Table of contents

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

2. Appendix ....................................................................................................................... 5

1.1 EV Solution Intellectual Property Rights Notice (the Notice) ........................................ 5

1.1.1 Definitions .................................................................................................................. 5

1.1.2 Copyright notice ........................................................................................................ 6

2 Annex: “sVote with Control Components Voting Protocol - Computational Proof of Complete Verifiability” ............... 6
1 Summary

In 2013 the Federal Chancellery published a new regulation for the authorization of Internet voting systems (VEleS) 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).

For the 50% electorate level, the voting system needs to provide cryptographic and symbolic proofs to demonstrate that the implemented cryptographic protocol provides 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 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 been 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 complete verifiability 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
2. Appendix

1.1 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 30\textsuperscript{th}, 2015.

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

1.1.1 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.

1.1.2 Copyright notice

1.1.2.1 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.

1.1.2.2 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).

1.1.2.3 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.

2 Annex: “sVote with Control Components Voting Protocol - Computational Proof of Complete Verifiability”
sVote with Control Components Voting Protocol - Computational Proof of Complete Verifiability

David Galindo, Scytl R&S

November 23, 2018

Abstract

This document describes sVote with Control Components voting protocol and assess its complete verifiability under the trust assumptions and threat model defined in [13]. The extended capabilities that an adversary has in the latter model are counterbalanced by the fact that an attack is successful only if it goes undetected by voters during the voting phase or by auditors in a post-election phase. sVote verifiability properties are defined and a computational cryptographic analysis is provided.

Keywords: electronic voting, cast-as-intended verifiability universal verifiability, complete verifiability, malicious voting client, choice return codes, malicious voting server, control components
## Contents

1 **Introduction** .................................................. 4  
1.1 Our contribution ............................................. 6  
1.2 Proving the security properties of the protocol ............. 6  

2 **Threat model and security goals** .............................. 7  
2.1 Security Assumptions and Threat Model ........................ 7  
2.1.1 Assumptions on parties ..................................... 7  
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 ........................................... 9  
2.2.1 Protocol participants in sVote ............................... 9  
2.2.2 Trust assumptions in sVote .................................. 10  
2.3 Correspondence between both security models ................. 11  

3 **Building blocks** ............................................... 11  
3.1 Encryption schemes ............................................ 12  
3.2 Key derivation functions ...................................... 13  
3.3 Representation of the voting options ........................... 13  
3.4 Pseudo-random functions ...................................... 13  
3.5 Signature schemes ............................................. 14  
3.6 Non-interactive zero-knowledge proofs of knowledge ........ 14  
3.7 Verifiable mixnet ............................................... 17  

4 **Protocol** ...................................................... 17  
4.1 Abstractions and relation to the base protocol ............... 18  
4.1.1 System setup ............................................... 18  
4.1.2 Election setup and voting phase ............................ 20  
4.1.3 Mapping content to the base protocol ....................... 22  
4.2 Codes value spaces ............................................ 22  
4.3 Setup phase algorithms ........................................ 24  
4.3.1 SetupElKey .................................................. 24  
4.3.2 Setup ....................................................... 24  
4.4 Voting phase algorithms ...................................... 27  
4.4.1 GetKey ........................................................ 28  
4.4.2 CreateVote .................................................. 28  
4.4.3 ProcessVote .................................................. 29  
4.4.4 CreateCL ..................................................... 29  
4.4.5 CreateRC ..................................................... 30  
4.4.6 AuditCodes ................................................... 31  
4.4.7 Confirm ...................................................... 31  
4.4.8 ProcessConfirm .............................................. 31  
4.5 Tally phase algorithms ....................................... 32  
4.5.1 Cleansing .................................................... 32  
4.5.2 MixDec ........................................................ 32  
4.5.3 MixDec ........................................................ 33  
4.6 Verify phase algorithm ....................................... 33  
4.6.1 VerifyTally ................................................... 33  
4.6.2 AuditorVerify ................................................ 34
5 Provable secure complete verifiability

5.1 Type 1 attacks (vote tampering) .............................................. 36
5.2 Type 2 attack (voting rejection) .............................................. 37
5.3 Type 3 attack (vote cast) ....................................................... 37
5.4 Type 4 attack (incorrect tally) ................................................. 37
5.5 Active adversaries in sVote ..................................................... 37
5.6 Adversary admissibility ......................................................... 37

6 Security analysis

6.1 Game A: vote tampering ......................................................... 41
6.2 Game B: vote rejection .......................................................... 44
   6.2.1 Security reduction of game B ........................................... 45
6.3 Game C: vote cast ............................................................... 49
   6.3.1 Security reduction of game C ........................................... 51
6.4 Game D: incorrect tally ......................................................... 53
   6.4.1 Security reduction of game D ........................................... 55

7 Conclusions

7.1 Final remarks and improvements ........................................... 56
7.2 VEieS security objectives ..................................................... 57
   7.2.1 Individual verifiability .................................................... 57
   7.2.2 Universal verifiability .................................................... 60
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 complete verifiability, 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 \cite{22, 23, 30}, 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 \cite{2}, 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.

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:** In Section 5 we define four security properties. In Section 6 we show the scheme of Section 4 enjoys these properties. Last, in
Section 3, the security objectives for complete verifiability defined in [13] are reviewed and mapped against our security properties.

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 complete verifiability 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 are provided in a different 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).

## 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 sVote 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 sVote 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 and complete verifiability</td>
<td>4.1</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>4.3</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

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>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
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.”

2.2 Trust model in sVote

2.2.1 Protocol participants in sVote

The protocol specification [18] 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 Printing service 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.

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 anticipée sans corrompre les électeurs ou leurs plates-formes utilisateurs respectives.”
- **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 **Printing service**), 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.

- **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, in 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 **Printing service** (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 **Printing service** and voters) is considered to be trustworthy.

- **Printing service** is trusted.

---

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

3 Building blocks

Our abstraction of the sVote voting protocol uses the following building blocks.
Table 4: Correspondence between the sVote and the Chancellery assumptions made on the communication channels

<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</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>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>

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

3.1 Encryption schemes

Public key encryption scheme. Formally, a public key encryption scheme is defined by the algorithms \( (\text{Gen}_e, \text{Enc}, \text{Dec}) \): the key generation algorithm \( \text{Gen}_e \) receives as input a security parameter \( 1^\lambda \) and outputs a key pair composed by a public key \( pk_e \) and a private key \( sk_e \), defines a message space \( \mathcal{M}_{sp} \), a ciphertext space \( \mathcal{C}_{sp} \) and a randomness space \( \mathcal{R}_{sp} \) (in case of a probabilistic encryption scheme); the encryption algorithm \( \text{Enc} \) takes as input a message \( m \in \mathcal{M}_{sp} \) and a public key \( pk_e \), and computes a ciphertext \( c \in \mathcal{C}_{sp} \). In case the algorithm is probabilistic, it uses random values \( r \in \mathcal{R}_{sp} \) for computing such ciphertext; the decryption algorithm \( \text{Dec} \) receives as input a ciphertext \( c \in \mathcal{C}_{sp} \) and a private key \( sk_e \), and outputs a message \( m \in \mathcal{M}_{sp} \) or \( \perp \) in case of error.

Our protocol uses the ElGamal encryption scheme \[20]: The key generation algorithm \( \text{Gen}_e \) 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 \( \text{Enc} \) 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 \( c = (c_1, c_2) = (g^r, pk_e \cdot m) \). The decryption algorithm \( \text{Dec} \) receives \( c \) and the private key \( sk_e \), and outputs a ciphertext \( m \in \mathcal{M}_{sp} \) or \( \perp \) in case of error.

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

sVote applies a secure $KDF$ function to the input key before feeding that key into the AES encryption scheme in GCM mode \[29\] 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 $c \leftarrow \text{Enc}^s(m; k)$, $t_a \leftarrow \text{ghash}(c, a)$, 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 $\text{Dec}^s(c; k)$ 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 $\delta$ 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 $\text{PBKDF2}$ 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 $\text{PBKDF2}$ 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 SHA2-256 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 $\mathcal{G}$ 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) ([6,8,31]), 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 \( \phi(n) \) \( \phi(n) = (p-1)(q-1) \). The private key \( sk_s \) takes the value of \( d \), where \( ed \equiv 1 \pmod{\phi(n)} \). \( \text{Sign} \) takes as input a message \( m \), which is not restricted to a specific space, and the private key \( sk_s \), and outputs \( \psi = (\text{ME}(m))^d \pmod{n} \), where \( \text{ME} \) denotes a transformation with random padding over \( H_s(m) \) and \( H_s \) denotes a hash function which maps strings to elements in \( \mathbb{Z}_n \). \( \text{Verify} \) takes as input the public key \( pk_s \), the message \( m \) and the signature \( \psi \), and checks that \( \text{ME}(m) = \psi^e \pmod{n} \). 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 \( crs \). \( \text{NIZKProve} \) is the proof generation algorithm. It receives as input a common reference string \( crs \), 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 $(a^z)^h \cdot t = a^z$.

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 (given that the output of the hash is shorter than the output of the function $\phi$). 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 $h' = H(a, a^x, (a^z)^{-h} \cdot a^z, aux)$ 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 [12] which we denote as $\text{EqDL}$:

- \text{ProveEq}((a_1, a_2, \ldots, a_n, a_1^2, a_2^2, \ldots, a_n^2), x)$ takes at random a value $s$ from $\mathbb{Z}_q$, computes $a_1^s, a_2^s, \ldots, a_n^s, h = H(a_1, a_2, \ldots, a_n, a_1^2, a_2^2, \ldots, a_n^2, a_1^2, a_2^2, \ldots, a_n^2)$ 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 $(h, z)$.

- \text{VerifyEq}((a_1, a_2, \ldots, a_n, a_1^2, a_2^2, \ldots, a_n^2), \pi_{\text{eqDL}}) computes $a_1^{t'} = a_1^z \cdot (a_1^2)^{-h}$, $a_2^{t'} = a_2^z (a_2^2)^{-h}, \ldots, a_n^{t'} = a_n^z (a_n^2)^{-h}$, and checks that $h = H(a_1, a_2, \ldots, a_n, a_1^s, a_2^s \ldots, a_n^s, a_1^2, a_2^2 \ldots, a_n^2)$. If the validation is successful, the algorithm outputs 1. Otherwise it outputs 0.

\footnote{Our NIZKPK schemes particularly do not use a common reference string, and therefore $\text{GenCRS}$ is not executed.}
- \text{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 \( H(a_1, a_2, \ldots, a_n, a_1', a_2', \ldots, a_n') \) the oracle returns the value \( h^* \).

\textbf{Knowledge of encryption exponent.} Based on the Schnorr identification protocol \([33]\), it is used for proving knowledge of the encryption exponent of the \textsc{ElGamal} ciphertext \( c \). We denote it as \textsc{ExpP}, and its construction is similar to the particular case of the \textsc{NIZK} proof of equality of discrete logarithms where \( n = 1 \). It additionally makes use of some auxiliary information \( \text{aux} \) that is known both to prover and verifier. Therefore:

- \text{ProveExp}((g, c_1, c_2), 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 \( \pi_{\text{sch}} = (h, z) \).
- \text{VerifyExp}((g, c_1, c_2), \pi_{\text{sch}}) computes \( g^{z^*} = g^z \cdot (c_1^{-h}) \) and checks that \( h = H(\text{aux}, g, c_1, c_2, g^{z^*}) \). If the validation is successful, the algorithm outputs 1. Otherwise it outputs 0.

The combination of \textsc{ElGamal} encryption together with a proof of knowledge of the encryption exponent is known as \textsc{Signed ElGamal}, and has been shown to be \textsc{NM-CPA} secure in \([32]\).

