
<td>untrustworthy</td>
</tr>
<tr>
<td>Print office → Voter</td>
<td>Print office → voter</td>
<td>trustworthy</td>
</tr>
<tr>
<td>CCM ↔ Voting Server</td>
<td>Control component ↔ system</td>
<td>untrustworthy</td>
</tr>
<tr>
<td>CCR ↔ Voting Server</td>
<td>System ↔ auditor’s technical aids</td>
<td>untrustworthy</td>
</tr>
<tr>
<td>Voting Server ↔ Verifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCM ↔ Verifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCR ↔ Verifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verifier ↔ auditor</td>
<td>Auditors’ technical aid ↔ auditors</td>
<td>trustworthy</td>
</tr>
<tr>
<td>no channel exists</td>
<td>Bidirectional channels for communication between control components</td>
<td>untrustworthy</td>
</tr>
</tbody>
</table>

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

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

3 Building blocks

Our abstraction of the sVote voting protocol uses the following building blocks.
3.1 Encryption schemes

**Public key encryption scheme.** Formally, a public key encryption scheme is defined by the algorithms (Gen\(_e\), Enc, Dec): the key generation algorithm Gen\(_e\) receives as input a security parameter 1\(^λ\) and outputs a key pair composed by a public key pk\(_e\) and a private key sk\(_e\), defines a message space \(M_{sp}\), a ciphertext space \(C_{sp}\) and a randomness space \(R_{sp}\) (in case of a probabilistic encryption scheme); the encryption algorithm Enc takes as input a message \(m \in M_{sp}\) and a public key pk\(_e\), and computes a ciphertext \(c \in C_{sp}\). In case the algorithm is probabilistic, it uses random values \(r \in R_{sp}\) for computing such ciphertext; the decryption algorithm Dec receives as input a ciphertext \(c \in C_{sp}\) and a private key sk\(_e\), and outputs a message \(m \in M_{sp}\) or \(⊥\) in case of error.

Our protocol uses the ElGamal encryption scheme \[20\]. The key generation algorithm Gen\(_e\) takes as input a subgroup \(\mathbb{G}\) which has a generator \(g\) of order \(q\) of elements in \(\mathbb{Z}_p^*\), where \(p\) is a safe prime such that \(p = 2q + 1\) and \(q\) is a prime number. It outputs an ElGamal public/secret key pair \((pk_e, sk_e)\), where \(pk_e \in \mathbb{G}\) such that \(pk_e = g^{sk_e} \mod p\) and \(sk_e \in \mathbb{Z}_q^*\). The encryption algorithm Enc receives as input a message \(m \in \mathbb{G}\) and a public key \(pk_e\), chooses a random \(r \in \mathbb{Z}_q\) and computes \(c = (c_1, c_2) = (g^r, pk_e^r \cdot m)\). The decryption algorithm Dec receives \(c\) and the private key \(sk_e\) and outputs \(m = c_2 / c_1^{sk_e}\).

**Symmetric key encryption scheme.** A symmetric key encryption scheme is defined by the algorithms (KGen\(_k\), Enc\(_k\), Dec\(_k\)): KGen\(_k\) receives as input a security parameter 1\(^λ\) and outputs a symmetric key \(k\) from the key space \(\mathcal{K}_{sp}\); Enc\(_k\) takes as input a message \(m \in \{0,1\}^λ\) and a key \(k \in \mathcal{K}_{sp}\), and produces a ciphertext \(c_s \in \{0,1\}^λ\); finally the decryption algorithm Dec\(_k\) takes as input a ciphertext \(c_s \in \{0,1\}^λ\) and a key \(k \in \mathcal{K}_{sp}\), and produces a decrypted message \(m \in \{0,1\}^λ\).

sVote applies a secure KDF function to the input key before feeding that key into the AES encryption scheme in GCM mode \[28\] for authenticated encryption/decryption. Particularly, this encryption mode provides a mechanism for checking the authenticity of encrypted data in the following way: given a message \(m\), a key \(k\) and an authentication information \(a\), the ciphertext and the authentication tag are computed as \(c \leftarrow \text{Enc}^a(m; k)\), \(t_a \leftarrow \text{ghash}(c, a)\), where \text{ghash} denotes a keyed hash function. Then, given the ciphertext \(c\), the authentication information \(a\) and the authentication tag \(t_a\), the authenticated decryption algorithm first checks that \(t_a = \text{ghash}(c, a)\) and if so, it runs Dec\(_k\)(c; k) to obtain the plaintext message \(m\). Otherwise, it returns failure. We use this encryption mode without any authentication information, but still provides a check on the authenticity of encrypted data.

As analysed in \[27\] and \[5\], the privacy and authenticity properties of this cryptosystem rely on the fact that the underlying block cipher cannot be distinguished from a random function in case the secret key is not known, which is the case for the AES encryption algorithm.

3.2 Key derivation functions

The protocol uses as a Key Derivation Function \(δ\) the function defined in ISO-18033-2 and in PKCS#1v2.2 with the name MGF1.
The protocol additionally uses a password-based key derivation function defined by the algorithm PBKDF2 which, on input a password string pwd, a security parameter $1^\lambda$ and a salt salt, derives a cryptographic key $K$. Specifically, we use the PBKDF2 algorithm specified in [24], which additionally receives a number of iterations $c$ and the output length $dkLen$. This algorithm derives the cryptographic key $K$ using a pseudo-random function based on the hash algorithm SHA-256 iterated $c$ times over the concatenation of the password and the salt. The security of this primitive relies on the one-way and collision-resistance properties of the underlying hash function.

### 3.3 Representation of the voting options

The voting options $\{v_1, \ldots, v_n\}$ are chosen as small bit-length primes belonging to the subgroup $G$ defined for the ElGamal encryption scheme. We use the operations of product $\prod$ and factorization $\text{fact}$ for encoding/decoding the selected voting options: a vote $\nu$ is the product of voting options chosen by the voter prior to the encryption. After the votes are decrypted, the individual voting options are recovered by factorizing the resulting value. Therefore, it has to be ensured that the product of $t$ of such primes, where $t$ is the number of selections a voter can make, is smaller than $p$. Only pre-configured voting options which are represented as primes are considered in the protocol description and in the security analysis.

### 3.4 Pseudo-random functions

A function family is a map $F : T \times D \rightarrow R$, where $T$ is the set of keys, $D$ is the domain and $R$ is the range. A pseudorandom function family (PRF) is a family of efficiently computable functions with the following property: a random instance of the family is computationally indistinguishable from a random function, as long as the key remains secret. The function $f_K(x) = y$ denotes a function $f$ from a family $F$, parameterized by a key $K$.

A keyed pseudo-random functions is used in the protocol: we denote by $f_k()$ an HMAC function composed by a SHA-256 hash function, parameterized by the symmetric key $k$. As detailed in [4], HMAC is a PRF whose resistance against collision is the one of the underlying hash scheme. Up to date, the collision probability of the SHA-256 hash function is considered to be negligible.

### 3.5 Signature schemes

A signature scheme is defined by three probabilistic algorithms $\text{Gen}$, $\text{Sign}$, $\text{Verify}$, that stand for key generation, signature generation and signature verification. $\text{Gen}$ receives as input a security parameter $1^\lambda$, outputs a signing key pair $(pk_s, sk_s)$ and defines a message space $M_{sp}$ and a signature space $S_{sp}$: $\text{Sign}$ receives a message $m \in M_{sp}$ and the signing private key $sk_s$, and outputs a signature $\psi \in S_{sp}$; $\text{Verify}$ receives the signing public key $pk_s$, a message $m \in M_{sp}$ and a signature $\psi \in S_{sp}$, and outputs 1 if the verification succeeds, 0 otherwise.

Our scheme uses the RSA Probabilistic Signature Scheme (PSS) ([6,8,30]), which is an RSA system with hash variant that uses a random padding: $\text{Gen}$ receives two primes $p, q$ of similar bit-length ($\lambda/2$) (which define a ring $\mathbb{Z}/n\mathbb{Z}$) and computes the public key $pk_s = (n, e)$, where $n = pq$ and $e$ is coprime with
The private key $sk_a$ takes the value of $d$, where $ed = 1 \mod \phi(n)$. Sign takes as input a message $m$, which is not restricted to a specific space, and the private key $sk_a$, and outputs $\psi = (\text{ME}(m))^d \mod n$, where ME denotes a transformation with random padding over $H_s(m)$ and $H_s$ denotes a hash function which maps strings to elements in $Z_n$. Verify takes as input the public key $pk_s$, the message $m$ and the signature $\psi$, and checks that $\text{ME}(m) = \psi^e \mod n$. It outputs 1 if the verification is successful, 0 otherwise.

This signature scheme is preferred instead of schemes with deterministic paddings such as RSA-FDH [7], given that it provides a tighter security proof [8,14] for signature unforgeability in the Random Oracle Model (ROM).

3.6 Non-interactive zero-knowledge proofs of knowledge

The sVote protocol uses the Fiat-Shamir [19] transformation to turn interactive zero-knowledge protocols, such as $\Sigma$-proofs, into non-interactive proofs, by using a hash function to compute the random challenge. The security of the resulting non-interactive zero-knowledge proof of knowledge (NIZKPK) is based on the assumption made in the ROM that a hash function behaves as a random oracle. Therefore the challenge has a resulting distribution similar to the original and the non-interactive version of the ZKPK maintains its properties [7].

A NIZKPK scheme is composed by the algorithms ($\text{GenCRS}$, $\text{NIZKProve}$, $\text{NIZKVerify}$, $\text{NIZKSImulate}$): The common reference string generation algorithm $\text{GenCRS}$ generates the parameters of the NIZKPK scheme. It receives as input a security parameter $1^\lambda$ and, in some cases, a mathematical group $G$, and it outputs a common reference string $\text{crs}$. $\text{NIZKProve}$ is the proof generation algorithm. It receives as input a common reference string $\text{crs}$, a statement $x$ and a witness $w$, and outputs a proof $\pi$. $\text{NIZKVerify}$ is the verification algorithm. It receives as input the common reference string $\text{crs}$, the statement $x$ and the proof $\pi$, and outputs 1 if the verification is successful, 0 otherwise. $\text{NIZKSImulate}$ is a proof simulation algorithm. It receives as input a (false) statement $x^*$ and outputs a simulated proof $\pi^*$.

Our implementation for NIZKPKs uses the Maurer framework [26] for a generalized implementation. This framework defines a common procedure for constructing interactive proofs for statements presenting an homomorphism $\phi$ such that $\phi(a; x) \rightarrow a^x$. We provide concrete example. Let’s say that a Prover wants to prove that it knows $x$ such that $\phi(a; x) \rightarrow a^x$:

- Prover computes witness $t = \phi(a; s) = a^s$, where $s$ is selected at random from the same value space than $x$, and sends it to the verifier.
- Verifier provides a random challenge $h$.
- Prover computes $z = s + x \cdot h$ and provides it to the verifier.
- Verifier checks that $(a^x)^h \cdot t = a^z$.

This procedure can be turned into non-interactive by computing the challenge as the hash function (Fiat-Shamir heuristic [19]) over some of the elements that participate in the proof generation, such as the input statement $\phi(a; x)$, the original value $a$, and some auxiliary string. Moreover, including the initial witness into the hash for computing the challenge $h$ the resulting proof is shorter.
The procedure then is as follows:

- Prover computes witness $t = \phi(a; s) = a^s$, where $s$ is selected at random from the same value space than $x$.
- Prover computes the challenge $h$ as $h \leftarrow H(a, \phi(a; x), t, aux)$.
- Prover computes $z = s + x \cdot h$ and provides it to the verifier, together with $h$.
- Verifier computes $h' = H(a, a^x, (a^x)^{-h} \cdot a^z, aux)$ and checks whether $h' = h$.

Following this method for the generation of NIZKPKs, we proceed to describe the four types of NIZKPKs used in the protocol:

**Equality of discrete logarithms.** This is a generalization of the Chaum-Pedersen proof system \cite{12} which we denote as EqDL:

- **ProveEq**$(\langle a_1, a_2, \ldots, a_n, a_1^*, a_2^*, \ldots, a_n^* \rangle, s)$.\footnote{Our NIZKPK schemes particularly do not use a common reference string, and therefore GenCRS is not executed.} takes at random a value $s$ from $\mathbb{Z}_q$, computes $a_1^*, a_2^*, \ldots, a_n^*$, $h = H(a_1, a_2, \ldots, a_n, a_1^*, a_2^*, \ldots, a_n^*, a_1^*, a_2^*, \ldots, a_n^*)$ and $z = s + x \cdot h$, being $H$ a hash function which maps strings to elements in $\mathbb{Z}_q$. The output proof $\pi_{\text{eqdl}}$ is $(h, z)$.

- **VerifyEq**$(\langle a_1, a_2, \ldots, a_n, a_1^*, a_2^*, \ldots, a_n^* \rangle, \pi_{\text{eqdl}})$ computes $a_1'^* = a_1^* \cdot (a_1^*)^{-h}$, $a_2'^* = a_2^* \cdot (a_2^*)^{-h}$, \ldots, $a_n'^* = a_n^* \cdot (a_n^*)^{-h}$, and checks that $h = H(a_1, a_2, \ldots, a_n, a_1^*, a_2^*, \ldots, a_n^*, a_1'^*, a_2'^*, \ldots, a_n'^*)$. If the validation is successful, the algorithm outputs 1. Otherwise it outputs 0.

- **SimEq**$(a_1, a_2, \ldots, a_n, a_1^*, a_2^*, \ldots, a_n^*)$ takes $z^*$ and $h^*$ at random from $\mathbb{Z}_q$ and forms the proof $\pi^*$. In this kind of proof, a programmed random oracle has to be used for simulation such that when the adversary asks for the value $H(a_1, a_2, \ldots, a_n, a_1^*, a_2^*, \ldots, a_n^*, a_1'^*, a_2'^*, \ldots, a_n'^*)$ the oracle returns the value $h^*$.

**Knowledge of encryption exponent.** Based on the Schnorr identification protocol \cite{12}, it is used for proving knowledge of the encryption exponent of the ElGamal ciphertext $c$. We denote it as ExpP, and its construction is similar to the particular case of the NIZK proof of equality of discrete logarithms where $n = 1$. It additionally makes use of some auxiliary information aux that is known both to prover and verifier. Therefore:

- **ProveExp**$(\langle g, c_1, c_2 \rangle, r)$ takes at random a value $s$ from $\mathbb{Z}_q$, computes $g^s$, $h = H(\text{aux}, g, c_1, c_2, g^s)$ and $z = s + r \cdot h$. The output is $\pi_{\text{sch}} = (h, z)$.

- **VerifyExp**$(\langle g, c_1, c_2 \rangle, \pi_{\text{sch}})$ computes $g'^* = g^z \cdot (c_1^{-h})$, and checks that $h = H(\text{aux}, g, c_1, c_2, g'^*)$. If the validation is successful, the algorithm outputs 1. Otherwise it outputs 0.
Proof of equality of encryptions. The proof system \textbf{EqEnc} consists of three algorithms (\textbf{ProveEqEnc}, \textbf{VerifyEqEnc}, \textbf{SimEqEnc}) and it is used to prove that \(c\) and \(\tilde{c}\) are \textit{ElGamal} encryptions of the same plaintext under public keys \((g, h_1)\) and \((g, h_2)\) respectively. They are defined as follows:

- \textbf{ProveEqEnc}(g, h_1, h_2, c = (c_1, c_2), \tilde{c} = (\tilde{c}_1, \tilde{c}_2), r_1, r_2)\) takes at random values \(s_1, s_2\) from \(\mathbb{Z}_q\), computes \(g^{s_1}, g^{s_2}, h_1^{s_1}(\frac{1}{h_2} s_2)\) and \(z_1 = s_1 + r_1 \cdot h, z_2 = s_2 + r_2 \cdot h\), being \(H\) a hash function which maps strings to elements in \(\mathbb{Z}_q\). The output proof \(\pi_{eqenc}\) is \((h, z_1, z_2)\).

- \textbf{VerifyEqEnc}(g, h_1, h_2, c = (c_1, c_2), \tilde{c} = (\tilde{c}_1, \tilde{c}_2), \pi_{eqenc})\) computes \(g^{s_1} \cdot c_1^{-h}, g^{s_2} = g^{s_2} \cdot (\tilde{c}_1)^{-h}, h_1^{s_1}(\frac{1}{h_2} s_2) = h_1^{s_1} \cdot (\frac{1}{h_2} s_2) = 1\), and checks that \(h = H(c_1, \tilde{c}_1, z_1, g^{s_1}, g^{s_2}, h_1^{s_1}(\frac{1}{h_2} s_2))\). If the validation is successful, the algorithm outputs 1. Otherwise it outputs 0.

- \textbf{SimEqEnc}(g, h_1, h_2, c = (c_1, c_2), \tilde{c} = (\tilde{c}_1, \tilde{c}_2))\) takes \(z_1^*, z_2^*\) and \(h^*\) at random from \(\mathbb{Z}_q\) and forms the proof \(\pi^*\). In this kind of proof, a programmed random oracle has to be used for simulation such that when the adversary asks for the value \(H(c_1, \tilde{c}_1, z_1^*, g^{s_1}, g^{s_2}, h_1^{s_1}(\frac{1}{h_2} s_2))\) the oracle returns the value \(h^*\).