\textbf{Proof of equality of encryptions.} The proof system \textsc{EqEnc} consists of three algorithms (\textsc{ProveEqEnc}, \textsc{VerifyEqEnc}, \textsc{SimEqEnc}) and it is used to prove that \( c \) and \( \tilde{c} \) are \textsc{ElGamal} encryptions of the same plaintext under public keys \((g, h_1)\) and \((g, h_2)\) respectively. They are defined as follows:

- \text{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 \( g^{s_1}, g^{s_2}, h_1^{s_1}(\frac{1}{h_2})^{s_2} \), \( h = H(c_1, \tilde{c}_1, \tilde{c}_2, g^{s_1}, g^{s_2}, h_1^{s_1}(\frac{1}{h_2})^{s_2}) \) 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 \( \pi_{\text{eqEnc}} \) is \((h, z_1, z_2)\).
- \text{VerifyEqEnc}(g, h_1, h_2, c = (c_1, c_2), \tilde{c} = (\tilde{c}_1, \tilde{c}_2), \pi_{\text{eqEnc}}) \) computes \( g^{z^*_1} = g^{z_1} \cdot c_1^{-h}, g^{z^*_2} = g^{z_2} \cdot (\tilde{c}_1)^{-h}, h_1^{z_1}(\frac{1}{h_2})^{z_2} = h_1^{z_1} \cdot (\frac{1}{h_2})^{z_2} \cdot (\tilde{c}_2)^{-h} \), and checks that \( h = H(c_1, \tilde{c}_1, \tilde{c}_2, g^{z^*_1}, g^{z^*_2}, h_1^{z_1}(\frac{1}{h_2})^{z_2}) \). If the validation is successful, the algorithm outputs 1. Otherwise it outputs 0.
- \text{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 \( H(c_1, \tilde{c}_1, \tilde{c}_2, g^{z_1^*}, g^{z_2^*}, h_1^{z_1^*}(\frac{1}{h_2})^{z_2^*}) \) the oracle returns the value \( h^* \).
Correct decryption. Proofs of correct decryption are based on the Chaum-Pedersen protocol. However, we use a different notation than in EqDL for simplicity in the protocol description. We denote them as DecP and describe the following algorithms:

- **ProveDec**(\((c,m),sk_e\)) receives a ciphertext \(c = (c_1,c_2)\) and a witness \(sk_e\), where \(c_1 = g^r\) and \(c_2 = pk_e^r \cdot m\), being \(pk_e = g^{sk_e}\). It takes at random \(s\) from \(Z_q\), computes \((g^r)^s, g^s, h = H(c,m,(g^r)^s, g^s)\) and \(z = s + sk_e \cdot h\). The proof is \(\pi_{dec} = (h,z)\).

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

- **SimDec**(\(c,m^*\)) takes at random \(z^*\) and \(h^*\) from \(Z_q\) 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 \(H(c,m^*,(g^r)^{s'}, g^{s'})\) the oracle returns the value \(h^*\).

The properties of NIZKPKs are completeness, soundness and zero-knowledge. 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 \(\pi_{mix}\) 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 \(\pi_{mix}\), 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. 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 \(\mathcal{ID}\) and a counting function \(\rho\) consists of the tuple \(\mathcal{V} = (\text{Setup}, \text{Vote}, \text{Tally}, \text{Verify})\) in\(^1\). A voting scheme, called sVote, is defined relative to the multiset function \(\mathcal{V}\), 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 ???. 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 $\text{bb}$, the transcript of the $j$-th choice return control component $\text{CCR}_j$ denoted with $\text{bb}^{\text{CCR}}_j$, and the transcript of the $j$-th mixing control component $\text{CCM}_j$ denoted with $\text{bb}^{\text{CCM}}_j$. Throughout this document, we sometimes refer to $\text{bb}^{\text{CCR}}_j$ and $\text{bb}^{\text{CCM}}_j$ 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 $\text{CCR}_j$ 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 CCRs and two CCMs, 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’ 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.

---

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.
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>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>Last CCM is corrupted, but can’t reconstruct its key.</td>
</tr>
<tr>
<td>Case 2</td>
<td>Last CCM is corrupted and can reconstruct its key.</td>
</tr>
<tr>
<td>Case 3</td>
<td>Last CCM is honest, but can’t reconstruct its key.</td>
</tr>
<tr>
<td>Case 4</td>
<td>Last CCM is honest and knows its key.</td>
</tr>
</tbody>
</table>

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 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 \( EL_{pk}^{(2)} \) 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 \( EL_{pk}^{(2)} \) 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 \( \rho : (\Omega \cup \{\perp\})^* \to R \) 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 [25] 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) printed in the voting card.
In the protocol model we assume, that all encrypted Verification Card Key-store 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<sub>id</sub>) that is printed in the voting card is private.

4.1.3 Mapping content to the base protocol

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

<table>
<thead>
<tr>
<th>This document</th>
<th>Algorithm</th>
<th>External reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 4.3.1</td>
<td>SetupEKey_{SDM}, SetupEKey_{CCM}</td>
<td>[18], Section 4.5</td>
</tr>
<tr>
<td>Section 4.3.2</td>
<td>Setup_{SDM}, Setup_{CCR}</td>
<td>[18], Section 4.6</td>
</tr>
<tr>
<td>Section 4.4.1</td>
<td>GetKey</td>
<td>[18], Section 5.3</td>
</tr>
<tr>
<td>Section 4.4.2</td>
<td>CreateVote</td>
<td>[18], Section 5.4.1</td>
</tr>
<tr>
<td>Section 4.4.3</td>
<td>ProcessVote</td>
<td>[18], Section 5.4.2</td>
</tr>
<tr>
<td>Section 4.4.4</td>
<td>CreateCC.&lt;sub&gt;serv&lt;/sub&gt;, CreateCC.&lt;sub&gt;ccr&lt;/sub&gt;</td>
<td>[18], Section 5.4.3</td>
</tr>
<tr>
<td>Section 4.4.5</td>
<td>CreateRC</td>
<td>[18], Section 5.4.4</td>
</tr>
<tr>
<td>Section 4.4.6</td>
<td>AuditCodes</td>
<td>None (voter checks using her voting card)</td>
</tr>
<tr>
<td>Section 4.4.7</td>
<td>Confirm</td>
<td>[18], Section 5.6.1</td>
</tr>
<tr>
<td>Section 4.4.8</td>
<td>ProcessConfirm</td>
<td>[18], Section 5.6.2</td>
</tr>
<tr>
<td>Section 4.5.1</td>
<td>Cleansing</td>
<td>[18], Section 6.1.1</td>
</tr>
<tr>
<td>Section 4.5.2</td>
<td>MixDec&lt;sub&gt;1&lt;/sub&gt;</td>
<td>[18], Sections 6.1.2.1, 6.1.2.2 and 6.1.2.3</td>
</tr>
<tr>
<td>Section 4.5.3</td>
<td>MixDec&lt;sub&gt;2&lt;/sub&gt;</td>
<td>[18], Section 6.1.2.5</td>
</tr>
<tr>
<td>Section 4.6.1</td>
<td>VerifyTally</td>
<td>[18], Section 6.1.2.4</td>
</tr>
<tr>
<td>Section 4.6.2</td>
<td>AuditorVerify</td>
<td>[18], Section 7</td>
</tr>
</tbody>
</table>

Table 6: Correspondence between algorithms described in this document and their equivalents in other documents [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<sub>svk</sub> consists of 32<sup>20</sup> values.
<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. Shared among CCMs.</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>VOC_{pk}</td>
<td>Vote Cast Return Code Signer public key</td>
<td>Public</td>
</tr>
<tr>
<td>VOC_{sk}</td>
<td>Vote Cast Return Code Signer private key</td>
<td></td>
</tr>
<tr>
<td>K_{id}^{(j)}</td>
<td>CCR_{j}'s Voter choice return code generation public key</td>
<td>Public</td>
</tr>
<tr>
<td>k_{id}^{(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>Kc_{id}^{(j)}</td>
<td>CCR_{j}'s Voter vote cast code generation public key</td>
<td>Public</td>
</tr>
<tr>
<td>kc_{id}^{(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'_{j}</td>
<td>CCR_{j}'s Choice return code generation public key</td>
<td>Public</td>
</tr>
<tr>
<td>k'_{j}</td>
<td>CCR_{j}'s Choice return code generation private key</td>
<td>Private. Known to CCR_{j} only.</td>
</tr>
<tr>
<td>skcc_{i,d}</td>
<td>i-th symmetric key to encrypt i-th choice return code for voter i.d.</td>
<td>Private. Known to the Voting Server. (The voting server does not know the index i).</td>
</tr>
<tr>
<td>skvcc_{i,d}</td>
<td>Symmetric key to encrypt vote cast code for voter i.d.</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_{i,d}</td>
<td>Voter’s Ballot Casting Key</td>
<td>Private. Known to the voter only.</td>
</tr>
<tr>
<td>SVK_{i,d}</td>
<td>Voter’s Start Voting Key</td>
<td>Private. Known to the voter only.</td>
</tr>
<tr>
<td>VCK_{i,d}</td>
<td>Keystore key to store SVK_{i,d}</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 [113]*
Ballot casting keys. These codes are 8-digit numbers. The space $\mathcal{A}_{ck}$ consists of $10^8$ values.

Long choice and vote cast return codes. These codes are sometimes seen as ElGamal plaintexts. The space $\mathcal{A}_{lc}$ 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 $\mathcal{A}_{sc}$ consists of $10^4$ values.

Short vote cast return codes. These codes are 8-digit numbers. The space $\mathcal{A}_{vcc}$ consists of $10^8$ values.

4.3 Setup phase algorithms

The setup phase consists on the following algorithms.

4.3.1 SetupElKey($1^\lambda$)

It is an interactive algorithm run by the mixing control components CCM$_1$, CCM$_2$ and the printing service. They proceed as follows.

1. SetupEKey.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 $(\text{EL}_{pk}^{(j)}, \text{EL}_{sk}^{(j)}) \leftarrow \text{Gen}_e(1^\lambda)$. It keeps $\text{EL}_{sk}^{(j)}$ secret and sends $\text{EL}_{pk}^{(j)}$ to the printing service.

2. SetupEKey.SDM($\text{EL}_{pk}^{(1)}, \text{EL}_{pk}^{(2)}$) is run by the printing service. It takes the election public key shares $\text{EL}_{pk}^{(1)}, \text{EL}_{pk}^{(2)}$ created by CCM$_1$, CCM$_2$ respectively, computes the Election Public Key as $\text{EL}_{pk} := \text{EL}_{pk}^{(1)} \cdot \text{EL}_{pk}^{(2)}$.

4.3.2 Setup($1^\lambda, ID$)

It is an interactive algorithm run by the printing service 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.SDM($ID$) it is executed in the printing service.

   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 ($\text{VCC}_{pk}, \text{VCC}_{sk}$) $\leftarrow \text{Gen}_d(1^\lambda)$.

For each voter $\text{id} \in ID$, where $ID$ denotes a list of voter pseudonyms, the printing service generates cryptographic keys, identifiers and short codes:
- generates a random Verification Card ID \( VC_{id} \)
- generates a Start Voting Key \( SVK_{id} \leftarrow \mathcal{A}_{svk} \)
- generates a keystore symmetric encryption key \( KS\text{key}_{id} \leftarrow \text{PBKDF2}(SVK_{id},\text{KEYseed}) \)
- generates the Verification Card key pair: \( (k_{id}, k_{id}) \leftarrow \text{Gen}(1^\lambda) \)
- computes the encryption of the Verification Card private key with the keystore encryption key: \( VC\text{ks}_{id} \leftarrow \text{Enc}(k_{id}; KS\text{key}_{id}) \)
- 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 \( SVCC_{id} \leftarrow \text{Sign}(VCC_{id}, VCCs_{sk}) \)

Next, the printing service engages with CCR1 and CCR2 in the computation of the pre-choice return codes \( (pCC_{id}^1, \ldots, pCC_{id}^n) \) and the pre-vote cast return code \( pVCC_{id} \) for each voter id \( \in \mathcal{I} \). The interaction consists of two rounds. In the first round the printing service does the following:

- encrypts under the Secure Data Manager public key \( pk_{SDM} \) the set of voting options \( \{v_1, \ldots, v_n\} \) as
  \[
  (ctv_{id}^1, \ldots, ctv_{id}^n) := (\text{Enc}(pk_{SDM}, v_1), \ldots, \text{Enc}(pk_{SDM}, v_n))
  \]
- computes the confirmation message \( CCh_{id} = (BCK_{id})^{2k_{id}} \) and a hash of it \( hbck_{id} = (H(CCh_{id}))^2 \) and sets \( ctbck_{id} := \text{Enc}(pk_{SDM}, hbck_{id}) \)
- set \( \text{init}_{CCR} = \left( \psi, pk_{SDM}, \{ (VC_{id}, k_{id}, K_{id}, (ctv_{id})_{k_{id}}^{\psi}, (ctbck_{id})_{k_{id}}^{\psi}) \}^{id \in \mathcal{I}} \right) \)
  - broadcast \( \text{init}_{CCR} \) to the control components CCR\( j \) for \( j = 1, 2 \)

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

- on reception of \( \text{init}_{SDM}^{j} \) from CCR\( j \), parse it as:
  \[
  \left( K_{j}^\psi, \left\{ pk_{CCR}^{(i)} \right\}_{i=1}^{\psi}, \left\{ \left( \text{VC}_{id}, Kc_{id}, K_{id}, (ctv_{id}^{(i)})_{K_{id}}^{(i)}, (ctbck_{id}^{(i)})_{K_{id}}^{(i)} \right) \right\}_{i=1}^{n} \right)_{id \in \mathcal{I}}
  \]
- computes the choice return codes public encryption key
  \[
  pk_{CCR}^{(i)} := pk_{CCR}^{(i)} \cdot pk_{CCR}^{(i)}
  \]
  where the CCR\( j \)'s choice return codes encryption public keys \( pk_{CCR}^{(i)} \) is extracted from \( \text{init}_{SDM}^{j} \)

\footnote{It is assumed that no two voters \( id \neq id' \) have an identical \( VC_{id} = VC_{id'} \).}
Setup

2. Please note, that entries in the mapping table are shuffled to avoid trivial correlation.

\[ \delta \]

Finally, the printing service outputs:

- computes encryptions \(c_{tpc}^{id}\) of pre-Choice Return codes \((p_{C1}^{id}, \ldots, p_{Cn}^{id})\) by exponentiating with the verification card private key:

\[ c_{tpc}^{id} := \left( (ctv_i^{id})^{k_{id}}, (ctv_i^{id})^{k_{id}} \right)^{k_{id}} \]

for \(i = 1, \ldots, n\)

- computes an encryption of the pre-Vote Cast Return Code as

\[ c_{tpvcc}^{id} := \left( ctbck^{id})^{k_{cc}}, (ctbck^{id})^{k_{cc}} \right)^{k_{cc}} \]

- pre-Choice Return Codes are decrypted as

\( (p_{C1}^{id}, \ldots, p_{Cn}^{id}) := \{ \text{Dec}(sk_{SDK}, c_{tpc}^{id}), \ldots, \text{Dec}(sk_{SDK}, c_{tpc}^{id}) \} \)

- the pre-Vote Cast Return Code is decrypted as \(p_{VCC}^{id} := \text{Dec}(sk_{SDK}, c_{tpvcc}^{id})\)

Last, the printing service generates the long choice return codes \((1C_{C1}^{id}, \ldots, 1C_{Cn}^{id})\) and long vote cast code, \(1V_{C_{C1}^{id}}\) for each \(id \in ID\) and the mapping table as follows:

- computes \(1C_{C1}^{id} = H(p_{C1}^{id}||VC_{id}) \forall i = 1, \ldots, n\)

- computes \(h_{CC}^{id} = H(1C_{C1}^{id}||C_{sk})\) and sets \(sk_{CC}^{id} = \delta(h_{CC}^{id}), \forall i = 1, \ldots, n\)

- computes \(1V_{C_{C1}^{id}} = H(p_{VCC}^{id}||VC_{id})\)

- computes \(h_{VC_{C1}^{id}} = H(1V_{C_{C1}^{id}}||C_{sk})\) and sets \(sk_{VC_{C1}^{id}} = \delta(h_{VC_{C1}^{id}})\)

- the mapping between long and short codes is defined as

\[ \text{map}^{id} = \left( \left\{ [H(1C_{C1}^{id}), \text{Enc}^s(CC_{id}^{id}; sk_{CC}^{id})] \right\}_{i=1}^n, [H(1V_{C_{C1}^{id}}), \text{Enc}^s(VCC_{id}||S_{VCC_{id}}; sk_{VC_{C1}^{id}})] \right) \]

where \(S_{VCC_{id}} := \text{Sign}(VCC_{sk}, SV_{CC_{id}})\) is the validity proof for the short code \(SV_{CC_{id}}\)

- sets \(CM_{table} := \{ \text{map}^{id} \}_{id \in ID} \)

Finally, the printing service outputs:

\( \{ SV_{id}, VC_{id}, K_{id}, VCCs_{id}, BCK_{id}, VCC_{id}, \{ v_i, CC_{id}\}_{i=1}^n \}_{id \in ID}, CM_{table} \)

and \( (pk_{SDK}, sk_{SDK}), (VCCs_{sk}, VCCs_{sk}), C_{sk} \).

2. Setup.CCR\(_i\)(init\(_{CCR}_j\)) is executed in CCR\(_j\) for \(j = 1, 2\).

CCR\(_j\) starts computing key pairs \((pk_{CCR_j}^{i(j)}, sk_{CCR_j}^{i(j)}) \leftarrow \text{Gen}_{\lambda}(1^\lambda)\) for \(i = 1, \ldots, \psi\) and parses init\(_{CCR}_j\) as

\[ \text{init}_{CCR} = \left( \psi, pk_{SDK}, \{ (VC_{id}, (ctv_i^{id}, \ldots, ctv_n^{id}), ctbck^{id}) \}_{id \in ID} \right) \]

and generates the key pair \((k_j', k_{j}' = g^{k_j'})\), where \(k_j'\) is randomly generated and used in the key derivation function \(\delta(\cdot)\) (see footnote 7).

\footnote{\(\delta\) stands for the mask generation function defined in ISO-18033-2 and in PKCS#1v2.2.}

\footnote{Please note, that entries in the mapping table are shuffled to avoid trivial correlation.}
Next, for every verification card ID \( VC_{id} \) does the following:

- derives the Voter Choice Return Code Generation private key \( k_{id}^j := \delta(VC_{id}||k'_j) \) and the corresponding public key \( K_{id}^j := (g)^{k_{id}^j} \)
- derives the Voter Vote Cast Return Code Generation private key \( k_{c_{id}}^j := \delta(VC_{id}||\text{ConfirmStr}||k'_j) \) and the corresponding public key \( K_{c_{id}}^j := (g)^{k_{c_{id}}^j} \)
- computes \( n \) different ciphertexts \( ctv_{id}^i := (ctv_{id}^i)^{k_{id}^j} \) for \( i = 1, \ldots, n \).
- computes \( n \) different NIZK proofs, where the \( i \)-th proof

\[
\pi_{\text{exp}}^{(i)} := \text{ProveEq}((g, ctv_{id}^i, K_{id}^j, ctv_{id}^i, k_{id}^j))
\]

proves that \( ctv_{id}^i \) is computed by raising the elements of \( ctv_{id}^i \) 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.

- computes ciphertext \( ctbck_{id}^i := (ctbck_{id}^i)^{k_{c_{id}}^j} \)
- computes the following NIZK proof

\[
\pi_{\text{exp}} := \text{ProveEq}((g, ctbck_{id}^i, K_{c_{id}}^j, ctbck_{id}^i, k_{c_{id}}^j))
\]

proves that \( ctbck_{id}^i \) is computed by raising the elements of \( ctbck_{id}^i \) to the secret key \( k_{c_{id}}^j \) corresponding to the public key \( K_{c_{id}}^j \). The proof is logged to be validated during the audit phase.

Last, sets init_{SDM}^j :=

\[
(k'_j, \left\{p_{\text{CCR}_i}^{(i)}\right\}_{i=1}^\psi, \left\{(VC_{id}, K_{c_{id}}^j, K_{id}^j, \{\left(ctv_{id}^i\right)^{k_{id}^j}\}_{i=1}^n, \{ctbck_{id}^i\}^{k_{c_{id}}^j}\}_{id \in ID}\right\})
\]

And

\[
\Pi^j = (\pi_{\text{exp}}^{(i)}, \pi_{\text{exp}})
\]

and outputs init_{SDM}^j and key pairs \( \left\{p_{\text{CCR}_i}^{(i)}, s_{\text{CCR}_i}^{(i)}\right\}_{i=1}^\psi \) and \((k'_j, k'_j)\)

3. **Setup.**\( \text{Verify}((\text{init}_{SDM}^j, \Pi^j)^{12}_{j=1},) \) is executed by auditors to verify the computation performed by CCRs. It takes as input all CCRs' responses \( \text{init}_{SDM}^j \) and NIZKs of correct exponentiation \( \Pi^j \).

If for every \( VC_{id}, j = 1, 2 \) and \( i = 1, \ldots, n \) both verifications \( \text{VerifyExp}(g, ctv_{id}^i, K_{id}^j, ctbck_{id}^i) \) and \( \text{VerifyExp}(g, ctbck_{id}^i, K_{c_{id}}^j, ctbck_{id}^i) \) hold, the algorithms outputs 1. Otherwise it returns \( \bot \).

This ends the description of **Setup**(\( 1^\lambda, ID \)).

### 4.4 Voting phase algorithms

The voting phase consists of the following algorithms.
4.4.1 GetKey(SVK_{id}, VCks_{id})

On input of the start voting key SVK_{id} and the verification card keystore VCks_{id}, it does the following actions:

- Generates the keystore encryption symmetric key \( KSkey_{id} \leftarrow PBKDF2(SVK_{id}, KEYseed) \).
- Runs the Dec^3 algorithm with inputs VCks_{id} and KSkey_{id}, and recovers the verification card private key \( k_{id} \).

It returns the verification card private key \( k_{id} \).

4.4.2 CreateVote \( (EL_{pk}, \{pk_{CCR}^{(i)}_{id}\}_{i=1}^{\psi}, VC_{id}, \{v_i\}_{i=1}^{\psi}, K_{id}, k_{id}) \)

It takes as input the election public key \( EL_{pk} \), the choice return codes encryption public keys \( pk_{CCR} = \{pk_{CCR}^{(i)}\}_{i=1}^{\psi} \), the verification card ID \( VC_{id} \), a set of voting options selected by the voter \( \{v_1, \ldots, v_\psi\} \) 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: \( E1 = (c_1, c_2) \leftarrow Enc(\nu, EL_{pk}; r) \)
- Generates \( \pi_{sch} \leftarrow ProveExp((g, E1), r) \) a schnorr proof (knowledge of the exponent \( r \) of the first component of \( E1 \)). Recall \( r \) is the encryption randomness used to compute \( E1 \), and \( VC_{id} \) is used as auxiliary information
- Computes partial choice return codes as \( \{pCC_{id}^{\psi}\}_{i=1}^{\psi} = (v_1^{k_{id}}, \ldots, v_\psi^{k_{id}}) \)
- Computes an ElGamal multiple encryption of those codes as
  \[
  E2 := mEnc \left( (pk_{CCR}^{(1)}_{id}, \ldots, pk_{CCR}^{(\psi)}_{id}), (pCC_{id}^{1}, \ldots, pCC_{id}^{\psi}) \right) = \left( g^r, (pk_{CCR}^{(1)}_{id})^{r'}, (pCC_{id}^{1})^{r'}, \ldots, (pk_{CCR}^{(\psi)}_{id})^{r'}, (pCC_{id}^{\psi})^{r'} \right)
  \]
- Computes \( \tilde{E}1 := (\tilde{c}_1, \tilde{c}_2) = (c_1^{k_{id}}, c_2^{k_{id}}) \)
- Computes \( \tilde{E}2 := (\tilde{c}_1, \tilde{c}_2) = (g^r, (pk_{CCR}^{(1)}_{id})^{r'}, (pCC_{id}^{1})^{r'}, \ldots, (pk_{CCR}^{(\psi)}_{id})^{r'}, (pCC_{id}^{\psi})^{r'}) \)
- Generates two NIZK 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:
  - \( \pi_{exp} = \text{ProveEq}(g, E1, K_{id}, \tilde{E}1, k_{id}) \) proves that \( \tilde{E}1 \) is computed by raising the elements of \( E1 \) to the verification card private key \( k_{id} \) corresponding to the verification card public key \( K_{id} \)
  - \( \pi_{pleqenc} = \text{ProveEqEnc}(g, EL_{pk}, \prod_{i=1}^{\psi} pk_{CCR}^{(i)}_{id}, \tilde{E}1, \tilde{E}2, r \cdot k_{id}, r') \) which proves that \( \tilde{E}1 \) is an encryption under the election public key \( EL_{pk} \) of the product of partial Choice Return Codes \( \{pCC_{id}^{i}\}_{i=1}^{\psi} \) contained in \( E2 \)