Correct decryption. Proofs of correct decryption are based on the Chaum-Pedersen protocol [12]. However, we use a different notation than in \textbf{EqDL} for simplicity in the protocol description. We denote them as \textbf{DecP} and describe the following algorithms:

- \textbf{ProveDec}(c, m, sk_a)\) receives a ciphertext \(c = (c_1, c_2)\) and a witness \(sk_a\), where \(c_1 = g^z\) and \(c_2 = pk_a \cdot m\), being \(pk_a = g^{sk_a}\). It takes at random \(s\) from \(\mathbb{Z}_q\), computes \((g^z)^s, g^z, h = H(c, m, (g^z)^s, g^z)\) and \(z = s + sk_a \cdot h\). The proof is \(\pi_{dec} = (h, z)\).

- \textbf{VerifyDec}(c, m, \pi_{dec})\) computes \((g^z)^s = (c_1)^s \cdot (c_2/m)^{-h} \cdot g^{s^*} = g^z \cdot pk_a^{-h}\) and checks that \(h = H(c, m, (g^z)^s, g^z)\). If the validation is successful, the algorithm outputs 1. Otherwise it outputs 0.

- \textbf{SimDec}(c, m^*)\) takes at random \(z^*\) and \(h^*\) from \(\mathbb{Z}_q\) and forms the proof \(\pi^*\). As in the previous proof, a programmed random oracle has to be used for simulation such that when the adversary asks for the value \(H(c, m^*, (g^z)^s, g^z)\) the oracle returns the value \(h^*\).

The properties of NIZKPKs are \textit{completeness}, \textit{soundness} and \textit{zero-knowledge} [17, 31]. Informally, completeness tells us that given a proof generated by
an honest prover, the verifier will always succeed on the verification. Soundness means that in case a dishonest prover generates a proof over an incorrect statement, the verification will fail with overwhelming probability. Finally, the property of zero-knowledge implies that the outputs of the proving and the simulation algorithms are indistinguishable.

3.7 Verifiable mixnet

A verifiable mixnet is composed by two algorithms: the algorithm Mix receives a set of ciphertexts $C = \{c_1, \ldots, c_\ell\}$ as input, and outputs a set of ciphertexts $C' = \{c'_1, \ldots, c'_\ell\}$ and a proof $\pi_{\text{mix}}$ of correct mixing. These ciphertexts correspond to the input values, randomly permuted and re-encrypted or partially decrypted, depending on the type of mixnet. The algorithm MixVerify receives as input two sets of ciphertexts $C$ and $C'$ and the proof of correct mixing $\pi_{\text{mix}}$, and outputs 1 or 0 depending on the result of the verification.

In our protocol we use the verifiable re-encryption mixnet proposed by Stephanie Bayer and Jens Groth [3]. This mixnet has been proven by its authors to be sound, meaning that MixVerify will output 0 given an incorrect execution of Mix with overwhelming probability, and zero-knowledge both in the standard model and in the random oracle model in case of using the Fiat-Shamir heuristic for making the proofs non-interactive.

4 Protocol

A voting scheme relative to a list of voters $ID$ and a counting function $\rho$ consists of the tuple $\mathcal{V} = (\text{Setup}, \text{Vote}, \text{Tally}, \text{Verify})$. In [18] a voting scheme, called sVote, is defined relative to the multiset function [10], that is, the function that outputs all casted votes in random order.

**Completeness.** A voting scheme as defined above is correct if, when the four phases are run the result $r$ output by the Tally algorithm is equal to the evaluation of the counting function $\rho$ over the voting options corresponding to the votes cast and confirmed by the voters.

**Cast-as-intended and recorded-as-cast** To provide these properties the scheme is based on return codes. In this approach, the system provides to the voter a proof of content of the vote she has cast, consisting on a set of Choice Return Codes which she can check in her voting card. In case the verification is successful, the voter proceeds to confirm her vote. Note that this mechanism consist in two rounds between the server, the voter and the voting client at the moment of voting. Also, we do not cover the provision of voting cards to the voters; it is assumed that this will be delivered by some external means, e.g. postal channel.

**Counted-as-recorded.** The scheme uses verifiable mixnets [3] to ensure privacy. These mixnets are enhanced with zero-knowledge proofs of correct shuffling and correct decryption, which allows to check robustness even in the presence of an almost corrupted server.
Protocol overview. sVote uses double ElGamal encryptions to cast a vote. The first ciphertext is used in the tally phase, and the second is used in the voting phase to obtain the choice return codes corresponding to the voting options. For verifiability, two NIZKs are used to ensure consistency across both encryptions and for privacy reasons, a third NIZK is used to sign the first ciphertext, making it non-malleable. The voting card id is used as auxiliary information to generate these proofs. Additionally, a key derivation function $\delta$ and a symmetric encryption scheme is used to retrieve the choice return codes. Last, during tally a chain of verifiable mixnets randomly permute and jointly decrypt the encrypted votes.

Section organization. In Section 4.1 we discuss simplifications and which parts from [18] are omitted because are not relevant in the security analysis provided in Section 5. In Section 4.2 we introduce the value spaces of the different codes, and the remaining of the sections are devoted to describe the algorithms used in each phase.

4.1 Abstractions and relation to the base protocol

In this section we point out the abstractions and simplifications that have been done in the protocol description presented in Section 4 in relation to the implementation of the system used in the Swiss Online Voting System.

4.1.1 System setup

We assume that global configuration, independent of specific election events, is set in advance and it is ready to use.

CA hierarchy. Constitution of a platform root CA, and generation of credentials for the different system contexts and tenants that wish to run an election is omitted. For more details, see [18, Sections 3.1, 3.2, 3.3]

Bulletin board, ballot box and logs. In sVote the ballot box is maintained by the voting server. In addition to this, both groups of control components log all the transcript they see during setup, voting and tally phases. The ballot box and the logs are later handed to the auditors. We model this situation as a distributed bulletin board $B$, comprised of the ballot box denoted with $bb$, the transcript of the $j$-th choice return control component $CCR_j$, denoted with $bb^{CCR}_j$, and the transcript of the $j$-th mixing control component $CCM_j$ denoted with $bb^{CCM}_j$. Throughout this document, we sometimes refer to $bb^{CCR}_j$ and $bb^{CCM}_j$ as the secure log of the control components. For details on how transcript is logged securely see [18, Section 3.5]. For example the secure log of $CCR_j$ includes the voter’s ballot submitted by the voting device, and the part of the transcript corresponding to the generation of the choice return codes and the vote cast codes.

Number of control components. For simplicity we consider only two $CCR$s and two $CCM$s, whereas in [18] four components per group are specified.
Without loss of generality we can use only two CCRs in our model because, the role of the CCRs is symmetric. Indeed, in the security analysis the only assumption we made on the number of the control components is that at least one member of each group is trusted.

As for CCMs, even though those components are executed in a sequence, we also claim that our reduction does not affect proof structure due to the mandatory audit performed before the last CCM is executed and the fact that the last key is distributed among members of Electoral Board.

Consider a case of \( N \) CCMs where only one of them is honest. This consideration can be done without loss of generality as any other scenario can be mapped to this extreme case. According to [18], the mandatory verification would be performed after \( N - 1 \) CCMs shuffled and partially decrypted votes. Verification would fail if at least one of the following is true: a) cleansing procedure is not correct b) one of mixing proofs is invalid or c) one of the decryption proofs is invalid.

If the verification holds, Electoral Board members would submit their private shares so the last CCM would reconstruct its key. Taking into account that Electoral Board can be viewed as a set of ‘human control components’\(^5\) at least one of the members is honest and refuses to submit the decryption key share if validation fails. Thus, the last CCM would be able to reconstruct the last CCM’s decryption key and perform the decryption process if and only if verification holds.

Table 5 shows all possible corruption scenarios in case of \( N \) CCMs and Electoral Board. Please bear in mind, that the table was constructed assuming that there is only one honest CCM in the whole chain and Honest Electoral Board members submit their private keyshares if and only if verification holds.

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
<th>Verification holds</th>
<th>Honest EB members submit their keyshares</th>
<th>Last CCM knows its key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>Last CCM is corrupted, but can’t reconstruct its key.</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Case 2</td>
<td>Last CCM is corrupted and can reconstruct its key.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Case 3</td>
<td>Last CCM is honest, but can’t reconstruct its key.</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Case 4</td>
<td>Last CCM is honest and knows its key.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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

Due to simplicity reasons, in proofs we abstract from Electoral Board members and assume that the second CCM’s is no different from the first one. Please also note, that in the cryptographic games, the challenger executes both a CCM and mandatory verification in a single function and outputs the result only if verification holds. This is done without loss of generality because, all possible

\(^5\)According to section 4.4.10 of [13], it is also permitted to implement a group of control components so that they take the form of people.
outcomes in such case are consistent with outcomes in case of $N$ CCMs and Electoral Board i.e.:

1. The Challenger runs the first CCM
   This case covers Case 2 as in this scenario the last CCM is corrupted and can reconstruct its key.
   Also Case 1 can be viewed as a strong version of Case 2, where Attacker controls the last node but is not able to reconstruct the key due to at least one missing share.

2. The Challenger runs the second CCM
   This case covers both Case 3 and Case 4 as in those scenarios only the last CCM is honest.

4.1.2 Election setup and voting phase

Offline mixing control component. We have abstracted away the generation and recovery of the election key private key share $E_{(2)}^{sk}$ corresponding to the last CCM. In the actual implementation, during setup this private key is secret-shared using Shamir secret sharing scheme among the electoral board members, and during tally, it is reconstructed accordingly. However, this details are irrelevant in the security analysis: it only matters that the adversary gets to know the key $E_{(2)}^{sk}$ in case the last CCM is compromised. See [18, Sections 4.5, 6.1.2, Annex 9.1.9, 9.1.10] for more details.

Generation of system parameters. Parameters not relevant to the security analysis are assumed to be generated ahead of time. These include election rules, ballot generation, initialization of the ballot box, and necessary public parameters needed to encrypt the voting options.

Voting options. We denote the set of valid votes by $\Omega$, which is composed by any combination of voting options $\{v_1, \ldots, v_\psi\}$ which is valid according to the election rules. Both the set of valid votes and the counting function $\rho : (\Omega \cup \{\bot\})^\ast \rightarrow \mathbb{R}$ are assumed to be defined in advance. As already mentioned, the set of possible results $R$ is given by the multiset function $\rho$, which provides the cleartext votes cast by the voters in a random order [10]. Additionally, if write-in values are permitted in the election, cast-as-intended for the write-in content cannot be provided, since it is not possible to provide the voter with a mapping of all possible write-in values to a choice return code. Therefore, we will not cover write-in values in our security and formal analysis. Last, vote correctness is also abstracted away since with the presence of auditors voting options not corresponding to the pre-defined prime numbers will be detected at the end of the election. Clearly, tampering with the range of voting options does not affect voter’s privacy.

Authentication. We have not covered details on how eligible voters authenticate in the system. This might happen via a dedicated protocol, or via a third party. In any case, the focus in this document is to show that privacy is maintained regardless of authentication procedure, so how the user authenticates is considered out of the scope. This is consistent with the current state-of-the-art
in computational security proofs for e-voting systems, where eligibility verifiability is very rarely analyzed.

In the actual implementation, users authenticate in the system using a challenge-response mechanism. The goal of the authentication procedure is to ensure that only a voter who proved that he opened an encrypted key store gets a valid authentication token. Namely, a voter sends a request that includes his Credentials ID to the server. The server sends back the corresponding encrypted Verification Card keystore and a challenge. Voter replies by sending a client message, which includes the server’s challenge signed with the key retrieved from the encrypted keystore. If the reply to the challenge is valid, Server generates an Authentication Token containing the Voter Information, a timestamp etc. and sends it to the voter. During the voting phase, the voter is going to include this token to every request it sends to the server. See [18, Sections 5.1, 5.2] for more details.

Details about authentication layer have been deliberately omitted in the proofs for the sake of clarity, and given the fact that they are not relevant for proving cast-as-intended verifiability, universal verifiability or privacy. We emphasize that our model is independent of the way encrypted Credential Data Keystore is delivered to the voters. Our only requirements are: Verification Card Keystore is generated and encrypted the way it is described in our model and Voting Cards are delivered to the voters via a trusted channel (i.e. post office).

Please notice, that a Verification Card Keystore can only be opened by the person in possession of the voting card: the keystore is encrypted with the key that is derived from the Start Voting Key (SVK_{id}) printed in the voting card.

In the protocol model we assume, that all encrypted Verification Card Keystore are public. Moreover, the Attacker can open a keystore if he controls the Voting Client or corrupts a voter and access his voting card. In case, when a voter is honest and Voting Client is not leaking any information to the Attacker, it is assumed that voter’s Start Voting Key (SVK_{id}) that is printed in the voting card is private.

4.1.3 Mapping content to the base protocol

In Table 6 the different algorithms of the protocol described in this document are mapped against their equivalents in the specifications [18].

In Table 7 all keys that are used in the protocol model are described. Please note, that in specifications [18] more keys are used, the reason for omitting them have been discussed in the preceeding sections.

4.2 Codes value spaces

As mentioned above, the protocol uses different codes to let a voter verify and confirm her vote. These codes should be large enough to argue security, and small enough to make the system usable. This apparent contradiction is solved by storing symmetric encryptions of the short codes using the long codes as
secret keys. In turn, the long codes are computed interactively during the voting phase using as input the start voting key and the ballot casting keys printed in the voter’s voting card.

START VOTING KEYS. These codes are 20-character strings encoded in base 32. The space $A_{svk}$ consists of $32^{20}$ values.

BALLOT CASTING KEYS. These codes are 8-digit numbers. The space $A_{bck}$ consists of $10^8$ values.

LONG CHOICE AND VOTE CAST RETURN CODES. These codes are sometimes seen as ElGamal plaintexts. The space $A_{lc}$ is the group $\mathbb{G}$, so there are $\text{Ord}(\mathbb{G}) = q$ different possible values.

SHORT CHOICE RETURN CODES. These codes are 4-digit numbers. The space $A_{cc}$ consists of $10^4$ values.

SHORT VOTE CAST RETURN CODES. These codes are 8-digit numbers. The space $A_{vcc}$ consists of $10^8$ values.

4.3 Setup phase algorithms

The setup phase consists on the following algorithms.
<table>
<thead>
<tr>
<th>Notation</th>
<th>Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>( pk_{SDM} )</td>
<td>Secure Data Manager public encryption key that is used by the Print Office to encrypt the prime numbers and the ballot casting key before sending them to CCRs during the Setup phase.</td>
<td>Public</td>
</tr>
<tr>
<td>( sk_{SDM} )</td>
<td>Secure Data Manager private encryption key that is used by the Print Office to decrypt the reply from CCRs during the Setup phase.</td>
<td>Private. Known to Print Office only.</td>
</tr>
<tr>
<td>( El_{pk} )</td>
<td>Election public key</td>
<td>Public</td>
</tr>
<tr>
<td>( El_{sk} )</td>
<td>Election secret key</td>
<td>Private. Known to the Print Office only.</td>
</tr>
<tr>
<td>( El_{pk}^{(j)} )</td>
<td>Election public key share of CCM(_j).</td>
<td>Public</td>
</tr>
<tr>
<td>( El_{sk}^{(j)} )</td>
<td>Election secret key share of CCM(_j).</td>
<td>Private. Known only to CCM(_j).</td>
</tr>
<tr>
<td>( C_{sk} )</td>
<td>Codes Secret Key</td>
<td>Generated by Print Office. Known to the Voting Server.</td>
</tr>
<tr>
<td>( VCCs_{pk} )</td>
<td>Vote Cast Return Code Signer public key</td>
<td>Public</td>
</tr>
<tr>
<td>( VCCs_{sk} )</td>
<td>Vote Cast Return Code Signer private key</td>
<td></td>
</tr>
<tr>
<td>( K_{1d}^{(j)} )</td>
<td>CCR(_j)’s Voter choice return code generation public key</td>
<td>Public</td>
</tr>
<tr>
<td>( k_{1d}^{(j)} )</td>
<td>CCR(_j)’s Voter choice return code generation private key</td>
<td>Private. Known to CCR(_j) only.</td>
</tr>
<tr>
<td>( K_{cid}^{(j)} )</td>
<td>CCR(_j)’s Voter vote cast code generation public key</td>
<td>Public</td>
</tr>
<tr>
<td>( k_{cid}^{(j)} )</td>
<td>CCR(_j)’s Voter vote cast code generation private key</td>
<td>Private. Known to CCR(_j) only.</td>
</tr>
<tr>
<td>( K_{c1}^{(j)} )</td>
<td>CCR(_j)’s Choice return code generation public key</td>
<td>Public</td>
</tr>
<tr>
<td>( k_{c1}^{(j)} )</td>
<td>CCR(_j)’s Choice return code generation private key</td>
<td>Private. Known to the Voting Server.</td>
</tr>
<tr>
<td>( sk_{ccid}^{(i)} )</td>
<td>( i )th symmetric key to encrypt ( i )th choice return code for voter ( id ).</td>
<td>Private. Known to the Voting Server.</td>
</tr>
<tr>
<td>( skvcc^{(i)} )</td>
<td>Symmetric key to encrypt vote cast code for voter ( id ).</td>
<td>Private. Known to the Voting Server.</td>
</tr>
<tr>
<td>( pk_{CCR_j}^{(i)} )</td>
<td>( i )th component of the CCR(_j) Choice Return Codes Encryption public key</td>
<td>Public</td>
</tr>
<tr>
<td>( sk_{CCR_j}^{(i)} )</td>
<td>( i )th component of the CCR(_j) Choice Return Codes Encryption private key</td>
<td>Private. Known to CCR(_j) only.</td>
</tr>
<tr>
<td>( BCK^{\text{id}} )</td>
<td>Voter’s Ballot Casting Key</td>
<td>Private. Known to the voter only.</td>
</tr>
<tr>
<td>( SVK_{\text{id}} )</td>
<td>Voter’s Start Voting Key</td>
<td>Private. Known to the voter only.</td>
</tr>
<tr>
<td>( VCks_{\text{id}} )</td>
<td>Keystore key to store ( SVK_{\text{id}} )</td>
<td>Private. Known to the Voting Device only.</td>
</tr>
</tbody>
</table>

Table 7: Keys that are used in the model

*The election secret key share of the offline CCM is further secret-shared with shamir among the electoral board members. See discussion on offline CCM in Section [1.1.2](#).*
4.3.1 SetupElKey(1^\lambda)

It is an interactive algorithm run by the mixing control components CCM_1, CCM_2 and the print office. They proceed as follows.

1. SetupElKey.\text{CCM}_j(1^\lambda) is run by CCM_j for j = 1, 2. On input a security parameter compute and the CCM_j mixing key pair (EL_{\text{pk}}^{(j)}, EL_{\text{sk}}^{(j)}) \leftarrow \text{Gen}_e(1^\lambda).
   It keeps EL_{\text{sk}}^{(j)} secret and sends EL_{\text{pk}}^{(j)} to the print office.

2. SetupElKey.\text{SDM}(EL_{\text{pk}}^{(1)}, EL_{\text{pk}}^{(2)}) is run by the print office. It takes the the election public key shares EL_{\text{pk}}^{(1)}, EL_{\text{pk}}^{(2)} created by CCM_1, CCM_2 respectively, computes the Election Public Key as EL_{\text{pk}} := EL_{\text{pk}}^{(1)} \cdot EL_{\text{pk}}^{(2)}.

4.3.2 Setup(1^\lambda, ID)

It is an interactive algorithm run by the print office and the return codes control components CCR_1, CCR_2 and later on verified by auditors. It receives as input a security parameter 1^\lambda and list of voter pseudonyms ID. It proceeds as follows.

1. Setup.\text{SDM}(ID) it is executed in the print office.
   It starts generating the following:
   - the Secure Data Manager encryption key pair (\text{pk}_{\text{SDM}}, \text{sk}_{\text{SDM}}) \leftarrow \text{Gen}_e(1^\lambda)
   - a Codes secret key \text{Csk} \leftarrow T, where T denotes the set of possible keys of the PRF function f
   - a Vote Cast Return Code Signer key pair (\text{VCC}_{\text{pk}}, \text{VCC}_{\text{sk}}) \leftarrow \text{Gen}_s(1^\lambda).

   For each voter id \in ID, where ID denotes a list of voter pseudonyms, the print office generates cryptographic keys, identifiers and short codes:
   - a random Verification Card ID VC_{id} \leftarrow \mathcal{A}_{\text{vc}}
   - a Start Voting Key SVK_{id} \leftarrow \mathcal{A}_{\text{svk}}
   - a keystore symmetric encryption key KSkey_{id} \leftarrow \text{PBKDF2}(SVK_{id}, \text{KEYseed})
   - the Verification Card key pair: (k_{id}, k_{id}) \leftarrow \text{Gen}_e(1^\lambda)
   - the encryption of the Verification Card private key with the keystore encryption key: VC_{\text{sk}}{id} \leftarrow \text{Enc}_e(k_{id}, \text{KSkey}_{id})
   - a random Ballot Casting Key BCK_{id} \leftarrow \mathcal{A}_{\text{bck}}
   - a short Choice Return Codes CC_{id} \leftarrow \mathcal{A}_{\text{cc}} at random for voter id, \forall i = 1, \ldots, n
   - a short Vote Cast Return Code VCC_{id} \leftarrow \mathcal{A}_{\text{vcc}} at random for voter id
   - Computes the signature value of the Vote Cast Return Code S_{\text{VCC}}{id} \leftarrow \text{Sign}(VCC_{id}, \text{VCC}_{\text{sk}})

\text{\[]It is assumed that no two voters id \neq id' have an identical VC_{id} = VC_{id'}\]
Next, the print office engages with CCR\(_1\) and CCR\(_2\) in the computation of the pre-choice return codes (\(pCC\text{id}_1, \ldots, pCC\text{id}_n\)) and the pre-vote cast return code \(pVCC\text{id}\) for each voter \(i\in ID\). The interaction consists of two rounds. In the first round the print office does the following:

- encrypts under the Secure Data Manager public key \(pk_{SDM}\) the set of voting options \(\{v_1, \ldots, v_n\}\) as
  \[
  (ctv^1\text{id}, \ldots, ctv^n\text{id}) := (\text{Enc}(pk_{SDM}, v_1), \ldots, \text{Enc}(pk_{SDM}, v_n))
  \]
- computes the confirmation message \(Ct^id = (B^id \cdot k^id)^2\) and a hash of it \(hbck^id = H(Ct^id)^2\) and sets \(ctbck^id := \text{Enc}(pk_{SDM}, hbck^id)\)
- set \(\text{init}_{CCR} = (\psi, pk_{SDM}, ((VC\text{id}, (ctv^i\text{id}, \ldots, ctv^n\text{id}), ctbck^id))_{i\in ID}\)
- broadcast \(\text{init}_{CCR}\) to the control components CCR\(_j\) for \(j = 1, 2\)

In the second round of interaction the print office does the following (see also Setup.CCR\(_j(\text{init}_{CCR})\)):

- on reception of \(\text{init}^j_{SDM}\) from CCR\(_j\), parse it as:
  \[
  \left(k'_j, \{pk_{CCR}^{(i)}\}_{i=1}^\psi, \left\{(VC\text{id}, K_{c^id}, K_{id^i}, \left\{(ctv_i^id)^{k_{id}}, (ctv_i^id)^{k_{id}^2}\right\}_{i=1}^n, (ctbck^id)^{k_{id}^2}\right\}_{i\in ID}ight)\]
- computes the choice return codes public encryption key
  \[
  pk_{CCR}^{(i)} := pk_{CCR}^{(i)}_1 \cdot pk_{CCR}^{(i)}_2
  \]
  where the CCR\(_j\)’s choice return codes encryption public keys \(pk_{CCR}^{(i)}_j\) is extracted from \(\text{init}^j_{SDM}\)
- computes encryptions \(ctp_{c^id}\) of pre-Choice Return codes \(\{pC_i^id, \ldots, pC_n^id\}\) by exponentiating with the verification card private key:
  \[
  ctp_{c^id} := \left((ctv_i^id)^{k_{id}}, (ctv_i^id)^{k_{id}^2}\right)^{k_{id}}
  \]
  for \(i = 1, \ldots, n\)
- computes an encryption of the pre-Vote Cast Return Code as
  \[
  ctp_{VCC} := (ctbck^id)^{k_{id}^2} \cdot (ctbck^id)^{k_{id}^2}
  \]
- pre-Choice Return Codes are decrypted as
  \[
  (pC_1^id, \ldots, pC_n^id) := (\text{Dec}(sk_{SDM}, ctp_{c^id}^1), \ldots, \text{Dec}(sk_{SDM}, ctp_{c^id}^n))
  \]
- the pre-Vote Cast Return Code is decrypted as \(pVCC^id := \text{Dec}(sk_{SDM}, ctp_{VCC})\)

Last, the print office generates the long choice return codes \(lCC_1^id, \ldots, lCC_n^id\) and long vote cast code, \(lVCC^id\) for each \(i\in ID\) and the mapping table as follows:
Please note, that entries in the mapping table are shuffled to avoid trivial correlation.

δ and (\(\{π_{CTVj}^{id}\}\) − computes different NIZK proofs, where the
i-th proof \(\overline{\text{ProveEq}}((g,ctv_{id}^{j},K_{id}^{j},\overline{\text{ctv}_{id}^{j}},k_{id}^{j})\)) proves that \(\overline{\text{ctv}_{id}^{j}}\) is computed by raising the elements of \(\text{ctv}_{id}^{j}\) to the Voter choice return code generation private key \(k_{id}^{j}\) corresponding to the public key \(K_{id}^{j}\). The proof is logged to be validated during the audit phase.

7 \(δ\) stands for the mask generation function defined in ISO-18033-2 and in PKCS#1v2.2.
8 Please note, that entries in the mapping table are shuffled to avoid trivial correlation.
- computes ciphertext $\text{ctbck}^{id} := (\text{ctbck}^{id})^{kc^{id}_j}$
- computes the following NIZK proof

$$\pi_{\text{exp}} := \text{ProveEq}((g, \text{ctbck}^{id}, Kc^{id}_j, \text{ctbck}^{id}, kc^{id}_j))$$

proves that $\text{ctbck}^{id}$ is computed by raising the elements of $\text{ctbck}^{id}$ to the secret key $kc^{id}_j$ corresponding to the public key $Kc^{id}_j$. The proof is logged to be validated during the audit phase.

Last, sets $\text{init}^j_{SDM} :=$

$$\left(K^{j'}, \left\{pk^{(i)}_{CCR_j}\right\}_{i=1}^{\psi} , \left\{(VC^{id}_j, Kc^{id}_j, K^{id}_j, \left\{(ctv^{id}_i)^{k^{id}_j}\right\}_{i=1}^{n}, (\text{ctbck}^{id})^{kc^{id}_j}\right\}_{id \in ID}\right\}_{i=1}^{\psi} \right),$$

and outputs $\text{init}^j_{SDM}$ and key pairs $\left\{(pk^{(i)}_{CCR_j}, sk^{(i)}_{CCR_j})\right\}_{i=1}^{\psi} \text{ and } (K^{j'}, k^{j'})$

3. $\text{Setup.Verify}(\left(\text{init}^j_{SDM}, II^{j}_{i=1,1}\right))$ is executed by auditors to verify the computation performed by CCRs. It takes as input all CCRs’ responses $\text{init}^j_{SDM}$ and NIZKs of correct exponentiation $II^{j}$.

If for every $VC^{id}_j, j = 1, 2 \text{ and } i = 1, \ldots, n$ both verifications $\text{VerifyExp}(g, ctv^{id}_i, K^{id}_j, ctv^{id}_i)$ and $\text{VerifyExp}(g, \text{ctbck}^{id}_j, Kc^{id}_j, \text{ctbck}^{id}_j)$ hold, the algorithms outputs 1. Otherwise it returns ⊥.

This ends the description of $\text{Setup}(1^\lambda, ID)$.

### 4.4 Voting phase algorithms

The voting phase consists on the following algorithms.

#### 4.4.1 GetKey($SVK^{id}_1, VCks^{id}_1$)

On input of the start voting key $SVK^{id}_1$ and the verification card keystore $VCks^{id}_1$, it does the following actions:

- Generates the keystore encryption symmetric key $KSkey^{id}_1 \leftarrow \text{PBKDF2}(SVK^{id}_1, KEYseed)$.
- Runs the Dec algorithm with inputs $VCks^{id}_1$ and $KSkey^{id}_1$, and recovers the verification card private key $k^{id}_1$.

It returns the verification card private key $k^{id}_1$.

#### 4.4.2 CreateVote $\left(EL_{pk}, \left\{pk^{(i)}_{CCR}\right\}_{i=1}^{\psi}, VC^{id}_1, \left\{v_i\right\}_{i=1}^{\psi}, K^{id}_1, k^{id}_1\right)$

It takes as input the election public key $EL_{pk}$, the choice return codes encryption public keys $pk_{CCR} = \left\{pk^{(i)}_{CCR}\right\}_{i=1}^{\psi}$, the verification card ID $VC^{id}_1$, a set of voting options selected by the voter $\{v_1, \ldots, v_\psi\}$ and the verification card key pair $(K^{id}_1, k^{id}_1)$, and does the following:
- Computes the aggregation of the voter’s selections: \( \nu = \prod_{i=1}^{\ell} v_i \).
- Encrypts the previous result: \( E_1 = (c_1, c_2) \leftarrow \text{Enc}(\nu, \text{EL}_{pk}; r) \)
- Generates \( \pi_{sch} \leftarrow \text{ProveExp}((g, E_1), r) \) a schnorr proof (knowledge of the exponent \( r \) of the first component of \( E_1 \)). Recall \( r \) is the encryption randomness used to compute \( E_1 \), and \( \text{VC}_{id} \) is used as auxiliary information.
- Computes partial choice return codes as \( \{p_{CC}^{\psi}_{i}\}_{i=1}^{\psi} = (v^1_{x_1}, \ldots, v^\psi_{x_\psi}) \)
- Computes an ElGamal multiple encryption of those codes as

\[
E_2 := m\text{Enc}\left((p_{CC}^{(1)}_{1}, \ldots, p_{CC}^{(\ell)}_{\psi}), (p_{CC}^{1}_{id}, \ldots, p_{CC}^{\psi}_{id})\right) = \\
= (g^{\nu}, (p_{CC}^{(1)}_{1})^{r}, p_{CC}^{1}_{id}, \ldots, (p_{CC}^{(\psi)}_{\psi})^{r}, p_{CC}^{\psi}_{id})
\]

- Computes \( \overline{E}_1 := (\overline{c}_1, \overline{c}_2) = (c_1^{x_1}, c_2^{x_\psi}) \) and \( \overline{E}_2 := (\overline{c}_1, \overline{c}_2) = (g^{\nu}, (p_{CC}^{(1)}_{1})^{r}, p_{CC}^{1}_{id}, \ldots, (p_{CC}^{(\psi)}_{\psi})^{r}, p_{CC}^{\psi}_{id}) \)
- Generates two NIZK proofs to prove that the voting options in the ciphertext \( E_1 \) and the voting options used for encrypting the partial Choice Return Codes in \( E_2 \) are the same:

- \( \pi_{exp} = \text{ProveEq}(g, E_1, K_{id}, \overline{E}_1, K_{id}) \) proves that \( E_1 \) is computed by raising the elements of \( E_1 \) to the verification card private key \( k_{id} \) corresponding to the verification card public key \( K_{id} \)

- \( \pi_{pleqenc} = \text{ProveEqEnc}(g, E_{pk}, \prod_{i=1}^{\ell} p_{CC}^{(i)}_{1}, \overline{E}_1, E_2, r, k_{id}, r') \) which proves that \( E_1 \) is an encryption under the election public key \( E_{pk} \) of the product of partial Choice Return Codes \( \{p_{CC}^{(i)}_{\psi}\}_{i=1}^{\psi} \) contained in \( E_2 \)

The output of this algorithm is the ballot \( b = (E_1, E_2, \overline{E}_1, E_2, K_{id}, P) \), where \( P := (\pi_{sch}, \pi_{exp}, \pi_{pleqenc}) \).

### 4.4.3 ProcessVote \((E_{pk}, p_{CC}^{(1)}_{1}, \ldots, p_{CC}^{(\ell)}_{\psi}, \text{VC}_{id}, b)\)

It receives as input the election public key \( E_{pk} \), the Choice Return Codes Encryption public keys \( p_{CC}^{(1)}_{1}, \ldots, p_{CC}^{(\ell)}_{\psi} \), a verification card id \( \text{VC}_{id} \) and a ballot \( b = (E_1, E_2, \overline{E}_1, E_2, K_{id}, P) \). It proceeds to validate the NIZK proofs \( \pi_{sch}, \pi_{exp}, \pi_{pleqenc} \) from the ballot \( b \) running:

- \( \text{VerifyExp}(g, E_1, \pi_{sch}) \)
- \( \text{VerifyEq}(g, E_1, K_{id}, \overline{E}_1, \pi_{exp}) \)

- \( \text{VerifyEqEnc}(g, E_{pk}, p_{CC}^{(1)}_{1}, \overline{E}_1, E_2, \pi_{pleqenc}) \) where \( p_{CC}^{(i)} := \prod_{i=1}^{\ell} p_{CC}^{(i)} \)

In case any of the validations fail, it stops and outputs 0. Otherwise, it outputs 1. This is the algorithm used by the voting server and the return codes control components to test a ballot.
4.4.4 CreateCC\((p_{\sigma}^{(1)}_{\text{CCR}}, \ldots, p_{\sigma}^{(n)}_{\text{CCR}}, \text{VC}_{\text{id}}, b), \text{CreateCC}()\), \{CreateCC.ccr\(_j()\)\} \_{j=1}^2\):

It is an interactive algorithm run by the voting server\(^9\) and the return codes control components CCR\(_1\), CCR\(_2\), that receives as input a security parameter \(1^\lambda\) and proceeds as shown next:

1. CreateCC\(\text{serv}(p_{\sigma}^{(1)}_{\text{CCR}}, \ldots, p_{\sigma}^{(n)}_{\text{CCR}}, \text{VC}_{\text{id}}, b)\)

   - If ProcessVote\((E_{\text{pk}}, p_{\sigma}^{(1)}_{\text{CCR}}, \ldots, p_{\sigma}^{(n)}_{\text{CCR}}, \text{VC}_{\text{id}}, b)\) outputs 1, sends \((\text{VC}_{\text{id}}, b)\) to CCR\(_1\) and CCR\(_2\). Otherwise abort. (Recall that \(E_{\text{pk}}\) is the election public key, and \(p_{\sigma}^{(i)}_{\text{CCR}}\) the \(i\)-th element of the Choice Return Codes Encryption Key used to encrypt the \(i\)-th voting option.)

   - On reception of \((E2)^{k}_{id}\) from CCR\(_j\) for \(j = 1, 2\), the voting server computes \(E2_{\text{cc}} := (E2)^{k}_{1} \cdot (E2)^{k}_{2}\).