The output of this algorithm is the ballot \( b = (E1, E2, \tilde{E}1, \tilde{E}2, K_{id}, P) \), where \( P := (\pi_{sch}, \pi_{exp}, \pi_{pleqenc}) \).
4.4.3 ProcessVote \( (EL_{pk}, pk_{CCR}^{(1)}, \ldots, pk_{CCR}^{(v)}, VC_{id}, b) \)

It receives as input the election public key \( EL_{pk} \), the Choice Return Codes Encryption public keys \( pk_{CCR}^{(1)}, \ldots, pk_{CCR}^{(v)} \), a verification card id \( VC_{id} \) and a ballot \( b = (E1, E2, E1, E2, K_{id}, P) \). It proceeds to validate the NIZK proofs \( \pi_{sch}, \pi_{exp}, \pi_{eqenc} \) from the ballot \( b \) running:

- VerifyExp((\( g, E1 \)), \( \pi_{sch} \))
- VerifyEq((\( g, E1, K_{id}, E1, \pi_{exp} \))
- VerifyEqEnc((\( g, EL_{pk}, pk_{CCR}, E1, E2 \)), \( \pi_{eqenc} \)) where \( pk_{CCR} := \prod_{i=1}^{v} pk_{CCR}^{(i)} \)

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\( (pk_{CCR}^{(1)}, \ldots, pk_{CCR}^{(v)}, VC_{id}, b) \), CreateCC.serv(\( \cdot \), \{CreateCC.crr(\( \cdot \))\}_{i=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.serv\( (pk_{CCR}^{(1)}, \ldots, pk_{CCR}^{(v)}, VC_{id}, b) \)

   - If ProcessVote\( (EL_{pk}, pk_{CCR}^{(1)}, \ldots, pk_{CCR}^{(v)}, VC_{id}, b) \) outputs 1, sends \( (VC_{id}, b) \) to \( CCR_1 \) and \( CCR_2 \). Otherwise abort. (Recall that \( EL_{pk} \) is the election public key, and \( pk_{CCR}^{(i)} \) the \( i \)-th element of the Choice Return Codes Encryption Key used to encrypt the \( i \)-th voting option\(^10\)

   - On reception of \( (E2)_{id}^{j} \) from \( CCR_j \) for \( j = 1, 2 \), the voting server computes \( E2_{CC} := (E2)_{id}^{j} \cdot (E2)_{id}^{2} \)

   - Parse \( E2_{CC} = (g^{r^j \cdot k}) \cdot (pk_{CCR}^{(1)})^{r^j \cdot k} \cdot PC_{id}^{1} \cdot \ldots \cdot (pk_{CCR}^{(v)})^{r^j \cdot k} \cdot PC_{id}^{v} \) where \( k := \sum_{j=1}^{2} K_{id} \cdot k \). Send \( g^{r^j \cdot k} \) to \( CCR_1 \) and \( CCR_2 \)

   - On reception of \( (g^{r^j \cdot k})^{pk_{CCR,j}^{(1)}} \), \( \ldots \), \( (g^{r^j \cdot k})^{pk_{CCR,j}^{(v)}} \) from \( CCR_j \) for \( j = 1, 2 \), the voting server computes partial decryptions

\[
p_{dec} := g^{-r^j \cdot k \cdot sk_{CC,1}^{(i)}} \cdot g^{-r^j \cdot k \cdot sk_{CC,2}^{(i)}} = (pk_{CCR}^{(i)})^{-r^j \cdot k}
\]

   for \( i = 1, \ldots, v \)

   - Next for \( i = 1, \ldots, v \) compute pre-choice return codes as \( PC_{id}^{j+1} := (E2_{CC})_{id}^{j+1} \cdot p_{dec} \) where \( (E2_{CC})_{id}^{j+1} \) is the \( (i+1) \)-th component of \( E2_{CC} \)

   - Output \( PC_{id}^{1}, \ldots, PC_{id}^{v} \)

\(^9\)For our analysis we have merged the Election Context, the Voting Workflow Context and the Vote Verification Context from \( [13] \) 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.
2. CreateCC.ccr \( \left( \{p_{CCr, i}^{(i)}, sk_{CCr, i}^{(i)} \}_{i=1}^{\psi}, \mathcal{VC}_{id}, b, k'_j \right) \) takes as input a Verification Card ID \( \mathcal{VC}_{id} \) and a ballot \( b \), sent by the Voting Server, as well the CCR\(_j\)'s Choice Return Codes Generation private key \( k'_j \). The return code control component maintains a list \( L_{ccr} \) with the verification card ids that have been already queried (the list is initialized to the empty list). It proceeds as follows:

- If \( \mathcal{VC}_{id} \in L_{ccr} \), abort. Else, set \( L_{ccr} \leftarrow L_{ccr} \cup \{\mathcal{VC}_{id}\} \)

- If ProcessVote(\( pk_{CCr}^{(i)}, \ldots, pk_{CCr}^{(\psi)}, \mathcal{VC}_{id}, b \)) outputs 0 then abort. Otherwise, let \( b = (E_1, E_2, E_1, E_2, K_{ld}, \hat{P}) \). Then compute \( E_2k_{id}^j \), where \( k_{id}^j \leftarrow \delta(\mathcal{VC}_{id}, k'_j) \)

- computes the following NIZK (implicitly, \( \psi \) different proofs)

\[
\sigma_{exp}^{1} := \text{ProveEq}((g, E_2, K_{ld}^{j}, E_2k_{id}^j), k_{id}^j)
\]

proves that \( E_2k_{id}^j \) is computed by raising the elements of \( E_2 \) to the Voter choice return code generation private key \( k_{id}^j \) corresponding to the Voter choice return code generation public key \( K_{ld}^{j} \). The proof is logged to be validated during the audit phase.

- Send back to the voting server \( (E_2k_{id}^j) \)

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

\[
(d_1, \ldots, d_{\psi}) := (g^{-r'} \hat{k})^{-sk_{CCr, i}^{(i)}}, \ldots, (g^{-r'} \hat{k})^{-sk_{CCr, i}^{(\psi)}}
\]

- computes the following NIZK

\[
\sigma_{exp}^{2} := \text{ProveEq}((g, g^{-r'} \hat{k}, \prod_{i=1}^{\psi} p_{CCr, i}^{(i)}, \prod_{i=1}^{\psi} d_i), \prod_{i=1}^{\psi} sk_{CCr, i}^{(i)})
\]

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

- Send back to the voting server \( \left( (g^{-r'} \hat{k})^sk_{CCr, i}^{(i)}, \ldots, (g^{-r'} \hat{k})^sk_{CCr, i}^{(\psi)} \right) \)

4.4.5 CreateRC(\( p_{C_1}^{id}, \ldots, p_{C_{\psi}}^{id}, \mathcal{VC}_{id}, C_{sk}, \text{CMtable} \))

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

- let \( 10C_{i}^{id} = H(p_{C_i}^{id} || \mathcal{VC}_{id}) \), and Choice Return Code encryption symmetric key \( sk_{C_{i}}^{id} = \delta(H(10C_{i}^{id} || \mathcal{VC}_{id})) \) for \( i \leq \psi \), where \( C_{sk} \) is the Code Secret key, and \( p_{C_i} \) are the pre-choice return codes computed by the voting server.

- output choice return codes

\[
\mathcal{CC}_{i}^{id} = \text{dec}^{c}(\text{Enc}^{c}(\mathcal{CC}_{i}^{id}; sk_{C_{i}}^{id}); sk_{C_{i}}^{id})
\]

iff \( [H(10C_{i}^{id}), \text{Enc}^{c}(\mathcal{CC}_{i}^{id}; sk_{C_{i}}^{id})] \) exists as an entry in \( \text{CMtable} \) \( \forall i \leq \psi. \)

\footnote{Recall that \( k_{id}^j \) is the Voter Choice Return Code Generation private key.}
This is the algorithm used by the voting server to compute the choice return codes \( \{CC_i^{id}\}_{i=1}^\psi \) to be sent to the voting client.

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

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

- output 1 iff \( CC_i \in \{CC_1^{id}, \ldots, CC_\psi^{id}\} \forall i = 1, \ldots, \psi \)
- 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 ProcessConfirm(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_1 checks if the number of confirmation attempts has been reached. If the voter exceeded the number of confirmation attempts – return \( \perp \). Otherwise, log the query from the voting server and the answer.
- Each CCR_1 computes
  \[
  h_{cm_1}^{id} = (H(CM_{id}))^{2k_{c_1}},
  \]
  where \( k_{c_1}^{id} := \delta(VC_{id}||ConfirmStr||k'_j)^{13} \) and sends it back to the voting server.
- Each CCR_1 generates the following NIZK proof of correct exponentation
  \[
  \pi_{exp} := \text{ProveEq} ((g, (H(CM_{id}))^{2}, k_{c_1}^{id}, h_{cm_1}^{id} , k_{c_1}^{id}))
  \]
  The proof is logged to be validated during the audit phase.

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

\[^{13}\text{Recall that } k_{c_1}^{id} \text{ is the Voter Vote Cast Return Code Generation private key, and } k'_j \text{ is the CCR}_j\text{'s Choice Return Code Generation private key.}\]
- The voting server computes the corresponding pre-vote cast return code as
  \[ pVCC^{id} = hcm^{id} \cdot hcm^{id} \]

- The voting server computes the long vote cast return code as
  \[ lVCC^{id} = H(pVCC^{id}||VC^{id}) \] and the vote cast return code encryption symmetric key
  \[ skvcc^{id} = \delta(H(lVCC^{id}||C^{sk})) \].

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

- The voting server checks that the retrieved short vote cast return code is correct by running
  \[ \text{Verify}(VCCs^{pk}, VCC^{id}||S^{VCC^{id}}) \]

- The pair (VCC^{id}, S^{VCC^{id}}) is added in the entry of the ballot box bb corresponding to identifier VC^{id}.

In case all the verifications succeed, the output of the algorithm is the pair (VCC^{id}, S^{VCC^{id}}). Otherwise, the output is \( \bot \).

4.5 Tally phase algorithms

The tally phase consists on the following algorithms.

4.5.1 Cleansing(EL^{pk}, pk^{(1)}_{CCR}, \ldots, pk^{(\psi)}_{CCR}, VCCs^{pk}, 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 \( BB = \{(VC^{id}, b, VC^{id}, VCC^{id}, S^{VCC^{id}})\}_{id \in ID} \)

- it creates a list \( L_{\text{checked}} \) with all \((VC^{id}, b) \in bb\) such that \( \text{ProcessVote}(EL^{pk}, pk^{(1)}_{CCR}, \ldots, pk^{(\psi)}_{CCR}, VC^{id}, b) = 1 \) and \( \text{Verify}(VCCs^{pk}, VC^{id}||S^{VCC^{id}}) = 1 \), where \((VC^{id}, VCC^{id}, S^{VCC^{id}})\) are not null.

- from each \((VC^{id}, b) \in L_{\text{checked}}\), obtains \( E1 \) from \( b \). Next, parses \( E1 \) as ciphertext components \((c_1, c_2)\), and adds \((c_1, c_2)\) to a list \( L \)

The output is the list \( L = \{(c_1, c_2)\} \)

4.5.2 MixDec^{(1)}(EL^{pkx}, L = \{(c_1, c_2)\})

It is run by CCM1 and proceeds as follows:

- from the ciphertext list \( L = \{(c_1, c_2)\}\) computes \( L^x, P_x := \text{Mix}(EL^{pkx}, L)\), namely a shuffled list \( L^x = \{t_1, t_2\}\)_{mix} and a proof \( P_x \) of correct mixing

- From \( L^x \) it computes a partially decrypted list \( L_1 = \{(c_1^{(1)}, c_2^{(1)})\}_{\text{dec}} \) and a list of zero-knowledge proofs \( P_{\text{dec}} := \{\pi_{\text{dec}}\} \) of correct partial decryptions computed as \( \pi_{\text{dec}} := \text{ProveDec} \left( (t_1, t_2), (t_1) - EL^{(1)}_{pkx}, EL^{(1)}_{pkx} \right) \) for each \( (t_1, t_2) \in L^x \)

32
- output $\left(L^s = \{ (r_1, r_2) \right\}_{mix}, L_1 = \{ (c_1^{(1)}, c_2^{(1)}) \}_\text{dec}, P_a, P_{\text{dec}} \right)$.  