   - Parse \(E2_{\text{cc}} = (g^{r'} \cdot \psi_{\text{CCR}}, p_{\text{CC}1}^{(i)} \cdot \ldots \cdot (p_{\text{CC}1}^{(i)})^{r'} \cdot p_{\text{CC}2}^{(i)}\) where \(k := \sum_{j=1}^{2} k_{id}\) and \(k := k_{id} \cdot \hat{k}.\) Send \(g^{r'} \cdot \psi\) to CCR\(_1\) and CCR\(_2\).

   - On reception of \((g^{r'} \cdot \hat{k})_{ccr}^{(i)} \cdot \ldots \cdot (g^{r'} \cdot \hat{k})_{ccr}^{(i)}\) from CCR\(_j\) for \(j = 1, 2\), the voting server computes partial decryptions

     \[
     \text{pdec}_i := g^{-(r' \cdot \hat{k})_{ccr1}^{(i)}} - (p_{\text{CC}1}^{(i)})^{r' \cdot \hat{k}}
     \]

     for \(i = 1, \ldots, \psi\).

   - Next for \(i = 1, \ldots, \psi\) compute pre-choice return codes as \(p_{\text{CC}1}^{(i)} := (E2_{\text{cc}})_{i+1} \cdot \text{pdec}_i\) where \((E2_{\text{cc}})_{i+1}\) is the \((i+1)\)-th component of \(E2_{\text{cc}}\).

   - Output \(p_{\text{CC}1}^{(i)}, \ldots, p_{\text{CC}1}^{(i)}\).

2. CreateCC.ccr\(\{\{p_{\sigma}^{(i)}_{\text{CCR}}, \text{sk}_{\text{CCR}}^{(i)}\}_{i=1}^{\psi}, \text{VC}_{\text{id}}, b, k'\}_j\) takes as input a Verification Card ID \(\text{VC}_{\text{id}}\), and a ballot \(b\), sent by the Voting Server, as well the CCR\(_j\)’s Choice Return Codes Generation private key \(k'\). The return code control component maintains a list \(L_{\text{ccv}}\) with the verification card ids that have been already queried (the list is initialized to the empty list). It proceeds as follows:

   - If \(\text{VC}_{\text{id}} \in L_{\text{ccv}}\), abort. Else, set \(L_{\text{ccv}} := L_{\text{ccv}} \cup \{\text{VC}_{\text{id}}\}\).

   - If ProcessVote\((p_{\sigma}^{(1)}_{\text{CCR}}, \ldots, p_{\sigma}^{(n)}_{\text{CCR}}, \text{VC}_{\text{id}}, b)\) outputs 0 then abort. Otherwise, let \(b := (E1, E2, \text{E1}, \text{E2}, k_{id}, P)\). Then compute \((E2)^{k}_{id}\), where \(k_{id} := \delta(\text{VC}_{\text{id}}, k'_{j})\) \(^{11}\).

---

\(^9\)For our analysis we have merged the Election Context, the Voting Workflow Context and the Vote Verification Context from \(^{12}\) into a single agent called Server. The rationale behind it is that all three contexts could be adversarially controlled, and hence we can group them into a single agent for security purposes.

\(^{10}\)The reason of having multiple public keys is to use same randomness across the \(\psi\) encryptions generated in the voting device.

\(^{11}\)Recall that \(k'_{id}\) is the Voter Choice Return Code Generation private key.
- computes the following NIZK (implicitly, \( \psi \) different proofs)

\[
\sigma_1^{exp} := \text{ProveEq}( (g, E^2, K^j_{id}, E^2_{k^{id}}, k_{id}) )
\]

proves that \( E^2_{k^{id}} \) is computed by raising the elements of \( E^2 \) to the Voter choice return code generation private key \( K^j_{id} \) corresponding to the Voter choice return code generation public key \( K^j_{id} \). The proof is logged to be validated during the audit phase.

- Send back to the voting server \( \bigl( E_{k^{id}} \bigr) \)

- On input \( g^r' \hat{k} \) from the voting server, where \( \hat{k} := \sum_{j=1}^{2^\psi} k_{id}^j \), computes \((d_1, \ldots, d_\psi) := \{(g^r' \hat{k})^{sk_{CCR}^{(j)}}, \ldots, (g^r' \hat{k})^{sk_{CCR}^{(\psi)}}\} \)

- computes the following NIZK

\[
\sigma_2^{exp} := \text{ProveEq}( (g, g^r' \hat{k}, \prod_{i=1}^{\psi} p_{CCR}^{(i)}_{k_{CCR}^{(i)}}, \prod_{i=1}^{\psi} d_i, \prod_{i=1}^{\psi} sk_{CCR}^{(i)}) )
\]

proves that each \( d_i \) is indeed computed by raising \( g^r' \hat{k} \) to the CCR Choice Return Code Encryption private key \( sk_{CCR}^{(i)} \). The proof is logged to be validated during the audit phase.

- Send back to the voting server \( \bigl( (g^r' \hat{k})^{sk_{CCR}^{(1)}}, \ldots, (g^r' \hat{k})^{sk_{CCR}^{(\psi)}} \bigr) \)

### 4.4.5 \text{CreateRC}(p_{Cid}^{(1)}, \ldots, p_{Cid}^{(\psi)}, V_{Cid}, C_{sk}, \text{CMtable})

It outputs a set of choice return codes \( \{CC_{id}^{(1)} \}^{\psi}_{i=1} \) and consists of the following steps:

- let \( lCC_{id}^{(i)} = H(p_{Cid}^{(i)} || V_{Cid}) \), and Choice Return Code encryption symmetric key \( sk_{CC}^{(i)} = \delta(H(lCC_{id}^{(i)} || C_{sk})) \) for \( i \leq \psi \), where \( C_{sk} \) is the Code Secret key, and \( p_{Cid} \) are the pre-choice return codes computed by the voting server.

- output choice return codes

\[
CC_{id}^{(i)} = \text{dec}^i(\text{Enc}(CC_{id}^{(i)}; sk_{cc}^{(i)}); sk_{cc}^{(i)} )
\]

iff \( [H(1CC_{id}^{(i)})], \text{Enc}^i(\text{CC}_{id}^{(i)}; sk_{cc}^{(i)})] \) exists as an entry in \( \text{CMtable} \) \( \forall i \leq \psi \).

- else output \( \perp \)

This is the algorithm used by the voting server to compute the choice return codes \( \{CC_{id}^{(1)} \}^{\psi}_{i=1} \) to be sent to the voting client.

### 4.4.6 \text{AuditCodes}((CC_1, \ldots, CC_\psi), (CC_{id}^{(1)}, \ldots, CC_{id}^{(\psi)}))

This is the algorithm used by the voter to check whether all expected choice return codes \( \{(v_{id}^{(1)}, CC_{id}^{(1)})\}^{\psi}_{i=1} \) corresponding to the voter’s intended choices \( \{v_{id}^{(1)}\}^{\psi}_{i=1} \) were indeed received. It outputs a boolean and consists of the following steps:

- output 1 iff \( CC_i \in \{CC_{id}^{(1)}, \ldots, CC_{id}^{(\psi)} \} \forall i = 1, \ldots, \psi \)

- else output 0
4.4.7 Confirm\((VC_{id}, b, k_{id}, BCK_{id})\)

It receives as input a verification card id \(VC_{id}\), a ballot \(b\), the verification card private key \(k_{id}\) and the voter’s ballot casting key \(BCK_{id}\), and outputs the confirmation message \([CM_{id} = (BCK_{id})^{2k_{id}}]\).

4.4.8 ProcessConfim\((bb, VC_{id}, CM_{id}, C_{sk}, VCCs_{pk})\)

It is run by the voting server and the control components CCRs. It receives as input a ballot box \(bb\), a verification card id \(VC_{id}\), a confirmation message \(CM_{id}\), the codes secret key \(C_{sk}\) and the vote cast return code signer public key \(VCCs_{pk}\). It performs the following steps:

- The voting server checks the following two things: there is a ballot in \(bb\) with id \(VC_{id}\), and this ballot has not been yet confirmed.

- The voting server sends \(CM_{id}\) to each component CCR\(_1\), CCR\(_2\).

- Each CCR\(_i\) checks if the number of confirmation attempts has been reached. If the voter exceeded the number of confirmation attempts – return ⊥. Otherwise, log the query from the voting server and the answer.

- Each CCR\(_i\) computes \(hcm_{id} = (H(CM_{id}))^{2k_{id}}\),

  where \(k_{id} := \delta(VC_{id}||ConfirmStr||k’_{j})\)

  and sends it back to the voting server.

- Each CCR\(_j\) generates the following NIZK proof of correct exponentation

\[\pi_{exp} := \text{ProveEq}((g, (H(CM_{id}))^{2}, Kc_{id}, hcm_{id}, k_{id}^j))\]

The proof is logged to be validated during the audit phase.

- The voting server computes the corresponding pre-vote cast return code as \(pVCC_{id} := hcm_{id} \cdot hcm_{2}\).

- The voting server computes the long vote cast return code as \(1VCC_{id} = H(pVCC||VC_{id})\) and the vote cast return code encryption symmetric key \(skvcc_{id} = \delta(H(1VCC_{id}||C_{sk}))\).

- The voting server takes the Codes Mapping Table \(CMtable\) and checks if there exists an entry \([H(1VCC_{id}), Enc^c((VCC_{id}||S_{VCC})|skvcc)]\). If so, recovers the Vote Cast Return Code \(VCC_{id}\) and the signature \(S_{VCC}\), using the decryption algorithm \(Dec^c\) with \(skvcc_{id}\) as the key

- The voting server checks that the retrieved short vote cast return code is correct by running \(Verify(VCCs_{pk}, VCC_{id}||S_{VCC})\).

---

\(^{12}\)Also, the voting device signs the confirmation message with the credential id signing key of voter \(id\).

\(^{13}\)Recall that \(k_{id}\) is the Voter Vote Cast Return Code Generation private key, and \(k’_{j}\) is the CCR\(_j\)’s Choice Return Code Generation private key.
4.5 Tally phase algorithms

The tally phase consists on the following algorithms.

4.5.1 Cleansing(ELpk, pkCCR(1), ..., pkCCR(s), VCCspk, BB)

This function can either be run by the voting server based on the content of his BB or by the auditors based on the Global BB. It proceeds as follows:

- let BB = \{ (VCid, b, VCCid, VCC+cid) \}_{id \in ID}
- it creates a list L_{checked} with all (VCid, b) ∈ bb such that ProcessVote(ELpk, pkCCR(1), ..., pkCCR(s), VCid, b) = 1 and Verif(VCCspk, VCCid||VCC+cid) = 1, where (VCid, VCCid, VCC+cid) are not null.
- from each (VCid, b) ∈ L_{checked}, obtains E1 from b. Next, parses E1 as ciphertext components (c1, c2), and adds (c1, c2) to a list L

The output is the list L = \{ (c1, c2) \}

4.5.2 MixDec1(ELpk(1), L = \{ (c1, c2) \})

It is run by CCM1 and proceeds as follows:

- from the ciphertext list L = \{ (c1, c2) \} computes \( L^* = \{ (\tau_1, \tau_2) \}_{mix} \) and a proof \( P_n \) of correct mixing
- from \( L^* \) it computes a partially decrypted list \( L_1 = \{ (c_1^{(1)}, c_2^{(1)}) \}_{dec} \) and a list of zero-knowledge proofs \( P_{dec} := \{ \pi_{dec} \} \) of correct partial decryptions computed as \( \pi_{dec} := \text{ProveDec} \left( (\tau_1, \tau_2, (\tau_1 - EL_{pk}(1)), EL_{pk}(1)) \right) \) for each \( (\tau_1, \tau_2) \in L^* \)
- output \( \left( L^* = \{ (\tau_1, \tau_2) \}_{mix}, L_1 = \{ (c_1^{(1)}, c_2^{(1)}) \}_{dec}, P_n, P_{dec} \right) \)

The output above is such that if we rewrite \( L = \{ \text{Enc}(EL_{pk}, V) \} \) and \( L_1 = \{ \text{Enc}(EL_{pk}(2), V') \}_{mix} \), then \( L_1 = \{ \text{Enc}(EL_{pk}(2), V) \}_{mix} \).

4.5.3 MixDec2 (ELpk(2), L = \{ (c1, c2) \}, L_1 = \{ (c_1^{(1)}, c_2^{(1)}) \}_{mix}, P_n, P_{dec})

It is run by CCM2 and proceeds as follows:

- from the ciphertext list \( L_1 = \{ (c_1^{(1)}, c_2^{(1)}) \}_{dec} \) computes \( (L_1^*, P_n) := \text{Mix}(EL_{pk}(2), L_1) \), namely a shuffled list \( L_1^* = \{ (c_1, c_2) \}_{mix} \) and a proof \( P_n \) of correct mixing
From $L_s$ it computes a decrypted list $L_v = \{V\}_{\text{dec}}$ and a list of zero-knowledge proofs $P_{\text{dec}} := \{\pi_{\text{dec}}\}$ of correct partial decryptions computed as $\pi_{\text{dec}} := \text{ProveDec}\left(\left(\tilde{c}_1, \tilde{c}_2, \tilde{c}_2 \cdot (\tilde{c}_1) - \text{EL}(2)_{\text{sk}}\right), \text{EL}(2)_{\text{sk}}\right)$ for each $(\tilde{c}_1, \tilde{c}_2) \in L_s$.

- output $(L_s^*, L_v, P_{\text{dec}})$

The output above is such that if we rewrite $L = \{\text{Enc}(\text{EL}_\text{pk}, V)\}$ and $L_1 = \{\text{Enc}(\text{EL}_\text{pk}, \{V\}_\text{mix})\}$, then $L_v = \{\{V\}_\text{mix}\}_{\text{mix}}$.

Finally, $\text{fact}(V_i)$ is run for every $V_i \in L_v$ to output the prime factors $\{v_i\}_{i=1}^\omega$ of $V_i$. It is tested that the combination of voting options $\{v_i\}_{i=1}^\omega \in \Omega$. Otherwise, the vote is discarded.

### 4.6 Verify phase algorithm

The verify phase consists on the following algorithms.

#### 4.6.1 VerifyTally($bb, bb_1^{CCR}, bb_2^{CCR}, bb_1^{CCM}, bb_2^{CCM}, r, \Pi$)

It takes as input the Server’s ballot box $bb$, logs of CCR1, CCR2, CCM1, CCM2 denoted as $bb_1^{CCR}, bb_2^{CCR}, bb_1^{CCM}, bb_2^{CCM}$, the tally result $r$ and the proof $\Pi$ of correct tally (this proof includes outputs of the first and second CCMs). Initially it runs AuditorVerify and aborts if the output is 0. Then it performs the following operations:

- if in $bb_2^{CCM}$ it’s logged that more than one distinct input had been shuffled for the given Ballot id, aborts.
- if $\text{MixVerify}\left(\left(L_1, L_s^*, P_{\text{dec}}\right) = 0$ aborts
- if some of the decryption proofs in $P_{\text{dec}}$ does not verify aborts
- then it checks that the list of voting choices corresponds to factoring the elements in $L_v$

If all the validations are successful, the process outputs 1. If any validation fails, it outputs 0.

#### 4.6.2 AuditorVerify($bb, bb_1^{CCR}, bb_2^{CCR}, bb_1^{CCM}, bb_2^{CCM}, L, L^*, L_1, \Pi_m, \Pi_{\text{dec}}$)

It takes as input the Server’s ballot box $bb$, logs of CCR1, CCR2, CCM1 denoted as $bb_1^{CCR}, bb_2^{CCR}, bb_1^{CCM}$, cleared Ballot Box $L$, and the output of the first mixing and partial decryption process $L^*, L_1, \Pi_m, \Pi_{\text{dec}}$. This function is run by Auditors and aims to verify the correctness of the cleansing process and also mixing and partial decryption performed by the first CCM1 (in case there are more than 2 CCMs, the auditors perform verification of all operations done before the final mixing and decryption).

1. If information about received and confirmed ballots in $bb, bb_1^{CCR}, bb_2^{CCR}$ does not match, aborts.
2. If computed long choice codes and long vote cast codes are not extractable from \( \text{bb}, \text{bb}_1^{CCR}, \text{bb}_2^{CCR} \) and CMtable aborts.

3. Re-computes the cleansed ballot box by running Cleaning on bb and verifies that the output matches \( L \).

4. If the list \( L \) is not identical to the results of cleansing process performed by the auditors, aborts and returns 0.

5. If in \( \text{bb}_1^{CCM} \) it’s logged that more than one distinct input had been shuffled for the given \( \text{BB}_{\text{id}} \), aborts.

6. If \( \text{MixVerify}(L, L^*, P_s) = 0 \), aborts.

7. If some of the decryption proofs in \( P_{\text{dec}} \) does not verify, aborts.

If all the validations are successful, the process outputs 1. If any validation fails, it outputs 0.

### 4.7 Phases in the protocol

The execution flow is organized in the following phases. As noted in Section 4.1, we do not aim to fully describe the phases specified in [18] but rather only the relevant parts used in the security analysis.

**Election configuration.** The election authorities (the system operator in Chancellery’s requirements 2.2.6 and 2.5.1) set up the public parameters of the election such as the list of voting options \( \{v_1, \ldots, v_\psi\} \), the set of valid votes (combinations of voting options) \( \Omega \) and the result function \( \rho \). All control components generate their private keys. The Print office computes the election public key \( EL_{\text{pk}} \) on inputs the shares \( EL_{\text{pk}}^{(1)}, EL_{\text{pk}}^{(2)} \) generated by the Mixing Control Components CCM1, CCM2 respectively.

The Print office runs the Setup algorithm in conjunction with the Return Codes Control Components CCR1, CCR2. At the end of the Setup phase the Print office publishes the resulting Electoral Board public key \( EL_{\text{pk}} \), the Vote Cast Return Code Signer key \( VCC_{\text{pk}} \) and the empty voter list \( V_{\text{id}} \) in the bulletin board BB. The Codes secret key \( C_{\text{sk}} \) is provided to the Voting Server.

For each voter with pseudonym \( \text{id} \), at the end of the setup phase the voter’s verification card data is computed as \( \{SVK_{\text{id}}, V_{\text{id}}, BCK_{\text{id}}, VCC_{\text{id}}, \{v_i, CC_{\text{id}}^{1:n} \}_{i=1} \} \).

**Voting phase.** This phase consists of several steps:

1. The voter provides \( SVK_{\text{id}} \) to the voting device, which runs the GetKey algorithm to open it and recover the Verification Card private key \( k_{\text{id}} \). At that point the voting device is prepared to create a vote.

2. The voter provides the set of selected voting options \( \{v_1, \ldots, v_\psi\} \in \Omega \) to the voting device. The voting device runs the CreateVote algorithm, producing a ballot \( b \).

---

\(^{14}\text{See [33, Section 8.5] for a precise meaning of extractable codes}\)
3. The ballot $b$ is sent to the Voting Server together with the Verification Card ID $VC_{id}$.

4. Upon reception of $(VC_{id}, b)$, the Voting Server runs the ProcessVote algorithm to verify the incoming vote. In case the result is 1, the ballot box $bb$ is updated with the pair $(VC_{id}, b)$ and the process continues. Otherwise, the process stops and the voting device receives an error message.

5. The Voting Server runs the CreateCC protocol in interaction with the Control Components $CCR_1$, $CCR_2$ and then CreateRC using the Codes Mapping Table $CMtable$ and the Codes secret key $Csk$. In case the execution is successful, it sends back to the voting device the generated Choice Return Codes, which are shown to the voter. Otherwise, the process stops and the voter receives an error message.

6. Before confirming her choice, the voter can ask the Voting Server to return the Choice Return Codes as many times as required.

7. The voter then compares that the received Choice Return Codes match those in her voting card linked to her selections (this action is captured by AuditCodes). In case the verification is satisfactory, the voter provides her Ballot Casting Key $BCK_{id}$ to the voting device, which generates a Confirmation Message $CM_{id}$ using the Confirm algorithm, that is sent to the bulletin board manager with the Verification Card ID $VC_{id}$. Otherwise the voter does not confirm her vote, and she can opt to vote through another channel.

8. The Voting Server then runs ProcessConfirm using as input the received $CM_{id}$ and interacting the Choice Return Codes Control Components. In case the operation is successful (the output is different from ⊥), it updates the Ballot Box with the retrieved Vote Cast Return Code $VCC_{id}$ and signature $S_{VCC_{id}}$, and sends the Vote Cast Return Code to the voter, who checks it matches the Vote Cast Return Code $VCC_{id}$ in her voting card. Otherwise, an error message is sent.

9. After the previous step, the voter can request the Voting Server to retrieve and show the value $VCC_{id}$ as many times as she requires until the end of the election.

At this point, in case the Vote Cast Return Code received by the voter is different from the one expected or an error has been returned, the voter may try to confirm her vote from another device or complain to the authorities, who may check if a vote has been indeed confirmed by that voter in the system. In case it has not, the voter may try to confirm her vote from another device or cast her vote through another channel.

COUNTING PHASE. It consists of several steps. We observe that in case there are more than two Mixing Control Components, step 2 below is executed before the last Mixing Control Component receives its inputs.

1. The Mixing Component $CCM_1$ runs algorithm $MixDec_1$. 

36
2. The auditors run algorithm \texttt{AuditorVerify} using as input the contents in the ballot boxes of the Voting Server together with those of Return Codes Control Components. In case the output is 0 an investigation is opened to detect the reason of failure.

3. The (last) Mixing Component CCM\textsubscript{2} runs algorithm \texttt{MixDec\textsubscript{2}}, obtaining and publishing in the bulletin board \texttt{BB} the result \( r \) and the proof \( \Pi \).

4. The auditors run algorithm \texttt{VerifyTally}. In case their output is 1, the result \( r \) is announced to be fair. Otherwise, an investigation is opened to detect the reason of failure.

5 Privacy

In this section we define the notion of ballot privacy for the protocol described in Section 4, and we prove it fulfills it.

We base our demonstration on the one provided in \cite{10} for the Helios scheme, though we also refer to the privacy definition given for the fully distributed Helios in \cite{15}. In \cite{10}, the authors start with a game where the adversary is presented with \texttt{BB}_0 and make gradual changes in order to end up with a game where the adversary is presented with \texttt{BB}_1. In each new game the adversary can notice the modifications with negligible probability. Since our protocol generates ballots with more information than the ciphertext itself, first we will show that such information is indistinguishable from random values. Then we prove the following theorem:

\textbf{Theorem 1} Let \((\text{Gen}, \text{Enc}, \text{Dec})\) combined with a NIZK proof be an NM-CPA secure encryption scheme, \((\text{ProveDec}, \text{VerifyDec}, \text{SimDec}), (\text{ProveEq}, \text{VerifyEq}, \text{SimEq})\) be NIZKPK schemes with the zero-knowledge property and the exponentiation function in \(G\) be a pseudo-random function. Then the protocol presented in Section 4 satisfies the ballot privacy property in the random oracle model.

Although the games are represented with one voting option for simplicity, they can be extended to ballots with multiple options as far as all of them are different and the number of voting options the voter can select is fixed\footnote{Otherwise, a malicious voting device might send more options than voter intended and hide choice codes corresponding to the adversarial options}. This means that, in case the voter can select up to \( t \) voting options and does not select all of them, the rest of the ballot will be filled with blank options which are all different, and the voter will receive choice return codes corresponding to such blank options besides her selection.

Please notice that in both \cite{10} and \cite{15} the adversary is given an indirect access to a ballot box that is controlled by a challenger. In our protocol implementation, an adversary has full control over the global ballot box since the server is not trusted. However, each control component maintains its own database and logs information about each operation it performs. An honest auditor is able to verify the correctness of the global ballot box and obtain ballots
that were confirmed by all CCRs. If found any discrepancy, an independent investigation is conducted to detect the reason of failure. In the games described below, we refer to a database maintained by an honest CCR as challenger’s database to which an Attacker is given an indirect access.

In order to be consistent with [10] and [15], where the Attacker is given a power to corrupt voters adaptively, we split the Setup algorithm described in Section 4 into Config, Config.CCRj and Registration to allow dynamic voter registration. We claim that this change does not affect the security of the entire scheme and serves merely for proof readability.

Config(1λ) runs on input a security parameter 1λ and proceeds as shown next:

- It starts by generating
  - an ElGamal encryption key pair \((pk_\text{SDM}, sk_\text{SDM}) \leftarrow \text{Gen}_e(1^\lambda)\)
  - a Codes secret key \(Csk \leftarrow T\), where \(T\) denotes the set of possible keys of the PRF function \(f\)
  - a Vote Cast Return Code Signer key pair \((VCC_{pk}, VCC_{sk}) \leftarrow \text{Gen}_e(1^\lambda)\).

- The output of the algorithm is \((pk_\text{SDM}, sk_\text{SDM}, Csk, VCC_{pk}, VCC_{sk})\)

Config.CCRj(1λ) runs on input a security parameter 1λ

- computes Control Components Choice Return Codes key pairs \((pk_{(i)\text{CCR}_j}, sk_{(i)\text{CCR}_j}) \leftarrow \text{Gen}_e(1^\lambda)\) for \(i = 1, \ldots, \psi\).

- generates a random Choice Return Code generation private key \(k'_j\) that will be used in Key Derivation Function \(\text{deltak}_j(\cdot) := \delta(k'_j, \cdot)\) and a corresponding public key \(K'_j = g^{k'_j}\).

Outputs \(\left\{ (pk_{(i)\text{CCR}_j}, sk_{(i)\text{CCR}_j}) \right\}_{i=1}^{\psi}, k'_j, K'_j \)

Registration(1λ, pkSDM, skSDM, Csk, VCCspk, VCCsak, \{Register.CCRj(\cdot)\}_{j=1}^{\psi}) is an interactive algorithm run by the Print office and the Return Codes Control Components CCR1, CCR2 that runs on input a security parameter 1λ, an ElGamal encryption key pair \((pk_{\text{SDM}}, sk_{\text{SDM}})\), a Codes secret key \(Csk\), a Vote Cast Return Code Signer key pair \((VCC_{pk}, VCC_{sk})\) and proceeds as shown next:

- generates a random Verification Card ID \(VC_{id}^{10}\)
- generates a Start Voting Key \(SVK_{id} \leftarrow A_{svk}\)
- generates a keystore for a symmetric encryption key \(K\text{Key}_{id} \leftarrow \text{PBKDF2}(SVK_{id}, KEYseed)\)
- generates the Verification Card key pair: \((k_{id}, k_{id}) \leftarrow \text{Gen}_e(1^\lambda)\)
- computes the encryption of the Verification Card private key with the keystore encryption key: \(VC_{ks} \leftarrow \text{Enc}(k_{id}; K\text{Key}_{id})\)

---

10It is assumed that no two voters \(id \neq id'\) have an identical \(VC_{id} = VC_{id'}\)
- chooses at random a Ballot Casting Key $BCKid \xleftarrow{\$} A_{bck}$
- assigns short Choice Return Codes $CCid_i \xleftarrow{\$} A_{cc}$ at random for voter $id_i$, $\forall i = 1, \ldots, n$
- assigns a short Vote Cast Return Code $VCCid \xleftarrow{\$} A_{vcc}$ at random for voter $id$
- Computes the signature value of the Vote Cast Return Code $S_{VCCid} \xleftarrow{\$} \text{Sign}(VCCid, VCCsk)$

Next, the print office initiates the computation of a series of Long Choice Return Codes $(pCid_1, \ldots, pCid_n)$ in interaction with CCR$_1$, CCR$_2$:

- encrypts under $pk_{sdm}$ the set of voting options $\{v_1, \ldots, v_n\}$ as $(ctv_{id_1}, \ldots, ctv_{id_n}) := (\text{Enc}(pk_{sdm}, v_1), \ldots, \text{Enc}(pk_{sdm}, v_n))$

Next, the print office sends $\text{init}_{CCR}$ to each CCR$_j$ and obtains back $\text{init}_{SDM}^j$, where $\text{init}_{SDM} := (K'^j, \{pk_{CCR}^j\}_{i=1}^n, \{VCid_i, Kc_{id_i}, K_{id_i}, \{ctv_{id_i}\}_{i=1}^n, \{ctbck_{id_i}\}_{i=1}^n\})$ for $j = 1, 2$. On inputs $\text{init}_{SDM}^1, \text{init}_{SDM}^2$, the print office computes a list of Choice Return Codes Public Encryption Keys $pk_{CCR}^{(1)} := pk_{CCR1}^{(1)} \cdot pk_{CCR2}^{(1)}$, $pk_{CCR}^{(2)}$ for $i = 1, \ldots, n$.

Next, the print office computes for each Verification Card identifier $VCid$ the following:

- encryptions $ctpc_{id}^i$ of pre-Choice Return codes $(pCid_1, \ldots, pCid_n)$ as $ctpc_{id}^i := ((ctv_{id_i})^{k_{id}}, (ctv_{id_i})^{k_{id}})^{k_{id}}$ for $i = 1, \ldots, n$
- encryption of the pre-Vote Cast Return Code as $ctpvcc_{id} := ((ctbck_{id_i})^{k_{id}}, (ctbck_{id_i})^{k_{id}})^{k_{id}}$

- pre-Choice Return Codes are decrypted as $(pCid_1, \ldots, pCid_n) := (\text{Dec}(sk_{sdm}, ctpc_{id}^1), \ldots, \text{Dec}(sk_{sdm}, ctpc_{id}^n))$

- the pre-Vote Cast Return Code is decrypted as $pv_{VCCid} := \text{Dec}(sk_{sdm}, ctpvcc_{id}^i)$

- long Choice Return Codes $1CCid_i = H(pCid_i|VCid) \forall i = 1, \ldots, n$, and symmetric keys $sk_{vccid} = \delta(H(1CCid_i||C_{sk}))$

- long Vote Cast Return Code $1VCCid_i = H(pVCCid_i|VCid)$ and symmetric key $sk_{vccid} = \delta(H(1VCCid_i||C_{sk}))$
- the mapping between long and short codes is defined as

\[
\text{map}^i = \left\{ [\text{H}(\text{lCC}_i), \text{Enc}^i(\text{CC}_i; \text{skcc}_i)] \right\}_{i=1}^n, [\text{H}(\text{lVCC}_i), \text{Enc}^i(\text{VCC}_i||S_{\text{VCC}}; \text{skvcc}_i)] \right\}
\]

where \( S_{\text{VCC}} := \text{Sign}(\text{VCCs}, \text{VCC}_i) \) is the validity proof for the short code \( \text{VCC}_i \n\)

Finally the print office outputs

\[
\left( \text{SVK}_i, \text{VC}_i, K_{i}, \text{VCK}_i, BCK_{i}, \text{VCC}_i, \{v_i, \text{CC}_i\}_{i=1}^n, \text{map}_i \right)
\]

1. **Register.** \( \text{CCR}_j(\text{init}_j, \text{kr}_{j}, \{ (\text{pk}_{\text{CCR}}^{(i)}, \text{sk}_{\text{CCR}}^{(i)}) \}_{i=1}^n ) \) is run by \( \text{CCR}_j \) for \( j = 1, 2 \) where

\[
\text{init}_j = (n, \text{pk}_{\text{SDM}}, \{ (\text{VC}_i, (\text{ctv}_i, \ldots, \text{ctv}_n, \text{ctbck}_i) ) \}_{i=1}^{\text{id} \in \mathcal{T}_D})
\]

is input provided by the print office and \( (\text{pk}_{\text{CCR}}^{(i)}, \text{sk}_{\text{CCR}}^{(i)}) \) are Control Components Choice Return Codes key pairs and \( \{ (\text{pk}_{\text{CCR}}^{(i)}, \text{sk}_{\text{CCR}}^{(i)}) \}_{i=1}^n \) are Control Components Choice Return Codes key pairs. Next, for every \( \text{VC}_i \) does the following:

- derives a secret key \( \text{kr}_i := \text{kr}_{j}(\text{VC}_i) \) and a public key \( \text{K}_{i} := (g)^{\text{kr}_i} \)
- derives a secret key \( \text{kc}_i := \text{kr}_{j}(\text{VC}_i) \) and a public key \( \text{K}_{i} := (g)^{\text{kc}_i} \)
- computes \( (\text{ctv}_i)^{\text{kr}_i}, \ldots, (\text{ctv}_n)^{\text{kr}_i} \) and \( (\text{ctbck}_i)^{\text{kc}_i} \)
- sets \( \text{init}_{\text{SDM}} := (k_{j}', \{ \text{pk}_{\text{CCR}}^{(i)} \}_{i=1}^n, \{ \text{VC}_i, \text{K}_{i}, \text{K}_{i}', \{ (\text{ctv}_i)^{\text{kr}_i} \}_{i=1}^n, (\text{ctbck}_i)^{\text{kc}_i} \}) \),

and outputs \( \text{init}_{\text{SDM}} \) and key pair \( (k_{j}', k_{j}') \)

2. **Setup.** \( \text{Verify} \) is identical to the one described in the Section 4.3.2 but run for a single \( \text{VC}_i \).

This ends the description of \( \text{Setup}(1^\lambda, \text{Setup.} \text{SDM}(\cdot), \{\text{Setup.} \text{CCR}_j(\cdot)\}_{j=1}^2, \text{Setup.} \text{Verify}(\cdot)) \)

### 5.1 Game E

Roughly speaking, the game of privacy consists of the following processes which are executed in different phases:

- **Configuration:** The Challenger configures an election by running \( \text{Config}(1^\lambda), \) \( \text{SetupEKey.} \text{CCM}_j(1^\lambda) \) and \( \text{Config.} \text{CCR}_j(1^\lambda) \) on behalf of the Print Office, honest CCR and CCM. Also, the Challenger flips a coin and sets up two different Local Bulletin Boards, where information from honest parties of the system would be logged, and shows one (chosen at random) to the Attacker.
• **Registration and Voting:** The Attacker $A$ can interact with the Challenger $C$ in the following way:

- **Asks to register a voter (corrupted or honest).** The Challenger executes $\text{Registration}$ and $\text{Register.CCR}$ on behalf of the Print Office and an honest CCR. Then it runs $\text{Setup}, \text{Verify}$ and aborts the game if the verification fails. Otherwise, the Challenger posts public voter information on both Local Bulletin Boards and keeps private data of honest voters. Please note, that a voter’s Encrypted Verification Card key store is assumed to be public. For corrupt voters, $A$ receives all voter’s secret information. For the honest voters, $A$ receives only the public information.

- **Asks the Challenger to cast a vote on behalf of an honest voter:** in such case, $A$ provides the Challenger with two voting options. The Challenger $C$ operates on the voter’s behalf and generates two valid ballots. $C$ keeps both ballots, but sends to the Attacker only one ballot based on his coin. This algorithm can only be executed on behalf of honest voters.