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

4.5.3 MixDec$_2 \left( \text{EL}_{pk}^{(2)}, L = \{ (c_1, c_2) \}, L_1 = \{(c_1^{(1)}, c_2^{(1)}) \}_{mix}, P_a, P_{\text{dec}} \right)$  

It is run by CCM$_2$ and proceeds as follows:  

- from the ciphertext list $L_1 = \{(c_1^{(1)}, c_2^{(1)}) \}_\text{dec}$ computes $(L_1^s, P_a) := \text{Mix}(\text{EL}_{pk}^{(2)}, L_1)$, namely a shuffled list $L_1^s = \{(\tilde{c}_1, \tilde{c}_2)\}_{mix}$ and a proof $P_a$ of correct mixing.  

- From $L_1^s$ 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 $\pi_{\text{dec}} := \text{ProveDec} \left( \{(\tilde{c}_1, \tilde{c}_2), \tilde{c}_2 \cdot (\tilde{c}_1)^{-\text{EL}_{pk}^{(2)}}, \text{EL}_{pk}^{(2)} \} \right)$ for each $(\tilde{c}_1, \tilde{c}_2) \in L_1^s$.  

- output $(L_1^s, L_v, P_a, P_{\text{dec}})$  

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

Finally, fact($V_i$) is run for every $V_i \in L_v$ to output the prime factors $\{v_i^0\}_{i=1}^\Omega$ of $V_i$. It is tested that the combination of voting options $\{v_i^0\}_{i=1}^\Omega \in \Omega$. Otherwise, the vote is discarded.  

4.6 Verify phase algorithm  

The verify phase consists on the following algorithms.

4.6.1 VerifyTally(bb, bb$^\text{CCR}_1$, bb$^\text{CCR}_2$, bb$^\text{CCM}_1$, bb$^\text{CCM}_2$, r, II)  

It takes as input the Server’s ballot box $bb$, logs of CCR$_1$, CCR$_2$, CCM$_1$, CCM$_2$ denoted as bb$^\text{CCR}_1$, bb$^\text{CCR}_2$, bb$^\text{CCM}_1$, bb$^\text{CCM}_2$, 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$^\text{CCM}_2$ it’s logged that more than one distinct input had been shuffled for the given BB$_{id}$, aborts.  

- if MixVerify($L_1, L_1^s, P_a$) = 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\((\bb, \bb^{\text{CCR}_1}, \bb^{\text{CCR}_2}, \bb^{\text{CCM}_1}, \bb^{\text{CCM}_2}, L, L^*, L_1, \Pi_m, \Pi_{\text{dec}})\)

It takes as input the Server’s ballot box \(\bb\), logs of \(\text{CCR}_1, \text{CCR}_2, \text{CCM}_1\) denoted as \(\bb^{\text{CCR}_1}, \bb^{\text{CCR}_2}, \bb^{\text{CCM}_1}\), 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 \(\text{CCM}_1\) (in case there are more than 2 \(\text{CCMs}\), the auditors perform verification of all operations done before the final mixing and decryption).

1. If information about received and confirmed ballots in \(\bb, \bb^{\text{CCR}_1}, \bb^{\text{CCR}_2}\) does not match, aborts.

2. If computed long choice codes and long vote cast codes are not extractable from \(\bb, \bb^{\text{CCR}_1}, \bb^{\text{CCR}_2}\) and \(\text{CMtable}\) aborts.\(^{13}\)

3. Re-computes the cleansed ballot box by running \(\text{Cleansing}\) 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 \(\bb^{\text{CCM}_1}\) it’s logged that more than one distinct input had been shuffled for the given \(\text{BB}_{\text{id}}\), aborts.

6. If \(\text{MixVerify}(L, L^*, P_m) = 0\), 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 Printing service 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 \(\text{CCM}_1, \text{CCM}_2\) respectively.

The Printing service runs the \(\text{Setup}\) algorithm in conjunction with the Return Codes Control Components \(\text{CCR}_1, \text{CCR}_2\). At the end of the Setup phase the Printing service 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 \(\text{VC}_{\text{id}}\) in the bulletin board \(\text{BB}\). 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 \(\{\text{SVK}_{\text{id}}, \text{VC}_{\text{id}}, \text{BCK}_{\text{id}}, \text{VCC}_{\text{id}}; \{v_i, \text{CC}_{i}^{\text{id}}\}_{i=1}^{m}\}\)

\(^{14}\)See [34, Section 8.5] for a precise meaning of extractable codes
**Voting phase.** This phase consists of several steps:

1. The voter provides $\text{SVK}_\text{id}$ 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 $b$.

3. The ballot $b$ is sent to the Voting Server together with the Verification Card ID $\text{VC}_\text{id}$.

4. Upon reception of $(\text{VC}_\text{id}, b)$, 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 $(\text{VC}_\text{id}, b)$ 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 $C_{sk}$. 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 $\text{BCK}_\text{id}$ to the voting device, which generates a Confirmation Message $\text{CM}_\text{id}$ using the Confirm algorithm, that is sent to the bulletin board manager with the Verification Card ID $\text{VC}_\text{id}$. 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 $\text{CM}_\text{id}$ and interacting the Choice Return Codes Control Components. In case the operation is successful (the output is different from $\bot$), it updates the Ballot Box with the retrieved Vote Cast Return Code $\text{VCC}_\text{id}$ and signature $S_{\text{VCC}_\text{id}}$, and sends the Vote Cast Return Code to the voter, who checks it matches the Vote Cast Return Code $\text{VCC}_\text{id}$ 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 $\text{VCC}_\text{id}$ 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\(_1\).

2. The auditors run algorithm 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\(_2\) runs algorithm MixDec\(_2\), obtaining and publishing in the bulletin board BB the result \(r\) and the proof \(\Pi\).

4. The auditors run algorithm 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 Provably secure complete verifiability

In the ordinance of the Swiss Chancellery [13], a set of security objectives are identified in order to authorize candidate voting schemes to function over certain percentage of the cantonal electorate. Starting from the smallest percentage (30 per cent) the security objectives are strengthen when the electorate's percentage is increased. Informally, these objectives state that, under the trust assumptions of Section 2, an attacker is unable to change, misappropriate, or cast a vote with a (possibly) compromised voting device and voting server. Furthermore, this should hold independently of whether or not the vote has been registered by a trusted component of the system.

We aim to define cryptographically secure properties accounting for these objectives. More concretely, using the provably security framework, we define four games that capture the different adversaries and their powers considered in the ordinance. Refer to Section 6 for a more detailed discussion. We shall see later in Section 6 that sVote enjoys such properties, and here we only note that the proofs are with respect the random oracle model.

We say that a voting scheme as defined in Section 4 is qualified to satisfy provably secure complete verifiability in our analysis if the four types of attacks presented here can only occur with negligible probability or they can be detected to happen with high probability. In these attacks, the adversary, who is targeting an honest user, may be any collusion of a compromised voting device, voting server, control component, and a fraction of the voters. However, recall that at least one control component is trusted.

5.1 Type 1 attacks (vote tampering)

In these types of attacks, an honest voter casts a vote, and the adversary manages to change her selection to one of his choice. The voter does not detect the modification although she follows the verification protocol.
5.2 Type 2 attack (voting rejection)

In these types of attacks, an honest voter casts and verifies a vote, but the adversary manages to avoid confirmation of the vote, resulting in a non-confirmed vote.

5.3 Type 3 attack (vote cast)

The adversary manages to cast a vote on behalf of an honest user who is not willing to vote. Thus, the voter starts the voting process, but changes her mind and does not cast any vote. The adversary ignores her decision and casts a vote on her behalf.

5.4 Type 4 attack (incorrect tally)

The adversary manages to change the contribution of a confirmed ballot in the election result. This includes being able to cancel ballot’s contribution to the result.

Adversary knowledge. The adversary has little chances to successfully carry out any of the first three attacks outlined above without the start voting key $SV_{id}$ of the (honest) target voter: this follows from the fact that generation of a valid ballot implies knowledge of the verification card key $k_{id}$, since one of the nizks inside the ballot is relative to equal exponent, with $k_{id}$ being such exponent.

5.5 Active adversaries in sVote

In the complete verifiability model, an adversary is given control of the voting server, this is unrivalled in the electronic voting literature when considering individual verifiability. In the case of sVote, it is the voting server who builds the ballot box $bb$, giving the adversary ample room to remove confirmed ballots or store unconfirmed ballots, allow multiple voting for the same voter etc. These attacks may not be stopped during the voting or tally phases, but they can be detected in the post-election phase by a trustworthy auditor who checks consistency between the ballot box $bb$ and the transcripts logged by CCR$_{1}$ and CCR$_{2}$. Given that one of these control components is trustworthy (wlog we assume CCR$_{1}$ is trustworthy), at least one of the logged transcripts will be faithfull. Therefore, if the adversary wishes to avoid detection, it will see its actions limited: in case an inconsistency is found with respect to an entry $(VC_{id}, b) \in bb$, this ballot is flagged as suspicious and will not be part of the final count.

5.6 Adversary admissibility

One example of transcript inconsistency is the presence of ballots with valid vote cast codes and signatures not extractable from the mapping table stored in

---

\(^{15}\) Revises the verifiability literature and every model considers the voting server/bulletin board honest w.r.t. protecting the voter’s choices against a malicious voting device.

\(^{16}\)See discussion on secure logs from Section 4.1.

---

37
the voting server (see Definition 1). During one of the security reductions we will use that such ballots are detectable and will not be present in the cleansed ballot box.

Admissibility via auditability. In Definition 2 (see below) we precisely specify the consistency checks carried out during the audit phase that we use. This definition is provided for clarity, but we emphasize that the conditions (tests) therein are indeed part of the tests conducted in the audit phase. With these tests at hand the type of adversaries against whom we need to show security is largely reduced.

Definition 1 (Valid confirmed ballot (VCB)) Let \( b = (\text{VC\textsubscript{id}}, b, \text{VCC\textsubscript{id}}, S_{\text{VCC\textsubscript{id}}}) \) be an entry in the ballot box. We say that \( b \) is a valid confirmed ballot if for \( j = 1, 2 \) there exist at least one entry \( (\text{VC\textsubscript{id}}, \text{CM\textsubscript{id}}, p_{\text{VCC\textsubscript{j}\textsubscript{id}}}) \) in the secure log of CCR\(_j\), where \( \text{CM\textsubscript{id}} \) is a query submitted by the voting server to CCR\(_j\) during ProcessConfirm procedure and \( p_{\text{VCC\textsubscript{j}\textsubscript{id}}} \) is the corresponding answer, allowing the retrieval of the short vote cast code \( \text{VCC}\textsubscript{id} \) from the mapping table \( \text{CMtable} \). More precisely, there exist at least one entry \( (h, ct) \in \text{CMtable} \) such that we have

\[
\begin{align*}
1\text{VCC}\textsubscript{id} &= H\left(p\text{VCC}\textsubscript{j}(1) \cdot p\text{VCC}\textsubscript{j}(2)||\text{VC\textsubscript{id}}\right) \\
h &= H(1\text{VCC}\textsubscript{id}) \\
\text{skvcc} &= \delta(H(1\text{VCC}\textsubscript{id}||C_{sk})) \\
\text{VCC}\textsubscript{id}||S_{\text{VCC}\textsubscript{id}} &= \text{Dec}^s(ct; \text{skvcc})
\end{align*}
\]

On auditing valid confirmed ballots. To verify that a ballot has a valid confirmation one does not need to extract the vote cast code from the mapping table. It is enough to check up to step (2). This is justified by the collision resistance property of \( H \) and because with high probability different symmetric keys are derived for different \( l\text{VCC}\textsubscript{id} \neq l\text{VCC}\textsubscript{id}' \). Another observation is that if a ballot is not a valid confirmed ballot, then during the voting phase the CCR\(_j\) is not queried the same confirmation message as the one generated during setup. This is what the following lemma shows.