- **Asks to compute Choice Return Code:** The Attacker provides the Challenger with a ballot for which the Choice Return Code should be computed. Once the Challenger obtains the ballot, he verifies it and, if the verification is successful, executes $\text{CreateCC}$ on behalf of honest CCR. If the ballot came from a corrupted voter, Challenger logs the ballot and the fact that Choice Return Code has been computed on both Local Bulletin Board. Otherwise, he recovers the corresponding to that honest voter $b_1-\beta$ ballot and logs each ballot in the corresponding Local Bulletin Board along with a fact that Choice Return Code has been computed.

- **Asks an honest voter to confirm a vote.** The Challenger verifies the obtained Choice Return Code and, if verification is successful, generates a confirmation message. This algorithm can only be executed on behalf of honest voters.

- **Asks to compute Vote Cast Code.** The challenger receives a confirmation message for which Vote Cast Code should be computed. He verifies the signature, and if verification is successful, computes hash of that message and raises it in his secret key. For all ballots from the corrupted voters Challenger logs the fact that it has been confirmed on both Local Bulletin Boards. For an honest voter ballot, the Challenger recovers the corresponding to that honest voter $b_1-\beta$ ballot and logs each ballot in the corresponding Local Bulletin Board along with a fact that it has been confirmed.

• **Mixing and Partial Decryption:** The Adversary asks the Challenger to execute honest mixing and partial decryption optionally providing as an input the shuffled and re-encrypted ciphers with a proof of correctness. The challenger always verifies proofs on behalf of an honest auditor before the last mixing and decryption. Thus, if the first $\text{CCM}_1$ is honest, challenger executes mixing procedure, but outputs the result if and only if cleansing and mixing proofs are correct. However, if the second $\text{CCM}_2$
is honest, the challenger verifies proofs before performing the last mixing and executes shuffling and decryption if and only if verification outputs 1. Please note, that the challenger performs partial decryption always based on the content of the first Local Bulletin Board regardless of which one is presented to the Attacker. This is due to the fact that, in case the Attacker is presented with different results depending on the Local Bulletin Board shown, he could use this information in order to distinguish between bulletin boards (as explained in [10]). Still, for the system to be verifiable, a proof of the correct decryption has to be provided. An extra algorithm SimProof is needed to simulate the proof of correct decryption when the ballot box shown and the one decrypted are not the same.

The objective of the Attacker in this game is to be able to tell which of Local Bulletin Boards was presented to him. For simplicity, we describe a privacy game for a 1 out of n election (a voter may chose one option out of n). It can be easily generalized to elections with several candidates. The formalization of the game is as follows:

**Configuration:**

Challenger \((EL_{pk}, pk_{CCR_A}, K_A')\): Computes \((EL_{pk}, EL_{sk}) \leftarrow Gen_c(1^\lambda)\)

Computes \((pk_{SDM}, sk_{SDM}, c_{sk}, VCC_{pk}, VCC_{sk}) \leftarrow Config(1^\lambda)\)

Initializes the empty voter lists \(ID_h, ID_c, ID = (ID_h \cup ID_c)\)

Initializes two empty CCR_c’s logs: \(BB_{CCR_c}, BB_{CCR_c}'\)

Initializes an empty CCM_c’s log: \(BB_{CCM_c}\)

Computes \((pk_{CCR_c}, sk_{CCR_c}, k'_C, K'_C) \leftarrow Config.CCR_j(1^\lambda)\)

Computes \(pk_{CCR} = pk_{CCR_A} \cdot pk_{CCR_c}\)

Posts \((EL_{pk}, VCC_{pk}, pk_{CCR}, pk_{CCR_c}, pk_{CCR_A}, k'_C, K'_C)\) in \(BB_{CCR_c}, BB_{CCR_c}'\)

\(\beta \leftarrow \{0, 1\}\)

Keeps \((EL_{sk}, c_{sk}, VCC_{sk}, sk_{SDM}, EL_{sk}, sk_{CCR_c}, k'_C)\)

Shows \(BB_{CCR_c}\) to the Attacker

**Registration:** The Attacker can ask the Challenger to run the following algorithms several times:

O\text{Challenger_RegisterHonest}(id): If generated \(id \in ID\), stop and return \(⊥\).

Otherwise:

Add \(id\) to \(ID_h\)

Register\((1^\lambda, pk_{SDM}, sk_{SDM}, c_{sk}, VCC_{sk}, VCC_{sk})\)

Note, that Register requires an interaction with CCRs = (CCR_c, CCR_A), where CCR_c is controlled by the Challenger and honestly executes Register.CCR_j, while CCR_A is controlled by the Attacker and can execute Register.CCR_j or any other algorithm of its choice.

Runs Setup.Verify, if it returns \(⊥\) - abort.

Posts \((id, VCC_{id}, K_{id}, map_{id}, VCC_s_{id})\) in \(BB_{CCR_c}, BB_{CCR_c}'\)

Keeps \((SVK_{id}, BCR_{id}, VCC_{id}, \{v_i, CC_{id,i}\}_{i=1}^n)\)

O\text{Challenger_RegisterCorrupt}(id): If generated \(id \in ID\), stop and return \(⊥\). Otherwise:

42
Adds \( \text{id} \) to \( \text{ID}_c \).

Register\((1, \lambda, \text{pk}_{\text{SDM}}, \text{sk}_{\text{SDM}}, \text{Csk}, \text{VCCs}_{\text{pk}}, \text{VCCs}_{\text{sk}})\)  

Note, that Register requires an interaction with CCRs = \((\text{CCR}_{\text{C}}, \text{CCR}_{\text{A}})\), where CCR\(_{\text{C}}\) is controlled by the Challenger and honestly executes \text{Register}\(_{\text{CCR}_{\text{C}}}\), while CCR\(_{\text{A}}\) is controlled by the Attacker and can execute \text{Register}\(_{\text{CCR}_{\text{A}}}\) or any other algorithm of its choice.

Runs \text{Setup}.\text{Verify}, if it returns \(-\perp -\) abort.

Posts \((\text{id}, \text{VC}_{\text{id}}, \text{K}_{\text{id}}, \text{map}_{\text{id}}, \text{VCcks}_{\text{id}})\) in \(\text{BB}_{\text{CCR}_0}, \text{BB}_{\text{CCR}_1}\)

Provides \((\text{SVK}_{\text{id}}, \text{BCK}_{\text{id}}, \text{VCC}_{\text{id}}, \{\text{v}_i, \text{CC}_{\text{id}}\}_{i=1}^n)\) to the Attacker.

\textbf{Voting}: The Attacker generates votes running the \text{CreateVote} algorithm, or any other algorithm of its choice, and can ask the Challenger to run the following algorithms on behalf of honest voters and an honest CCR several times:

\textbf{OChallenger.Vote(id, v_0, v_1)}: Performs the following steps, stops in case of any error:
- Checks that \(v_0, v_1 \in \Omega\)
- Checks that \(\text{id} \in \text{ID}_h\)
- If for the honest voter \(\text{id}\), ballots have been already computed – abort
- Runs \(\text{k}_{\text{id}} \leftarrow \text{GetKey}(\text{SVK}_{\text{id}}, \text{VCcks}_{\text{id}})\)
- \(b_0 \leftarrow \text{CreateVote}(\text{pk}_{\text{CCR}}, \text{VC}_{\text{id}}, v_0, \text{K}_{\text{id}}, \text{k}_{\text{id}})\)
- \(b_1 \leftarrow \text{CreateVote}(\text{pk}_{\text{CCR}}, \text{VC}_{\text{id}}, v_1, \text{K}_{\text{id}}, \text{k}_{\text{id}})\)
- Checks that \(\text{ProcessVote}(\text{pk}_{\text{CCR}}, \text{VC}_{\text{id}}, b_0)\) outputs 1
- Checks that \(\text{ProcessVote}(\text{pk}_{\text{CCR}}, \text{VC}_{\text{id}}, b_1)\) outputs 1
- Keeps \((\text{id}, v_0, v_1, b_0, b_1)\)
- Provides \(b_\beta\) to the Attacker.

\textbf{OChallenger.computeCRC(id, b)}: Performs the following steps, stops in case of any error:
- Checks that \(\text{id} \in \text{ID}\)
- Computes \(((\text{E}_2)^{k^\ast_{\text{id}}}_{\text{C}}, (g^{-r^\prime_{\text{id}}})^{\text{sk}_{\text{CCR}_c}}_{\text{CCR}_c}) \leftarrow \text{CreateCC.ccr}_{\text{CCR}_c}(\text{pk}_{\text{CCR}_c}, \text{sk}_{\text{CCR}_c}, \text{VC}_{\text{id}}, b, k'_{\text{C}})\), if output is 0 then abort. Otherwise,
- If \(\text{id} \in \text{ID}_h\):
  - Recovers \(b_0, b_1\) that correspond to the honest voter \(\text{id}\)
  - If \(b \neq b_\beta\) abort
  - Adds \((\text{id}, b_1, \text{CRC Computed})\) in \(\text{BB}_{\text{CCR}_1}\)
  - Adds \((\text{id}, b_0, \text{CRC Computed})\) in \(\text{BB}_{\text{CCR}_0}\)
- If \(\text{id} \in \text{ID}_c\):
  - Adds \((\text{id}, b, \text{CRC Computed})\) in \(\text{BB}_{\text{CCR}_1}, \text{BB}_{\text{CCR}_0}\)

\text{Add} \((\text{id}, b, ((\text{E}_2)^{k^\ast_{\text{id}}}_{\text{C}}, (g^{-r^\prime_{\text{id}}})^{\text{sk}_{\text{CCR}_c}}_{\text{CCR}_c}))\) in \(\text{BB}_{\text{CCR}_0}\)

Provides the output of \text{CreateCC.ccr}_{\text{CCR}_c} to the Attacker.

\textbf{OChallenger.createCM(id, pC)}: Performs the following steps, stops in case of any error:
- Checks that \(\text{id} \in \text{ID}_h\)
AuditCodes(pC, CC\textsuperscript{id}), if output is 0, aborts. Otherwise, computes CM\textsuperscript{id} ← Confirm(VC\textsuperscript{id}, b_β, k_\text{id}, BCK\textsuperscript{id})
Provides CM\textsuperscript{id} to the Attacker.

O\textsuperscript{Challenger_computeVCC}(\text{id}, CM\textsuperscript{id}): Performs the following steps, stops in case of any error:
Computes (H(CM\textsuperscript{id}))\textsuperscript{2kC}\textsuperscript{id}
If \text{id} ∈ ID\textsubscript{h}:
- recovers b_0, b_1 that correspond to the honest voter \text{id}
  - if b ≠ b_β abort
  - adds (\text{id}, b_1, Confirmed) in BB\textsubscript{CCR}\textsuperscript{C}
  - adds (\text{id}, b_0, Confirmed) in BB\textsubscript{CCR}\textsuperscript{C}
If \text{id} ∈ ID\textsubscript{e}:
  - adds (\text{id}, b, Confirmed) in BB\textsubscript{CCR}\textsuperscript{C}, BB\textsubscript{CCR}\textsuperscript{C}
Adds (\text{id}, CM\textsuperscript{id}) in BB\textsubscript{CCR}\textsuperscript{C}
Provides (H(CM\textsuperscript{id}))\textsuperscript{2kC}\textsuperscript{id} to the Attacker.

Counting: The Attacker can corrupt only one of two CCMs. If he corrupts the first one, he executes mixing and partial decryption by running the MixDec\textsubscript{1} or any other algorithms of its choice, and then ask the Challenger to run MixDec\textsubscript{2}. Otherwise, he asks the Challenger to run MixDec\textsubscript{1} and then executes MixDec\textsubscript{2} or any other algorithms of its choice.

O\textsuperscript{Challenger_MixDec1}(BB, BB\textsuperscript{CCR}\textsubscript{A}, L = \{(c_1, c_2)\}): Does the following:

Adds information about the shuffle request in BB\textsuperscript{CCR}\textsubscript{C}
if β = 0:
(L\textsuperscript{*}, L_1, P\textsubscript{m}, P\textsubscript{dec}) ← MixDec\textsubscript{1}(EL\textsubscript{pk}, L)
If AuditorVerify(BB, BB\textsubscript{CCR}\textsuperscript{A}, BB\textsubscript{CCR}\textsuperscript{C}, BB\textsuperscript{CCR}\textsubscript{C}, L, L\textsuperscript{*}, L_1, P\textsubscript{m}, P\textsubscript{dec}) outputs 0, aborts
Otherwise, provides (L\textsuperscript{*}, L_1, P\textsubscript{m}, P\textsubscript{dec}) to the Attacker.

if β = 1:
Computes L_0 ← SimpleCleansing(BB\textsubscript{CCR}\textsuperscript{C})
Computes (L\textsuperscript{*}, P\textsubscript{m}) := Mix(EL\textsubscript{pk}, L)
Computes (\tilde{L}\textsuperscript{*}, \tilde{P}\textsubscript{m}) := Mix(EL\textsubscript{pk}, L_0)
From \tilde{L}\textsuperscript{*} it computes a partially decrypted list \tilde{L}_1 = \{(d_1, d_2)\} and a list of simulated zero-knowledge proofs P\textsubscript{dec} ← SimProof(L\textsuperscript{*}, \{d_2\}), where d_2 ∈ \tilde{L}_1
If AuditorVerify(BB, BB\textsuperscript{CCR}\textsubscript{A}, BB\textsubscript{CCR}\textsuperscript{C}, BB\textsuperscript{CCR}\textsubscript{C}, L, L\textsuperscript{*}, \tilde{L}_1, P\textsubscript{m}, P\textsubscript{dec}) outputs 0, aborts

44
Otherwise, provides \((L^*, \hat{L}_1, \hat{P}_n, \hat{P}_{\text{dec}})\) to the Attacker.

\[ \text{OChallenger}_\text{MixDec2}(\text{BB}, \text{BB}^{\text{CCR}_{A}}, \text{BB}^{\text{CCM}_{A}}, L = \{c_1, c_2\}, {L^*, \hat{L}_1, \hat{P}_n, \hat{P}_{\text{dec}}}) : \]

Does the following:

Adds information about shuffle request in \(\text{BB}^{\text{CCM}_{c}}\)

if \(\beta = 0:\)

If \(\text{AuditorVerify}(\text{BB}, \text{BB}^{\text{CCR}_{A}}, \text{BB}^{\text{CCR}_{C}}, \text{BB}^{\text{CCM}_{A}}, L = \{c_1, c_2\}, {L^*, \hat{L}_1, \hat{P}_n, \hat{P}_{\text{dec}}})\) outputs 0, aborts

\((L^*_1, L_v, \hat{P}_n, \hat{P}_{\text{dec}}) \leftarrow \text{MixDec}_2(\text{EL}_{\text{sk}}, L, L^*, L_1, \hat{P}_n, \hat{P}_{\text{dec}})\)

Provides \((L^*_1, L_v, \hat{P}_n, \hat{P}_{\text{dec}})\) to the Attacker.

if \(\beta = 1:\)

If \(\text{AuditorVerify}(\text{BB}, \text{BB}^{\text{CCR}_{A}}, \text{BB}^{\text{CCR}_{C}}, \text{BB}^{\text{CCM}_{A}}, L = \{c_1, c_2\}, {L^*, \hat{L}_1, \hat{P}_n, \hat{P}_{\text{dec}}})\) outputs 0, aborts

Computes \((L^*_1, \hat{P}_n) := \text{Mix}(\text{EL}_{\text{sk}}, L)\)

From \(L^*_1\) computes a decrypted list \(L_v\)

Based on the kept values \((\text{id}, v_0, v_1)\) obtains two lists \(L^*_v\) and \(L^1_v\) of provided by the Attacker choices for confirmed votes of honest voters.

For every \(v\) in \(L^1_v\) find an entry in \(L_v\) that matches \(v\) and substitute it with a corresponding option in \(L^*_v\).

- Compute a list of simulated zero-knowledge proofs \(\hat{P}_{\text{dec}} \leftarrow \text{SimProof}(L^*_1, L^*_v)\)

- Provide \((L^*_1, L_v, \hat{P}_n, \hat{P}_{\text{dec}})\) to the Attacker.

First of all, we proceed to define the implementation of the algorithm \(\text{SimProof}\), which is necessary in order to provide both privacy and verifiability:

\(\text{SimProof}(C'_1, M^*)\) takes as input an output of the honest mixing based on the \(\text{BB}_1\) and list of desired results \(M^*\) and provides a simulated NIZK proof \(\Pi'\) which is generated as follows:

- For each of the ciphertexts \(c'_i \in C'_1\) it computes the simulation of a decryption proof running the algorithm \(\text{SimDec}(c'_i, m^*_i)\) from the NIZKPK scheme where \(m^*_i\) is obtained from \(M^*\) which where obtained based on ballots in \(\text{BB}_0^{\text{CCR}_{c}}\).

Outputs \(\Pi'\) contains the list of simulated decryption proofs \(\Pi'_{\text{dec}}\).

Then we define the implementation of the algorithm \(\text{SimpleCleansing}\), which is used by the Challenger during the mixing and partial decryption process. Note, that while the voting protocol defines \(\text{Cleansing}\) algorithm, it can not be used by the Challenger as it requires verification of the retrieved short Vote Cast Return Code, which is computed by the Attacker.