Lemma 1 Let \( b = (\text{VC\textsubscript{id}}, b, \text{VCC\textsubscript{id}}, S_{\text{VCC\textsubscript{id}}}) \) be a ballot stored in the ballot box, and let \( \text{CM\textsubscript{id}} = (\text{BCK}\textsubscript{id})^2 k_{id} \) be the confirmation message generated during the setup phase (using Setup, SDM). Then, if the (potentially corrupted) voting server queries \( q_v = \text{CM\textsubscript{id}} \) to all CCR\(_s\), \( b \) is a valid confirmed ballot with high probability.

Proof sketch. First, unless the adversary breaks soundness of EqDL we may assume the following:

1. During the setup phase, when the printing office queries \( q_s = H(\text{CM\textsubscript{id}})^2 \), it receives back \( p\text{VCC}\textsubscript{j}(s) = q_s k_{id} \) from each CCR\(_j\).

2. During the voting phase, when the voting server queries \( q_v \), it receives back \( xp\text{VCC}\textsubscript{id}(v) = H(q_v)^2 k_{id} \) from each CCR\(_j\).
It follows from the above two observations that if \( q_v = \text{CM}^{id} \), then \( p_{VCC}^{(j)} = p_{VCC}^{(i)} \) for \( j = 1, 2 \). Second, using the completeness of the protocol, we have that \( p_{VCC} = xp_{VCC} \), and therefore \( b \) is a valid confirmed ballot by definition.

**Definition 2 (Suspicious ballot)** A \((V_{C^{id}}, b) \in bb\) is suspicious if it satisfies any of the following tests.

**Test 1:** The ballot box, or the secure logs of CCR\(_1\) or CCR\(_2\) contain multiple different ballots for the same identifier \( V_{C^{id}} \).

**Test 2:** There is a ballot in the ballot box but it does not appear in the secure logs of CCR\(_1\) or CCR\(_2\).

**Test 3:** There are different (possibly not valid) confirmed ballots for the same identifier \( V_{C^{id}} \).

**Test 4:** There is an entry in the ballot box that is not a valid confirmed ballot (see Defn. 1).

**Test 5:** There exists a voter with a confirmed ballot in the secure logs, but there exists no entry for the ballot in the voting server.

### 6 Security analysis

The complete verifiability of sVote is proven using game-based proof techniques, where we end up relating the security of the protocol to some specific properties of the cryptographic primitives which implement it. The games are played between a Challenger, which represents the honest part of the system, and an Attacker, which represents the dishonest parties. During the development of the games, bold text is used to mark changes between transitions for a better identification.

Although the games are represented with one voting option for simplicity, they can readily be extended to multiple options as far as none of the voter’s choices is repeated and the number of voting options the voter can select is fixed. This means that, in case the voter can select up to \( \psi > 1 \) 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 different Choice Return Codes corresponding to such blank options besides her selections.

We define four games, each one representing one attack type. In the first two the adversary tries to subvert the actions of the voter, while still having to get the correct verification data so that the voter does not notice any change in the expected application flow. In the third game the adversary tries to get a ballot confirmed without the collaboration of the voter and still get a valid vote cast code generated by the challenger (one for which the published signature validates, so that this vote is accepted in the tally phase). In the fourth game the attacker seeks to manipulate the tally phase of the election, by giving a result in which some of the ballots confirmed by honest voters contribute to the election result in a way different to what was intended by the voter.

The Attacker can adaptively register both honest and corrupt voters. While for corrupt voters the Attacker is provided with all the registration information including the mapping among voting options and Choice Return Codes, for
honest voters the Attacker receives only the information necessary to cast a vote (but in some of the games the adversary is not given the data to confirm a vote). The public registration information from all voters is published by the Challenger on an append-only bulletin board that is accessible to all parties. The Attacker has also the ability to cast votes, which are processed by the Challenger to return the corresponding Choice Return Codes.

Adversaries considered. We say that an attacker (adversary) is admissible if the actions triggered by the attacker in the security games do not produce any suspicious ballot corresponding to an honest voter (see Definition 2).

Type 1-3 attacks. The main strategy for an attacker in type 1-3 attacks is to undetectably manipulate voter id’s intended choice $x_{id}$ by recovering the target values $CC_{id}$ and $VCC_{id}$ from the mapping

$$\text{map}^id = \left\{ \left[ \{H(1CC_{id}), \text{Enc}^s(CC_{id}; skcc_{id}) \} \right]_{j=1}^n, \left[ H(1VCC_{id}), \text{Enc}^s(VCC_{id}||S_{VCC_{id}}; skvcc_{id}) \right] \right\}$$

while casting a vote for a different choice. Those target values are encrypted using symmetric keys $\{ skcc_{id} \}_{j=1}^n$ and $skvcc_{id}$ computed as shown next.

$$1CC_{id} = H \left( (x_{id})^{k_{id}}x_{id}^1 \cdot (x_{id})^{k_{id}}x_{id}^2 || VC_{id} \right)$$

(5)

$$1VCC_{id} = H \left( (CM_{id})^{k_{1id}} \cdot (CM_{id})^{k_{2id}} || VC_{id} \right)$$

(6)

$$skcc_{id} = \delta(H(1CC_{id}||C_{sk})) ; skvcc_{id} = \delta(H(1VCC_{id}||C_{sk}))$$

(7)

Since the attacker controls CCR2 and the Voting Device it can compute on knowledge $BCK_{id}$ the quantities $(CM_{id})^{k_{1id}} = (BCK_{id})^2 k_{1id}k_{2id}$ and $(x_{id})^{k_{1id}}x_{id}^2$ by itself. However the attacker needs the cooperation of the trustworthy CCR1, to recover the missing quantities $(CM_{id})^{k_{1id}}$ and $(x_{id})^{k_{1id}}x_{id}^2$. See also Section 4.3.2

Type 4 attacks. The main strategy for an attacker to undetectably manipulate the election result is to tamper with the different operations that take place during the tally phase. The attacker could try to modify the voter’s voting choices when casting and storing ballots, but it has been previously shown that the adversary cannot do so undetectably. Still, in the universal verifiability game defined below the attacker is allowed to submit ballots on behalf of honest voters, giving the attacker the possibility to alter the voter’s choice before it is stored in the ballot and confirmation boxes.

We prove the following result. We note that the probability of an Attacker succeeding on breaking complete verifiability is defined as the maximum probability of an Attacker winning in any of the above four games.

Result 1 Let $(\text{ProveEqEnc}, \text{VerifyEqEnc}, \text{SimEqEnc})$ and $(\text{Mix}, \text{MixVerify}, \text{SimMix})$ be sound NIZKP schemes, the symmetric encryption scheme $(\text{KGen}^s, \text{Enc}^s, \text{Dec}^s)$ be modeled as a pseudo-random function, the voter’s Verification Card key pairs be uniformly sampled from the key space and the size of the group $G$ be much
larger than the number of voters, the hash function $H$ be collision-resistant. Then the Attacker advantage in the protocol abstractions described in Section 4 is negligible when trying to defeat the complete verifiability property in the random oracle model.

6.1 Game A: vote tampering

In this game, the goal of the adversary is to cast a ballot on behalf of an honest voter \( id^* \), containing a choice different to the voter’s option \( x^* \) but compute the right choice return code \( \sigma^* = CC_{id}^{x^*} \).

**Configuration:**

**Challenger:**

\[
\begin{align*}
\text{Compute } & EL_{pk} = EL_{pk}^{(1)} \cdot EL_{pk}^{(2)} \\
\text{Compute } & (C_{sk}, VCC_{sk}, VCC_{sk}) \leftarrow \text{Setup}_{SDM}(\lambda) \\
\text{Initialize} & \text{ the empty voter lists } ID_h, ID_c, ID = (ID_h \cup ID_c) \\
\text{Post } & (EL_{pk}, VCC_{sk}, ID) \text{ in } BB \\
\text{Keep } & (EL_{sk}, C_{sk}, VCC_{sk})
\end{align*}
\]

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

- **OChallenger_RegisterHonest**(\( id \)):
  
  \[
  \begin{align*}
  \text{If generated } id & \in ID, \text{ stop and return } \perp. \text{ Otherwise:} \\
  \text{Add } & id \text{ to } ID_h \\
  \text{Select } & x_{id} \leftarrow \{v_1, \ldots, v_n\} \\
  \text{Generate } & (SVK_{id}, VC_{id}, K_{id}, VCC_{id}, VC_{sk}, \{v_j, CC_{id}^{v_j}\}_{j=1}^{n}, \text{map}^{id}) \\
  & \text{ in interaction with the Attacker} \\
  \text{Post } & (id, VC_{id}, K_{id}, VCC_{id}, \text{map}^{id}) \text{ in } BB \\
  \text{Provide } & SVK_{id}, x_{id} \text{ to the Attacker} \\
  \text{Keep } & \{v_i, CC_{id}^{v_i}\}_{i=1}^{n}, \text{VCC}^{id}
  \end{align*}
  \]

- **OChallenger_RegisterCorrupt**(\( id \)):
  
  \[
  \begin{align*}
  \text{If generated } id & \in ID, \text{ stop and return } \perp. \text{ Otherwise:} \\
  \text{Add } & id \text{ to } ID_c \\
  \text{Generate } & (SVK_{id}, VC_{id}, K_{id}, VCC_{id}, VC_{sk}, \{v_j, CC_{id}^{v_j}\}_{j=1}^{n}, \text{map}^{id}) \\
  & \text{ in interaction with the Attacker} \\
  \text{Post } & (id, VC_{id}, K_{id}, VCC_{id}, \text{map}^{id}) \text{ in } BB \\
  \text{Provide } & (SVK_{id}, VC_{id}, BC_{id}, VCC_{id}, \{v_i, CC_{id}^{v_i}\}_{i=1}^{n}) \text{ to the Attacker}
  \end{align*}
  \]

**Voting:** The Attacker generates votes running the CreateVote algorithm, or confirm votes running ProcessConfirm, and can ask the Challenger to run the following algorithms several times:
OCChallenger_Vote(id, b):
run ProcessVote(pk_{CCR}^{(1)}, VC_{id}, b). If result is 1:
compute σ ← CreateCC(b, c_{sk}, map_{id})
in interaction with the Attacker
add (VC_{id}, b) to bb^{CCR}_1
send σ
Else send ⊥

OCChallenger_Condirm(id, CM_{id}):
Run (μ, S_μ) ← ProcessConfirm(bb, VC_{id}, CM_{id}, c_{sk}, VCCs_{pk}),
in interaction with the Attacker
add (VC_{id}, (CM_{id})k_{c_{id}}) entry to BB^{CCR}_1
if (μ, S_μ) ≠ ⊥: add (VC_{id}, μ, S_μ) entry in BB,
Send μ

Attacker: Is in control or has additional knowledge of the following:
Computes key pairs (El_{pk}^{(2)}, El_{sk}^{(2)}) and (k_2', k_2'); key pairs (k_{id}^{(2)}, k_{id}^{(2)}) and
(k_{c_{id}}^{(2)}, k_{c_{id}}^{(2)}) for every voter id ∈ ID.
Receives c_{sk} and CMtable
Controls boxes bb, bb^{CCR}_2.

Security reduction of game A

The transitions (or hops) in this game are oriented to prove that all the information
the Attacker may have access to is independent from the Choice Return
Code it has to generate, and therefore that it has no significant advantage with
respect to guessing the value at random. Throughout the hybrids, for Game A.i,
we denote with S_A^{i} the event that ∃ (VC_{∗_{id}}), b) ∈ BB s.t. Dec(b) ≠ x_{id} and σ∗ = CC_{id}^{x_{id}}, for
VC_{∗_{id}} ∈ ID_{h}.

GAME A.1 First we proceed with a game transition based on the soundness of
one of the NIZK proof systems. Let this be the game in which the adversary
is restricted to submit to the ProcessVote algorithm from game A.1
ballots whose encrypted options and encrypted partial Choice Return
Codes have been computed over the same voting option x.

Define S_A^{i} as the event that ∃ (VC_{∗_{id}}), b) ∈ BB s.t. Dec(b) ≠ x_{id} and σ∗ = CC_{id}^{x_{id}}, for
VC_{∗_{id}} ∈ ID_{h}. We claim that

| Pr{S_A^{i}} − Pr{S_A^{0}} | = \epsilon_{\text{sound}}^{\text{EqEnc}},

where \epsilon_{\text{sound}}^{\text{EqEnc}} is the advantage of an efficient algorithm against the soundness of
the NIZKP scheme for the equality of encryptions (which is negligible assuming
the properties of NIZKPs).
GAME A.2  This transition is also based on indistinguishability: let this be the game in which for honest voters \( \text{id} \in \mathcal{TD}_h \), the Challenger changes the computation of \( k^1_{\text{id}} = \delta(k'_1, \mathcal{VC}^*_{\text{id}}) \) and \( k^1_{\text{id}} = \delta(k'_1 || \mathcal{VC}^*_{\text{id}} || \text{ConfirmStr}) \), where 

\[
(k'_1, k'_1) \leftarrow \text{Gen}_e(G) \quad \text{and} \quad (k_{c_1}, k'_1) \leftarrow \text{Gen}_e(G), \quad \text{to} \quad k^1_{\text{id}}, k^1_{\text{id}} \leftarrow \mathcal{R} \subset \mathbb{Z}_q.
\]

Still, the Challenger will keep a list \( \{ (\text{id}, k^1_{\text{id}}, k^1_{\text{id}}) \}_{\text{id} \in \mathcal{TD}_h} \) of created keys internally, so that they can be provided whenever needed.

We claim that

\[
| \Pr\{S^A_i\} - \Pr\{S^A_i\} | = \epsilon_{\text{dl-kdf}}.
\]

where \( \epsilon_{\text{dl-kdf}} \) is the KDF-advantage of an efficient KDF distinguisher algorithm, that tries to distinguish \( K, K_c \leftarrow \mathcal{R} \subset \mathcal{K}_{\text{sp}} \) from \( K \leftarrow k^1_{\text{id}} = \delta(k'_1, \mathcal{VC}^*_{\text{id}}), K_c \leftarrow k^1_{\text{id}} = \delta(k'_1 || \mathcal{VC}^*_{\text{id}} || \text{ConfirmStr}) \), with \( k'_1 = (g)^k \) known.

GAME A.3  This transition is again based on indistinguishability: let this be the game in which for honest voters \( \text{id} \in \mathcal{TD}_h \), the Challenger changes for \( j = 1, \ldots, n \) the computation of \( (v_j)^{k^1_{\text{id}}} \) in Setup.SDM(\cdot) and CreateCC(\cdot) by replacing them with \( R^1_{\text{id}} \leftarrow \mathcal{G} \). Still, the Challenger will keep the list of \( \{ R^1_{\text{id}} \}_{\text{id} = 1}^n \) for every \( \text{id} \in \mathcal{TD}_h \), so that they can be used whenever needed.

We claim that

\[
| \Pr\{S^A_i\} - \Pr\{S^A_i\} | = \epsilon_{\text{ddh}},
\]

where \( \epsilon_{\text{ddh}} \) is the DDH-advantage of an efficient distinguisher algorithm of Diffie-Hellman tuples in \( G \), that tries to distinguish tuples \( (g, k^1_{\text{id}}, v_j, (v_j)^{k^1_{\text{id}}}) \) from \( (g, k^1_{\text{id}}, v_j, R^1_{\text{id}}) \) for \( R^1_{\text{id}} \leftarrow \mathcal{R} \subset \mathcal{G} \) and \( j = 1, \ldots, n \). We observe that during the voting phase the Attacker has to submit to CCR, a ballot containing one of the voting options generated during setup; acting otherwise leads to a trivial loose in either of the games.

GAME A.4  This transition is also based on indistinguishability: let this be the game in which for honest voters \( \text{id} \in \mathcal{TD}_h \), the Challenger associates a random Choice Return Code \( \mathcal{CC}^*_{\text{id}} \) that is independent of the value appearing in the ciphertext \( \{ H(\mathcal{CC}_{\text{id}}^*), \text{Enc}^* (\mathcal{CC}_{\text{id}}^* || 1\text{CC}^*_{\text{id}}) \} \) that are stored in the mapping \( \text{map}^* \). Still, the Challenger will keep the list of Choice Return Codes internally, so that they can be provided during the voting phase.

We claim that

\[
| \Pr\{S^A_i\} - \Pr\{S^A_i\} | = \epsilon_{\text{enc}},
\]

where \( \epsilon_{\text{enc}} \) is the advantage of an efficient algorithm that distinguishes the output of \( \text{Enc}^* \) from a random function, which is negligible considering that \( \text{Enc}^* \) can be seen as a pseudo-random function.

Finally, we can see that the values the Attacker has access to: the Choice Return Codes \( \sigma \) generated during the voting phase, the information posted in the bulletin board, as well as the Choice Return Codes produced by the registration
algorithm and given to the Attacker for the corrupt voters; as well as the Choice Return Codes for the honest voter $id$ corresponding to every choice but $x_{id}$ are independent from the expected value $CC_{id}$ the Attacker has to generate, and therefore $\Pr \{ S_{x}^{A} \}$ is the probability of randomly guessing it, which is $\frac{1}{|A_{cc}|}$. We can conclude that

$$\Pr \{ S_{x}^{A} \} = \epsilon_{\text{sound}} + \epsilon_{\text{dl-kdf}} + \epsilon_{\text{ddh}} + \epsilon_{\text{enc}} + \frac{1}{|A_{cc}|},$$

and therefore

$$\text{CCadv}[A] = \epsilon_{\text{sound}} + \epsilon_{\text{dl-kdf}} + \epsilon_{\text{ddh}} + \epsilon_{\text{enc}},$$

which is negligible given the assumption that the voters’ keys are uniformly sampled from the key space, the order of the group $G$ is larger than the number of voters, cryptographic hash functions are collision-resistant, NIZKP schemes are sound, and the used key derivation function and the symmetric encryption scheme are computationally indistinguishable from random distributions and DDH is hard in $G$.

### 6.2 Game B: vote rejection

We model security against type 2 attacks with a game in which, for a pair $(VC_{id}, VCC_{id})$ corresponding to an honest user who casts a ballot $b_{id}$, the adversary wins if he guesses the right vote cast code $VCC_{id}$ and the ballot $b_{id} \in \mathbb{bb}$ is not a valid confirmed ballot (see Definition 1), with probability significantly larger than $\frac{1}{|A_{vcc}|}$. More formally the game is as follows.

**Configuration:**

**Challenger:**

- Compute $EL_{pk} = E_{pk}^{(1)} \cdot EL_{pk}^{(2)}$
- Compute $(C_{sk}, VCC_{sk}, VCC_{ask}) \leftarrow \text{Setup.SDM}(1^\lambda)$
- Initialize the empty voter lists $ID_{h}, ID_{c}, ID = (ID_{h} \cup ID_{c})$
- Post $(EL_{pk}, VCC_{sk}, ID)$ in $BB$
- Keep $(EL_{sk}^{(1)} \cdot C_{sk}, VCC_{sk})$

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

- **OChallenger_RegisterHonest(id):**
  - If generated $id \in ID$, stop and return ⊥. Otherwise:
  - Add $id$ to $ID_{h}$
  - Select $x_{id} \in \{v_{1}, \ldots, v_{n}\}$
  - Generate $(SVK_{id}, VC_{id}, K_{id}, VCC_{id}, BCK_{id}, VCC_{id}, \{v_{j}, CC_{id}^{j}\}_{j=1}^{n}, \text{map}^{id})$
    - in interaction with the Attacker
  - Post $(id, VC_{id}, K_{id}, VCC_{id}, \text{map}^{id})$ in $BB$
  - Provide $SVK_{id}, x_{id}, BCK_{id}$ to the Attacker
  - Keep $\{v_{i}, CC_{id}^{i}\}_{i=1}^{n}$, $VCC_{id}$
\textbf{OChallenger\_RegisterCorrupt}(id): 
If generated id \(\in T_D\), stop and return \(\perp\). Otherwise:
Add id to \(T_D\).
Generate \((SV_{id}, VC_{id}, K_{id}, VC_{ks_{id}}, BCK_{id}, VCC_{id}, \{v_j, CC_{id_i}\}_{i=1}^n, map_{id})\)
in interaction with the Attacker.
Post \((id, VC_{id}, K_{id}, VC_{ks_{id}}, map_{id})\) in \(BB\).
Provide \((SV_{id}, VC_{id}, BCK_{id}, VCC_{id}, \{v_i, CC_{id_i}\}_{i=1}^n)\) to the Attacker.

\textit{Voting:} The Attacker generates votes running the CreateVote algorithm, or any other algorithm of its choice, and can ask the Challenger to run the following algorithms several times:

\textbf{OChallenger\_Vote}(id, b):
- run \text{ProcessVote}(pk_1^{CCR}, VC_{id}, b). If result is 1:
  - compute \(\sigma \leftarrow \text{CreateCC}(b, C_{sk}, map_{id})\) in interaction with the Attacker.
  - add \((VC_{id}, b)\) to \(bb_{CCR}^1\).
  - send \(\sigma\)
Else send \(\perp\).

\textbf{OChallenger\_Confirm}(id, CM_{id}):
- Run \((\mu, S_\mu) \leftarrow \text{ProcessConfirm}(bb, VC_{id}, CM_{id}, C_{sk}, VCC_\text{pk}),\)
in interaction with the Attacker.
- add \((VC_{id}, (CM_{id})_{k_{C1}^2})\) entry to \(BB_{CCR}^1\).
- if \((\mu, S_\mu) \neq \perp\): add \((VC_{id}, \mu, S_\mu)\) entry in \(BB\), Send \(\mu\).

\textit{Attacker:} Is in control or has additional knowledge of the following:

\textbf{Attacker:}
- Computes key pairs \((EL_{pk}^{(2)}, EL_{sk}^{(2)})\) and \((K_2', k_2')\); \((K_2, k_2)\) and \((K_{2_{id}}, k_{2_{id}})\)
  for every voter id \(\in T_D\).
- Receives \(C_{sk}\) and \(CM_\text{table}\).
- Controls boxes \(bb, bb_{CCR}^2\).

At the end of the game, the Attacker provides \((VC_{id^*}, \mu^*)\).
Define \(S_0^B\) as the event “\(\mu^* = VCC_{id^*}\) and the corresponding ballot \((VC_{id^*}, b)\) \(\in BB\) is not a valid confirmed ballot”. Recall that the meaning of a valid confirmed ballot is given in Definition \([\ref{def:valid_confirmed_ballot}].\)

We define the advantage of an adversary as:

\[ VCCadv[A] = \left| \Pr[S_0^B] - \frac{1}{|A_{\text{vcc}}|} \right| \]

\textbf{6.2.1 Security reduction of game B}

The reduction proceeds by applying incremental changes to the algorithms of the voting scheme \(V = (\text{Setup}, \text{Vote}, \text{Tally}, \text{Verify})\) presented in Section \([\ref{sec:security_reduction}].\) and argue indistinguishability between these changes. The final tweaked scheme...
generates during algorithm Setup.SDM a mapping table with the target entry map^{id} containing no information whatsoever about the target vote cast code \text{VCC}^{id}.

Throughout the hybrids, we denote with \( S^{B}_1 \) the event “\( \mu^* = \text{VCC}^{id} \) and the corresponding ballot \( (\text{VCC}^{id}, b) \) \( \in \) \( \text{BB} \) is not a valid confirmed ballot”.

**Game B.1** In this game, the challenger replaces algorithms Setup.CCR_1 and ProcessConfirm with Setup.CCR_1^1 and ProcessConfirm^1 from the original scheme \( \mathcal{V} \). The changes are:

1. Algorithm Setup.CCR_1^1. For honest voters \( id \in \mathcal{ID}_h \), the Challenger computes \( k_{c}^{id} \leftarrow \mathbb{Z}_q \), and it stores \( k_{c}^{id} \) in a list \( L_{k_{c}^{id}} = \{ (id, k_{c}^{id}) \}_{id \in \mathcal{ID}_h} \). In the original scheme \( \mathcal{V} \) of the previous game, \( k_{c}^{id} \) was computed as \( k_{c}^{id} = \delta(k'_{1}, ||\text{VCC}^{id}|||\text{ConfirmStr}) \), where \( (K_{c1}, k'_{1}) \leftarrow \text{Gen}_s(G) \).

2. Algorithm ProcessConfirm^1. For honest voters \( id \in \mathcal{ID}_h \), the Challenger retrieves \( k_{c}^{id} \) from \( L_{k_{c}^{id}} \) and proceeds as in ProcessConfirm of the previous game.