\(\text{SimpleCleansing}(\text{BB})\) takes a ballot box as input and outputs the list of ciphertexts which is generated as follows:
Let this be the game where 5.2 Security Reductions control component in Vote Cast Return Code generation. Confirmed only when the Attacker is requesting the participation of the honest the Choice Return Code generation process. Similarly, the ballot is marked as component only when an Attacker is requesting the challenger to participate in malicious server would process this vote or not. The ballot is logged by the control internal database of the honest control component as it is unclear if a malicious lots on behalf of honest voting device. At this moment votes are not logged in algorithms from game D were substituted with SimMixDec, SimMixDec, SimDec decryption proofs, for which it runs the per- form exactly the same operations with the exception of the generation of the DecP property of the where \( \epsilon \) is the advantage of an adversary against the zero-knowledge protection

Please note, that upon calling the Vote function, challenger generates ballots on behalf of honest voting device. At this moment votes are not logged in the internal database of the honest control component as it is unclear if a malicious server would process this vote or not. The ballot is logged by the control component only when an Attacker is requesting the challenger to participate in the Choice Return Code generation process. Similarly, the ballot is marked as Confirmed only when the Attacker is requesting the participation of the honest control component in Vote Cast Return Code generation.

5.2 Security Reductions

GAME E.1 Let this be the game where \( \beta = 0 \) and the MixDec1, MixDec2 algorithms from game D were substituted with SimMixDec1, SimMixDec2, which performs exactly the same operations with the exception of the generation of the decryption proofs, for which it runs the SimDec algorithm from the NIZKPK scheme instead of ProveDec.

SimMixDec1 \((\text{El}_{\text{pk}}^{(1)}, L = \{(c_1, c_2)\})\) is run by CCM1 and proceeds as follows:

- from the ciphertext list \( L = \{(c_1, c_2)\} \) computes \((L^*, P_a) := \text{Mix(El}_{\text{pk}}, L)\), namely a shuffled list \( L^* = \{(\overline{c_1}, \overline{c_2})\}\text{mix} \) and a proof \( P_a \) of correct mixing

- From \( L^* \) it computes a partially decrypted list \( L_1 = \{(c_1^{(1)}, c_2^{(1)})\}\text{dec} \) and a list of zero-knowledge proofs of correct partial decryptions computed as \( P_{\text{dec}} \leftarrow \text{SimDec}(L^*, \{c_2^{(1)}\}) \) for each \( c_2^{(1)} \in L_1 \)

- output \( \left( L^* = \{(\overline{c_1}, \overline{c_2})\}\text{mix}, L_1 = \{(c_1^{(1)}, c_2^{(1)})\}\text{dec}, P_a, P_{\text{dec}} \right) \)

SimMixDec2 \((\text{El}_{\text{sk}}^{(2)}, L = \{(c_1, c_2)\}), L_1 = \{(c_1^{(1)}, c_2^{(1)})\}\text{mix}, P_a, P_{\text{dec}} \) is run by CCM2 and proceeds as follows:

- from the ciphertext list \( L_1 = \{(c_1^{(1)}, c_2^{(1)})\}\text{dec} \) computes \((L_1^*, P_a) := \text{Mix(El}_{\text{sk}}, \text{L}_1)\), namely a shuffled list \( L_1^* = \{(\tilde{c}_1, \tilde{c}_2)\}_{\text{mix}} \) and a proof \( P_a \) of correct mixing

- From \( L_1^* \) it computes a decrypted list \( L_v = \{V\}_{\text{dec}} \) and a list of zero-knowledge proofs of correct partial decryptions computed as \( P_{\text{dec}} \leftarrow \text{SimDec}(L_1^*, L_v) \)

- output \( \left( L_1^*, L_v, P_a, P_{\text{dec}} \right) \)

Define \( S_D^1 \) as the event that \( \hat{\beta} = \beta \). We claim that

\[
| \Pr\{S_D^1\} - \Pr\{S_D^1\} | = \epsilon_{zkdec},
\]

where \( \epsilon_{zkdec} \) is the advantage of an adversary against the zero-knowledge property of the DecP NIZKPK scheme, which is negligible.
GAME E.2 Let this be the game where the algorithm CreateVote for honest voters from the previous game is substituted with SimVote, where the same operations than in the original algorithm are executed, but \( \pi_{\text{exp}} \) and \( \pi_{\text{eqenc}} \) are computed by using the simulation algorithms SimEq and SimEqEnc from the NIZKPK scheme respectively.

SimVote \( \left( \{p_{\text{CCR}}^{(i)}\}_{i=1}^\psi, V_{\text{CID}}, \{v_i\}_{i=1}^\psi, K_{\text{id}}, k_{\text{id}} \right) \) takes as input the Verification Card ID \( V_{\text{CID}} \), a set of voting options selected by the voter \( \{v_1, \ldots, v_\psi\} \) and the Verification Card key pair \((K_{\text{id}}, k_{\text{id}})\), and does the following:

- Computes the aggregation of the voter’s selections: \( \nu = \prod_{j=1}^\psi v_j \)
- Encrypts the previous result: \( E_1 = (c_1, c_2) \leftarrow \text{Enc}(\nu, EL_{pk}; r) \)
- Generates \( \pi_{\text{sch}} \leftarrow \text{ProveExp}(g, c_1, c_2, r) \) a proof of knowledge of the encryption exponent, where \( r \) is the encryption randomness used to compute \( E_1 \), and \( V_{\text{CID}} \) is used as auxiliary information
- Computes partial Choice Return Codes as \( \{\text{pCC}_{\text{id}}^{v_j}\}_{j=1}^\psi = (v_1^{k_{\text{id}}}, \ldots, v_\psi^{k_{\text{id}}}) \)
- Computes an ElGamal multiple encryption of those codes as
  \[
  E_2 := \text{mEnc} \left( \left( p_{\text{CCR}}^{(1)}, \ldots, p_{\text{CCR}}^{(\psi)} \right), \left( \text{pCC}_{\text{id}}^{1}, \ldots, \text{pCC}_{\text{id}}^{\psi} \right) \right) = \left( g', \left( p_{\text{CCR}}^{(1)} \right)^{r'}, \text{pCC}_{\text{id}}^{1}, \ldots, \left( p_{\text{CCR}}^{(\psi)} \right)^{r'}, \text{pCC}_{\text{id}}^{\psi} \right)
  \]
- Computes \( \tilde{E}_1 := (\tilde{c}_1, \tilde{c}_2) = (c_1^{k_{\text{id}}}, c_2^{k_{\text{id}}}) \) and
  \( \tilde{E}_2 := (\tilde{c}_1, \tilde{c}_2) = (g', \left( p_{\text{CCR}}^{(1)} \right)^{r'}, \text{pCC}_{\text{id}}^{1}, \ldots, \left( p_{\text{CCR}}^{(\psi)} \right)^{r'}, \text{pCC}_{\text{id}}^{\psi} ) \)
- Generates two NIZKPK proofs to prove that the voting options in the ciphertext \( E_1 \) and the voting options used for encrypting the partial Choice Return Codes in \( E_2 \) are the same:
  \( \forall \pi_{\text{exp}} = \text{SimEq}(g, c_1, c_2, k_{\text{id}}, \tilde{c}_1, \tilde{c}_2) \) proves that \( \tilde{E}_1 \) is computed by raising the elements of \( E_1 \) to the private key \( k_{\text{id}} \) corresponding to the public key \( K_{\text{id}} \)
  \( \forall \pi_{\text{eqenc}} = \text{SimEqEnc}(g, EL_{pk}, \prod_{j=1}^\psi \mathbf{E}_1, \tilde{E}_2) \) which proves that \( \tilde{E}_1 \) is an encryption under the Election public key \( EL_{pk} \) of the product of partial Choice Return Codes \( \{\text{pCC}_{\text{id}}^{v_j}\}_{j=1}^\psi \) contained in \( \tilde{E}_2 \)

The output of this algorithm is the ballot \( b = (E_1, E_2, \tilde{E}_1, \tilde{E}_2, K_{\text{id}}, P) \), where \( P := (\pi_{\text{sch}}, \pi_{\text{exp}}, \pi_{\text{eqenc}}) \).

Define \( S_1^P \) as the event that \( \hat{\beta} = \beta \). Using the fact that nothing but proofs changed and output of SimEq and ProveEq is \((h, z)\), where \( h \) in both cases is under control of Random Oracle and \( z \) is either sampled randomly or computes as \( s + x \cdot h \) for a randomly sampled \( s \), we claim that outputs’ distribution is identical in the Random oracle model. Same logic can be also applied to SimEqEnc and ProveEqEnc. Therefore, we claim that

\[
|Pr\{S_1^P\} - Pr\{S_2^P\}| = 0
\]
GAME E.3 Let this be the game where the algorithms Registration, Confirm and SimVote for honest voters from the previous game are changed into RndRegistration, RndCreateVote and RndConfirm in which the partial choice return codes and vote cast codes are generated using a random oracle $O_{lc}$ (instead of exponentiating in power of CCR’s private key $k_{id}$ or $k_{c id}$). To support this change, the response of an honest CCR in ProcessConfirm for honest voters is also slightly modified.

RndRegistration($\lambda$, $pk_{sdm}$, $sk_{sdm}$, $C_{sdm}$, $VCC_{sk}$, $VCC_{pa}$, $\{RndRegister.CCR(\cdot)\}$) is an interactive algorithm run by Print office and requires participation of both CCRs. Please note, that only honest CCR runs $RndRegister.CCR$ in this game.

- generates a random Verification Card ID $VC_{id}$
- generates a Start Voting Key $SVK_{id} \leftarrow \mathcal{A}_{svk}$
- generates a keystore for a symmetric encryption key $KSkey_{id} \leftarrow \text{PBKDF2}(SVK_{id}, KEYseed)$
- samples two random elements ($g_{id}^1, g_{id}^2$) and sets ($K_{id}, k_{id}$) = ($g_{id}^1, g_{id}^2$)
- computes the encryption of the Verification Card private key with the keystore encryption key: $VC_{ks_{id}} \leftarrow \text{Enc}(k_{id}; KSkey_{id})$
- chooses at random a Ballot Casting Key $BCK_{id} \leftarrow \mathcal{A}_{bck}$
- assigns short Choice Return Codes $CC_{id} \leftarrow \mathcal{A}_{cc}$ at random for voter $id$, $\forall i = 1, \ldots, n$
- assigns a short Vote Cast Return Code $VCC_{id} \leftarrow \mathcal{A}_{vcc}$ at random for voter $id$
- Computes the signature value of the Vote Cast Return Code $SVCC_{id} \leftarrow \text{Sign}(VCC_{id}, VCC_{sk})$

Next, the Print office initiates the computation of a series of Long Choice Return Codes ($pC_{id}^1, \ldots, pC_{id}^\psi$) in interaction with CCR1, CCR2:

- encrypts under $pk_{sdm}$ the set of voting options \{O_{lc}(v_{id}^1), \ldots, O_{lc}(v_{id}^{\psi})\} as

\[
(\text{ctv}_{id}^1, \ldots, \text{ctv}_{id}^\psi) := (\text{Enc}(pk_{sdm}, O_{lc}(v_{id}^1)), \ldots, \text{Enc}(pk_{sdm}, O_{lc}(v_{id}^{\psi})))
\]

- computes $hbck_{id} = H[O_{lc}([BCK_{id}^{k_{id}}])^{k_{id}}]^2$ and sets $ctbck_{id} := \text{Enc}(pk_{sdm}, hbck_{id})$

Next, the Print office sends initCCR to each CCR, and obtains back init_{SDM}^i, where where

\[
\text{init}_{SDM}^1 := (k_{j}'', \{pk_{CCR}^{(i)}\}_{j=1}^\psi \cdot \{VC_{id}, Kc_{id}, K_{id}, \{(ctv_{id}^1)^{k_{id}}\}_{i=1}^n, (ctbck_{id})^{k_{id}}\})
\]

for $i = 1, 2$. On inputs $init_{SDM}^1, init_{SDM}^2$, the print office computes a list of Choice Return Codes Public Encryption Keys $pk^{(i)}_{CCR} := pk^{(i)}_{CCR_1}, pk^{(i)}_{CCR_2}$ for $i = 1, \ldots, \psi$.\

\footnote{It is assumed that no two voters $id \neq id'$ have an identical $VC_{id} = VC_{id'}$}
Next, the print office computes for each Verification Card identifier \( VC_{id} \) the following:

- encryptions \( \text{ctpc}_{id}^{1} \) of pre-Choice Return codes \( \{ \text{pcC}_{id}^{1}, \ldots, \text{pcC}_{id}^{n} \} \) as
  \( \text{ctpc}_{id}^{1} := (\text{ctv}_{id}^{1})^{k_{id}}, (\text{ctv}_{id}^{1})^{k_{id}} \) for \( i = 1, \ldots, n \)
- computes an encryption of the pre-Vote Cast Return Code as
  \[
  \text{ctv}_{id}^{1} : = \left(\text{ctbck}_{id}^{1}\right)^{k_{id}} \cdot \left(\text{ctbck}_{id}^{1}\right)^{k_{id}}
  \]
- pre-Choice Return Codes are decrypted as
  \[
  \left(\text{pcC}_{id}^{1}, \ldots, \text{pcC}_{id}^{n}\right) := \left(\text{Dec}(\text{sk}_{\text{SDM}}, \text{ctpc}_{id}^{1}), \ldots, \text{Dec}(\text{sk}_{\text{SDM}}, \text{ctpc}_{id}^{n})\right)
  \]
- the pre-Vote Cast Return Code is decrypted as
  \[
  \text{pVCC}_{id} := \text{Dec}(\text{sk}_{\text{SDM}}, \text{ctv}_{id}^{1})
  \]
- computes \( \text{hlCC}_{id}^{1} = \text{H}(\text{VCC}_{id}^{1}) \) for each \( i = 1, \ldots, n \)
- computes \( \text{hlVCC}_{id}^{1} = \text{H}(\text{VCC}_{id}^{1}) \)
- the mapping between long and short codes is defined as
  \[
  \text{map}_{id}^{1} = \left(\left\{\text{H}(\text{VCC}_{id}^{1}), \text{Enc}^{s}(\text{CC}_{id}^{1}; \text{skCC}_{id}^{1})\right\}\right)_{i = 1, \ldots, n}
  \]
where \( \text{SVCC}_{id} := \text{Sign}(\text{VCC}_{sk}, \text{VCC}_{id}^{1}) \) is the validity proof for the short vote Cast Code \( \text{VCC}_{id}^{1} \)

Finally the print office outputs

\[
\left( \text{id}, \text{SVK}_{id}, \text{VC}_{id}, \text{K}_{id}, \text{VCK}_{id}, \text{BCK}_{id}, \text{VCC}_{id}, \{v_{i}, \text{CC}_{id}^{1}\}_{j = 1, \ldots, n}, \text{map}_{id}^{1} \right)
\]

1. **RndRegister.CCR.** \( \text{initCCR}, \psi \left\{ \text{pk}_{\text{CCR}}, \text{sk}_{\text{CCR}} \right\}_{i = 1} \psi \), where

\[
\text{initCCR} = \left(\psi, \text{pk}_{\text{SDM}}, \left\{ \left(\text{id}, \{\text{ctv}_{id}^{1}, \ldots, \text{ctv}_{id}^{n}\}, \text{ctbck}_{id}^{1}\right) \right\}_{i = 1} \right)
\]
is input provided by the print office and \( \{\text{pk}_{\text{CCR}}, \text{sk}_{\text{CCR}}\}_{i = 1} \psi \) are Control Components Choice Return Codes key pairs and \( \{\text{pk}_{\text{CCR}}, \text{sk}_{\text{CCR}}\}_{i = 1} \psi \) are Control Components Choice Return Codes key pairs. Next, for every \( \text{VC}_{id} \) does the following:

- samples two random group elements \( (g_{id}^{1}, g_{id}^{4}, g_{id}^{5}, g_{id}^{6}) \) and sets \( (k_{id}^{1}, k_{id}^{1}) := (g_{id}^{1}, g_{id}^{4}) \)
- sets \( \{\text{ctv}_{id}^{1}, \ldots, \text{ctv}_{id}^{n}\} := \{\text{ctv}_{id}^{1}, \ldots, \text{ctv}_{id}^{n}\} \)
- sets \( \text{ctv}_{id}^{1} \cdot \text{ctv}_{id}^{1} = \text{O}_{\text{LC}} \left(\text{ctv}_{id}^{1}\right)^{k_{id}} \cdot \ldots \cdot \left(\text{ctv}_{id}^{1}\right)^{k_{id}} = \text{O}_{\text{LC}} \left(\text{ctv}_{id}^{1}\right)^{k_{id}} \cdot \ldots \cdot \left(\text{ctv}_{id}^{1}\right)^{k_{id}} \)
- sets \( \text{ctbck}_{id}^{1} \cdot \text{ctbck}_{id}^{1} = \text{O}_{\text{LC}} \left(\text{ctbck}_{id}^{1}\right)^{k_{id}} \cdot \ldots \cdot \left(\text{ctbck}_{id}^{1}\right)^{k_{id}} \)

49
- sets \( \text{init}_i^{\text{SDM}} := \left( K_i', \left\{ p_{\text{CCR}}^{(i)} \right\}_{i=1}^\psi, \left\{ \left( \text{VC}_{1d}, K_{c1d}, K_{1d} \right), \left( \left( \text{ctv}_{1d} \right)^{K_{1d}} \right)_{i=1}^\nu, \left( \text{ctbck}_{1d} \right)^{\text{xc}_{1d}} \right\} \right) \),
  
  and outputs \( \text{init}_i^{\text{SDM}} \) and key pair \( (K_i', K_i) \)

\( \text{RndCreateVote} \left( \left\{ p_{\text{CCR}}^{(i)} \right\}_{i=1}^\psi, \text{VC}_{1d}, \{ v_i \}_{i=1}^\psi, K_{1d}, k_{1d} \right) \) takes as input the Verification Card ID \( \text{VC}_{1d} \), a set of voting options selected by the voter \( \{ v_1, \ldots, v_\psi \} \) and the Verification Card key pair \( (K_{1d}, k_{1d}) \), and does the following:

- Computes the aggregation of the voter’s selections: \( \nu = \prod_{i=1}^\psi v_i \)
- Encrypts the previous result: \( E_1 = \left( c_1, c_2 \right) \leftarrow \text{Enc}(\nu, \text{EL}_p; r) \)
- Generates \( \pi_{\text{sch}} \leftarrow \text{ProveExp}((g, c_1, c_2), r) \) a proof of knowledge of the  
  encryption exponent, where \( r \) is the encryption randomness used to compute \( E_1 \), and \( \text{VC}_{1d} \) is used as auxiliary information
- Computes partial Choice Return Codes as \( \{ p_{\text{CC} \text{i}d} \}_{i=1}^\psi = (O_{i_c}(v_1^{c_{1d}}), \ldots, O_{i_c}(v_\psi^{c_{1d}})) \)
- Computes an ElGamal multiple encryption of those codes as
  \[
  E_2 := \text{mEnc} \left( \left\{ p_{\text{CC} \text{i}d}^{(i)} \right\}_{i=1}^\psi = (O_{i_c}(v_1^{c_{1d}}), \ldots, O_{i_c}(v_\psi^{c_{1d}})) \right) = \left( g^{c_1}, (p_{\text{CC} \text{i}d}^{(1)})^{c_1}, \ldots, (p_{\text{CC} \text{i}d}^{(\psi)})^{c_1} \right)
  \]
- Computes \( \tilde{E}_1 := (\tilde{c}_1, \tilde{c}_2) = (c_1^{\psi}, c_2^{\psi}) \) and
  \[
  \tilde{E}_2 := (\tilde{c}_1, \tilde{c}_2) = \left( g^{\psi}, \left( p_{\text{CC} \text{i}d}^{(1)} \right)^{\psi}, \ldots, \left( p_{\text{CC} \text{i}d}^{(\psi)} \right)^{\psi} \right)
  \]
- Generates two NIZKPK proofs to prove that the voting options in the ciphertext \( E_1 \) and the voting options used for encrypting the partial Choice Return Codes in \( E_2 \) are the same:
  \[
  \circ \pi_{\text{exp}} = \text{SimEq}(g, c_1, c_2, K_{1d}, \tilde{c}_1, \tilde{c}_2) \text{ proves that } \tilde{E}_1 \text{ is computed by raising the elements of } E_1 \text{ to the private key } K_{1d} \text{ corresponding to the public key } K_{1d}
  \]
  \[
  \circ \pi_{\text{eqenc}} = \text{SimEqEnc}(g, \text{EL}_{pk}, \prod_{E=1}^{\psi}, \tilde{E}_1, \tilde{E}_2) \text{ which proves that } \tilde{E}_1 \text{ is an encryption under the Election public key } \text{EL}_{pk} \text{ of the product of partial Choice Return Codes } \{ p_{\text{CC} \text{i}d} \}_{i=1}^\psi \text{ contained in } \tilde{E}_2
  \]

The output of this algorithm is the ballot \( b = (E_1, E_2, \tilde{E}_1, \tilde{E}_2, K_{1d}, P) \), where \( P := (\pi_{\text{sch}}, \pi_{\text{exp}}, \pi_{\text{eqenc}}) \).

\( \text{RndConfirm}(\text{VC}_{1d}, b, K_{1d}, \text{BCK}_{1d}) \) It receives as input a verification card id \( \text{VC}_{1d} \),
  
a ballot \( b \), the verification card private key \( K_{1d} \) and the voter’s ballot casting key \( \text{BCK}_{1d} \),
  
  and outputs the confirmation message \( \text{CM}_{1d} = O_{i_c}([\text{BCK}_{1d}]^{K_{1d}})^2 \).

Thus, in a \( \text{ProcessConfirm}(bb, \text{VC}_{1d}, \text{CM}_{1d}, C_{ak}, \text{VCC}_{pk}) \), an honest \( \text{CCR}_j \) computes response as \( \text{hcm}^{j} = O_{i_c} \left( \text{H} \left( \text{CM}^{j} \right) \cdot \text{xc}_{1d} \right)^2 \)
This hop can be unfolded in three hops:

Δ First, please notice that RndRegistration sends to each control component 
(Enc(pk_{SDM}, O_{lc}(v_{1}^{k_{id}})), \ldots, Enc(pk_{SDM}, O_{lc}(v_{n}^{k_{id}}))) while Registration sends 
(Enc(pk_{SDM}, v_{1}), \ldots, Enc(pk_{SDM}, v_{n})). We claim that those messages are indis-
tinguishable. Observe, that \( v_{i} \in G \) and O_{lc} outputs elements in G, therefor the 
message space is identical. Also recall definition of non-malleability – Definition 1. The indistinguishability of such change is based on the NM-CPA property of the ElGamal scheme combined with the proof of knowledge of the 
encryption exponent (this combination is also known as Signed ElGamal) that has been shown to be NM-CPA secure in [9].

Definition 1 The indistinguishability-based definition of non-malleability (NM-CPA) is represented in the following game:

- The Challenger runs \((pk_{e}, sk_{e}) \leftarrow Gen_{e}(1^{\lambda})\) and gives \(pk_{e}\) to the adversary.
- The adversary chooses two messages \(m_0\) and \(m_1\) and passes them to the Challenger, who randomly selects \(\beta \leftarrow \{0, 1\}\) and returns the encryption \(c^{*}\) of \(m_{\beta}\).
- The adversary may send a vector \(C\) of ciphertexts. For each \(c_{i} \in C\), if \(c_{i} = c^{*}\) then the Challenger returns \(\bot\), otherwise it returns \(Dec(c_{i}, sk_{e})\)
- The adversary submits a guess \(\beta'\) for \(\beta\).

Δ Second, during registration, voter’s Verification Card public key \(g^{k_{id}}\) is substi-
tuted with a randomly sampled value \(g^{k_{id}}\), and a tuple in the honest CCR’s reply 
\((g^{v_{1}^{k_{id}}}, g^{v_{n}^{k_{id}}}, \{O_{lc}(v_{1}^{k_{id}})\}_{i=1}^{n}, \{O_{lc}(v_{n}^{k_{id}})\}_{i=1}^{n})\) is substituted with random and inde-
pendent elements \((g^{id}, g^{id}, \{O_{lc}(v_{1}^{k_{id}})\}_{i=1}^{n}, \{O_{lc}(v_{n}^{k_{id}})\}_{i=1}^{n})\). Also, 
during vote confirmation, \((BC^{k_{id}})^{2k_{id}}\) is changed into a random element \(O_{lc}([BC^{k_{id}}]^{2k_{id}})^{2}\).

The problem of distinguishing such tuples is a variant of DDH problem called L-DDH as it is proven in [22]. However notice, that an Attacker who knows \(k_{id}\) can easily distinguish between an \(K_{id} = g^{k_{id}}\) and a randomly sampled element \(g^{id}\). Since the probability of breaking symmetrically encrypted key store is neg-
ligible, the best strategy for an attacker would be to guess the SVK_{id} key, thus the probability of obtaining \(k_{id}\) is \(\frac{1}{|A|_{svk}}\).

Δ Third, during registration and voting phases, \((v_{1}^{k_{id}}, \ldots, v_{n}^{k_{id}})\) are substituted with \((O_{lc}(v_{1}^{k_{id}}), \ldots, O_{lc}(v_{n}^{k_{id}}))\), where \(\{v_{1}, \ldots, v_{n}\}\) are chosen as small bit-length primes belonging to the subgroup G defined for the ElGamal encryption scheme. The advantage of an efficient distinguisher algorithm for solving Decision Diffie-
Hellman problems in case of small primes is negligible considering that the exponentiation function of small primes in \(G\) can be considered a pseudo-random function. According to [22] the best known way to solve DDH problem in such case is computing discrete logarithms.
Define $S^D_3$ as the event that $\hat{\beta} = \beta$. Based on discussions above, we claim that

$$|\Pr\{S^D_2\} - \Pr\{S^D_3\}| = \epsilon_{\text{nm-cpa}} + \epsilon_{l\text{-ddh}} + \frac{1}{|A_{svk}|} + \epsilon_{dl}$$

where $\epsilon_{\text{nm-cpa}}$ is the advantage of an efficient adversary for the NM-CPA property of the encryption scheme, $\epsilon_{l\text{-ddh}}$ is the adversarial advantage for solving L-DDH problem, $\frac{1}{|A_{svk}|}$ is the adversarial advantage for breaking encrypted Verification Card key store, and $\epsilon_{dl}$ is the adversarial advantage for solving discrete-log problem.

**Game E.4** Let this be the game where the algorithm $\text{RndCreateVote}$ from the previous game is changed into $\text{RndSimCreateVote}$, where the values $\tilde{c}$ are randomly chosen from a uniform distribution in $G$ (the mathematical group for the encryption scheme).

$\text{RndSimCreateVote}\left(\left\{\text{pk}^{(i)}_{\text{CCR}}\right\}_{i=1}^{\psi}, \text{VC}_{id}, \left\{v_i\right\}_{i=1}^{\psi}, K_{id}, k_{id}\right)$ takes as input the Verification Card ID $\text{VC}_{id}$, a set of voting options selected by the voter $\left\{v_1, \ldots, v_\psi\right\}$ and the Verification Card key pair $(K_{id}, k_{id})$, and does the following:

- Computes the aggregation of the voter’s selections: $\nu = \prod_{i=1}^{\psi} v_i$
- Encrypts the previous result: $E1 = (c_1, c_2) \leftarrow \text{Enc}(\nu, \text{EL}_\text{pk}; r)$
- Generates $\pi_{\text{sch}} \leftarrow \text{ProveExp}(\left(\text{g, c}_1, \text{c}_2\right), r)$ a proof of knowledge of the encryption exponent, where $r$ is the encryption randomness used to compute $E1$, and $\text{VC}_{id}$ is used as auxiliary information
- Computes partial Choice Return Codes as $\left\{\text{pCC}_{id}\right\}_{i=1}^{\psi} = (O_{lc}(v_{iK_{id}}^1), \ldots, O_{lc}(v_{iK_{id}}^\psi))$
- Computes an ElGamal multiple encryption of those codes as

$$\text{E2} := \text{mEnc}\left((\text{pk}_{\text{CCR}}^{(1)}, \ldots, \text{pk}_{\text{CCR}}^{(\psi)}), (\text{pCC}_{id}^{(1)}, \ldots, \text{pCC}_{id}^{(\psi)})\right) = \left((\text{g}', (\text{pk}_{\text{CCR}}^{(1)})^{\nu'} \cdot \text{pCC}_{id}^{(1)}, \ldots, (\text{pk}_{\text{CCR}}^{(\psi)})^{\nu'} \cdot \text{pCC}_{id}^{(\psi)})\right)$$

- Computes $\tilde{E1} := (\tilde{c}_1, \tilde{c}_2) \leftarrow \text{G}$
- Computes $\text{pk}_{\text{CCR}} := \prod_{i=1}^{\psi} \text{pk}_{\text{CCR}}^{(i)}$
- Generates two NIZKPK proofs to prove that the voting options in the ciphertext $E1$ and the voting options used for encrypting the partial Choice Return Codes in $E2$ are the same:

$\diamond \pi_{\text{exp}} = \text{SimEq}(g, c_1, c_2, K_{id}, \tilde{c}_1, \tilde{c}_2)$ proves that $\tilde{E1}$ is computed by raising the elements of $E1$ to the private key $k_{id}$ corresponding to the public key $K_{id}$

$\diamond \pi_{\text{eqenc}} = \text{SimEqEnc}(g, \text{EL}_\text{pk}, \text{pk}_{\text{CCR}}, \tilde{E1}, \tilde{E2})$ which proves that $\tilde{E1}$ is an encryption under the Election public key $\text{EL}_\text{pk}$ of the product of partial Choice Return Codes $\left\{\text{pCC}_{id}\right\}_{i=1}^{\psi}$ contained in $E2$
The output of this algorithm is the ballot \( b = (E_1, E_2, \overrightarrow{E_1}, \overrightarrow{E_2}, K_{id}, P) \), where 

\[ P := (\pi_{sch}, \pi_{exp}, \pi_{eqenc}). \]

Define \( S^P_i \) as the event that \( \hat{\beta} = \beta \). We claim that

\[ |\Pr\{S^P_i\} - \Pr\{S^P_{i-1}\}| = \epsilon_{l-ddh}. \]

Note that at the end of this game sequence, the only values that depend on the voting options in the ballots created by the Challenger when the **Challenger** CreateVote \((id, v_0, v_1)\) algorithm is run are \( c, \pi_{exp} \). Now we are ready to evolve the games by gradually changing the ballots from \( BB_0 \) into those from \( BB_1 \), as in [15].

The indistinguishability of the games described below is based on the NM-CPA property of the Pedersen’s distributed key generation applied to ElGamal combined with the proof of knowledge of the encryption exponent. In [15] it has been proven that a ciphertext \( C \leftarrow \text{Enc}(m, EL_{pk}) \) where \( EL_{pk} \leftarrow \text{SetupElKey}(EL_A_{pk}, EL_C_{pk}) \), can be seen as a ciphertext of a fully distributed NM-CPA scheme.

**Game D.4.** Each game is obtained from game \( D.4_{i-1} \) by letting the Challenger do one of the following operations over the ballot of honest voters \( 1 \ldots i \) during the execution of the **Challenger** CreateVote \((id, v_0, v_1)\) algorithm:

For \( j = 1 \ldots i \):
- If \( b_0^j = b_1^j \), then do nothing.
- If \( b_0^j \neq b_1^j \), then replace \( b_0^j \) by \( b_1^j \).

Define \( S^P_i \) as the event that \( \hat{\beta} = \beta \). We claim that

\[ |\Pr\{S^P_i\} - \Pr\{S^P_{4_{i-1}}\}| = \epsilon_{nm-cpa}, \]

where \( \epsilon_{nm-cpa} \) is the advantage of an efficient adversary for the NM-CPA property of the fully distributed encryption scheme.

**GAME E.5** Finally we can rename the last game \( D.4_i \) into game D.5, where the view of the adversary is equivalent to the situation where \( \beta = 1 \).

Define \( S^P_i \) as the event that \( \hat{\beta} = \beta \). We can see that the only information provided to the Attacker which depends on different voting options per bulletin board is the ciphertext \( c \), which has no benefit in the Attacker decision. Therefore the best chance for the Attacker is to guess at random and therefore \( \Pr\{S^P_i\} = \frac{1}{2} \). Finally,

\[ \text{PRIVadv}[A] = \epsilon_{akdec} + \epsilon_{nm-cpa} + \epsilon_{l-ddh} + \frac{1}{|A_{svk}|} \left( \epsilon_{dl} + \epsilon_{l-ddh} + i \cdot \epsilon_{nm-cpa} \right) \]

which is negligible given the properties of the NIZKPK schemes, pseudorandom functions and NM-CPA encryption schemes used in the protocol.
6 Conclusions

In this paper, we have presented the Swiss electronic voting protocol to be used in the Swiss Post Online Voting platform, and specifically, we have focused on the ballot privacy property. Our analysis has confirmed that ballot privacy property of sVote holds in our interpretation of the threat model for complete verifiability.

However, it needs to be pointed out that our adversarial model does not account for trivial cases of privacy breaches such as when all honest voters vote for the same options, election authorities run the decryption procedure before results from other voting channels are obtained or if the adversary controls user platform. We emphasize that no voting channel electronic or not can preserve the privacy of honest voters if they all chose identical options. Similarly, we are not considering as 'provisional' results known by authorities before results from other channels are acquired as it is solely a procedural issue. Also, we highlight that no system that relays on client-side encryption can satisfy ballot privacy property in cases when an attacker controls user platform that is used for encryption the ballot as he instantly learns voter’s preferences.

Another thing worth mentioning is the importance of auditing cleansing and mixing procedures before performing the last decryption (or executing the last control component). This verification ensures the result would be checked before decryption and if any manipulation attempt detected privacy of individual votes would not be affected. It is important to stress that auditing intermediate results before executing last CCM is the requirement only due to privacy concerns, as an attacker may attempt to send re-encryptions of a particular ballot instead of cleansing output in order to break privacy of a specific voter. Even though it is arguable that ballot privacy is broken in scenarios when re-election is required, we prevent any interpretation ambiguity by demanding verification. However, if it can be ensured that the last CCM does not leak provisional results, this check can be omitted.

References


vol. 7237, pp. 263–280. Springer (2012), http://dx.doi.org/10.1007/978-3-642-29011-4_17


[33] Scytl R&S: PRO SP RS Audit Protocol Control Components v3.1 (November 2018)
Scytl Standard Software  All intellectual property rights in the Scytl Standard Software are Scytl's sole property. Scytl owns and shall retain all rights, title and interest in and to the Scytl Standard Software. Scytl Standard Software is licensed to Swiss Post under the terms and conditions described in the Framework Agreement.

Swiss Post-Scytl Software  All intellectual property rights in the Swiss Post-Scytl Software are the joint property of Scytl and Swiss Post (Joint IP).

EV Solution  All intellectual property rights in the EV Solution other than Joint IP will be owned by Scytl or by third parties as applicable.