We claim that
\[
|\Pr\{S^{B}_0\} - \Pr\{S^{B}_1\}| = \epsilon_{dl-kdf},
\]
where \( \epsilon_{dl-kdf} \) is the KDF-advantage of an efficient KDF distinguisher algorithm, that tries to distinguish \( K, K_c \leftarrow \mathcal{K}_\exp \) from \( K \leftarrow k_{c}^{id} = \delta(k'_{1}, ||\text{VCC}^{id}|||\text{ConfirmStr}) \), with \( k'_{1} = (g)^{k_{1}} \) known.

**Game B.2** In this game, the challenger replaces algorithms Setup.CCR_1 and ProcessConfirm from the previous scheme \( \mathcal{V} \) with algorithms Setup.CCR_2 and ProcessConfirm^2. The changes are:

1. Algorithm Setup.CCR_2. For honest voters \( id \in \mathcal{ID}_h \), the Challenger fakes the proof regarding exponentiation of the received encryption containing the ballot casting key. Specifically the Challenger now computes:
\[
\pi_{\exp} := \text{SimEq}(g, \text{ctbck}^{id}, K_{c}^{id}, \text{ctbck}^{id})
\]
In the previous game \( \pi_{\exp} \) is computed using \text{ProveEq}.

2. Algorithm ProcessConfirm^2. Likewise, for honest voters \( id \in \mathcal{ID}_h \), the Challenger fakes the proof of the confirmation message using \text{SimEq}.

Note that now the knowledge of \( k_{c}^{id} \) is not used to generate the nizk proofs. We have that:
\[
|\Pr\{S^{B}_1\} - \Pr\{S^{B}_2\}| = \epsilon_{skdl},
\]
where \( \epsilon_{skdl} \) is the advantage of an adversary against the zero-knowledge property of the \text{EqDL} proof system. That is, any distinguisher against both games, can be used to distinguish between the output of \text{ProveEq} and \text{SimEq}: the latter distinguisher runs the former and outputs the same.
In this game, the challenger replaces algorithm $\text{Setup.SDM}^2$ from the previous scheme $V^2$ with $\text{Setup.SDM}^3$. The changes are:

1. Algorithm $\text{Setup.SDM}^3$. For honest voters $id \in ID_h$ the Challenger computes the pre-Vote Cast Return Code $pVCC^{id}$ as follows

$$pVCC^{id} := (hbck^{id})^{kc^i_{id}} \cdot \text{Dec}(sk_{SDM}, (ctbck^{id})^{k_1^{id}})$$  \hspace{1cm} (8)

where $(ctbck^{id})^{k_1^{id}}$ is given by the (corrupted) CCR$_2$. Thus, now the Challenger does not use the ciphertext given by CCR$_1$. Instead raises $kc^i_{id}$ to $hbck^{id}$ in clear. Note the Challenger can do this because it controls the SDM and the (honest) CCR$_1$.

Using the perfect correctness and the homomorphic property of ElGamal, $pVCC^{id}$ in this game is identically distributed to $pVCC^{id}$ in the previous game, so

$$|Pr\{S^B_1\} - Pr\{S^B_2\}| = 0.$$

In this game, the challenger replaces algorithms $\text{Setup.SDM}^3$ and $\text{Setup.CCR}^3_1$ from the previous scheme $V^3$ with algorithms $\text{Setup.SDM}^4$ and $\text{Setup.CCR}^4_1$. The changes are:

1. Algorithm $\text{Setup.SDM}^4$. For honest voters $id \in ID_h$ the Challenger samples two random group elements $(g_1, g_2)$ and sets $ctbck^{id} = (g_1^{id}, g_2^{id})$. In the previous game $ctbck^{id}$ was an encryption of $hbck^{id}$.

2. Algorithm $\text{Setup.CCR}^4_1$. For honest voters $id \in ID_h$ the Challenger samples two random group elements $(g_1^{id}, g_4^{id})$ and sets $ctbck^{id} = (g_3^{id}, g_4^{id})$. It also samples random element $g_5^{id}$ and sets $Kc^i_{id} = g_5^{id}$ instead of setting $Kc^i_{id} = g^{kc^{i}_{id}}$. Observe that the Challenger can fake a proof $\pi_{exp}$ because it is using SimEq.

This hop can be unfolded in three hops. First, $ctbck^{id}$ is replaced with an encryption of one; this is guaranteed by the ind-cpa property of ElGamal. Second, the encryption of one can be replaced with the (random and independent) group elements $(g_1, g_2)$, using DDH. Third, the tuple

$$(g, g_1^{id}, g_2^{id}, \overline{ctbck}^{id}, Kc) = (g, g_1^{id}, g_2^{id}, (g_1^{id})^{k_1^{id}}, (g_2^{id})^{k_1^{id}}, g^{k_1^{id}}),$$

where $\overline{ctbck}^{id}$, and $Kc$ are as in the previous game is computationally indistinguishable from tuple

$$(g, g_1^{id}, g_2^{id}, \overline{ctbck}^{id}, Kc) = (g, g_1^{id}, g_2^{id}, g_3^{id}, g_4^{id}, g_5^{id}),$$

where $\overline{ctbck}^{id}$, and $Kc$ are as in this game. This follows from a variant of the DDH problem. See for example \cite{23} where this problem is called L-DDH (and shown equivalent to DDH). We have that:

$$|Pr\{S^B_1\} - Pr\{S^B_3\}| = \epsilon_{\text{ind-cpa}} + \epsilon_{\text{ddh}} + \epsilon_{\text{l-ddh}}$$

where $\epsilon_{\text{ind-cpa}}$ is the advantage of an adversary against ind-cpa of ElGamal encryptions under the public of the SDM.
Game B.5  In this game, the challenger replaces algorithms Setup,SDM\(^4\) and ProcessConfirm\(^4\) from the previous scheme \(\mathcal{V}^4\) with algorithms Setup,SDM\(^5\) and ProcessConfirm\(^5\). The changes are:

1. Algorithm Setup,SDM\(^5\). For honest voters \(id \in \mathcal{I}D_h\) the challenger computes the pre-Vote Cast Return Code \(p_{VCC}^id\) as follows

\[
p_{VCC}^id := h_s \cdot \text{Dec}(sk_{SDM}, (ctbck^{id})^{kc_2^id}), \tag{9}
\]

where \(h_s\) is a random group element. In the previous game, \(p_{VCC}^id\) was computed as in Equation 8.

2. Algorithm ProcessConfirm\(^5\). For honest voters \(id \in \mathcal{I}D_h\) the (honest) CCR\(_1\) controlled by the challenger answers with \(h_s\) on query \(CM^id\) from the (corrupted) Voting Server. In the previous game, this query was answered with \(H(CM^id)^2 \cdot kc_1^id\).

In this hop we use that the function \(f_{kc_1^id}(x) = (H(x))^{kc_1^id}\) is a member of a (weak) prf family (but note that inputs to the prf are the outputs of a random oracle). For a formal proof of this fact, see for example [28]. We have that:

\[
| \Pr\{S^B_5\} - \Pr\{S^B_4\} | = \epsilon_{\text{prf}}
\]

where \(\epsilon_{\text{prf}}\) is the advantage of an adversary against the (weak) prf security of \(\{f_k(\cdot)\}\). The actual reduction is trivial, noting that at this point the challenger only uses \(kc_1^id\) for \(f_{kc_1^id}(\cdot)\).

Game B.6  In this game, the challenger replace algorithms Setup,SDM\(^5\) and ProcessConfirm\(^5\) from the previous scheme \(\mathcal{V}^5\) with algorithms Setup,SDM\(^6\) and ProcessConfirm\(^6\). The changes are:

1. Algorithm Setup,SDM\(^6\). For honest voters \(id \in \mathcal{I}D_h\) the challenger computes the pre-Vote Cast Return Code \(p_{VCC}^id\) by sampling a random group element \(h_s\). In the previous game \(p_{VCC}^id\) was computed as in Equation 9.

2. Algorithm ProcessConfirm\(^6\). For honest voters \(id \in \mathcal{I}D_h\) the (honest) CCR\(_1\) controlled by the challenger keeps a list with pairs \((q_s = (BCK^{id})^2 \cdot h_s, h_s)\). On query \(CM^id\) from the (corrupted) Voting Server answers with \(h_s\) if \(CM^id = q_s\), otherwise returns a fresh random group element \(h_v\).

In order to see that ProcessConfirm\(^5\) is indistinguishable from ProcessConfirm\(^6\), and that Setup,SDM\(^5\) is indistinguishable from Setup,SDM\(^6\) we argue as follows.

First, as long as the adversary never gets to see \(h_s\) generated during setup, then \(p_{VCC}\) in both games are identically distributed; this follows because in the previous game the adversarial contribution to \(p_{VCC}\) is randomized with \(h_s\) (see Eq. 9), which is not seen by the adversary, and in this game \(p_{VCC}\) is a random group element.

Second, the adversary sees \(h_s\) only if it queries \(q_s\) in ProcessConfirm acting as the Voting Server; however, using Lemma 4 generating a non valid confirmed ballot and querying \(q_s\) to CCR\(_1\) happens only if the Attacker breaks the soundness of EqDL.
We have that:
\[
|\Pr\{S^B_6\} - \Pr\{S^B_5\}| = 2 \cdot \epsilon_{\text{sound-dl}}
\]
where \(\epsilon_{\text{sound-dl}}\) is the advantage of an adversary against the soundness of the EqDL proof system.

**Game B.7** In this game, the challenger replace algorithm Setup.SDM\(^6\) from the previous scheme \(\mathcal{V}\) with algorithm Setup.SDM\(^7\). The changes are:

1. Algorithm Setup.SDM\(^7\). For honest voters \(id \in \mathcal{I}_h\), the Challenger generates the mapping table encrypting a different short Vote Cast Return Code \(\text{VCC}^{id}\) than the one printed in the voting card of the honest voter \(id\).

We have that:
\[
|\Pr\{S^B_7\} - \Pr\{S^B_6\}| = \epsilon_{\text{ind-cpa}^s}
\]
where \(\epsilon_{\text{ind-cpa}^s}\) is the advantage of an adversary against the ind-cpa security of the (symmetric) encryption used during the generation of the mapping table. The reduction is trivial noting that the symmetric key
\[
\text{skvcc}^{id} = \delta(H(H(\text{pVCC}^{id}|\text{VCC}_id)|\text{Csk})) = \delta(H(H(h_s||\text{VCC}_id)||\text{Csk}))
\]
looks random to the adversary as long as he doesn’t know \(h_s\). The latter is not possible as argued in the previous hybrid.

Finally, we can see that all the information the Attacker can get during the game is independent from the expected value \(\text{VCC}^{id}\), and therefore \(\Pr\{S^B_7\}\) is the probability of guessing it at random, which is \(\frac{1}{|A_{\text{vcc}}|}\). We conclude that
\[
\text{VCC}_{\text{adv}}[A] = \epsilon_{\text{dl-kdf}} + \epsilon_{\text{skdl}} + \epsilon_{\text{ind-cpa}^s} + \epsilon_{\text{ddh}} + \epsilon_{\text{i-ddh}} + \epsilon_{\text{prf}} + 2 \cdot \epsilon_{\text{sound-dl}} + \epsilon_{\text{ind-cpa}^s},
\]
which is negligible.

### 6.3 Game C: vote cast

The Attacker confirms a ballot without the collaboration of the voter. As in previous games, the Attacker can register honest and corrupt voters. For corrupt voters, the Attacker is provided with the whole of the registration information, including the Ballot Casting Key. However, for honest voters \(id \in \mathcal{I}_h\), the Attacker is given the information required to cast a vote, but not to confirm it (i.e it is not given \(\text{BCK}^{id}\)). The public registration information from all the voters is, as in previous cases, published by the challenger on a bulletin board. The Attacker has also the ability to cast and confirm votes by sending vote cast and vote confirmation queries that are processed by the challenger.

The objective of the Attacker in this game is, for any registered honest voter, to successfully registered a vote/ballot on her behalf without the participation of that voter. The admissible attacker must do so without leaving a trace that would be detected by any of the Tests 1-4 of Definition.\(^2\)

*In particular the adversary cannot ask \(\text{CCR}_1\) to compute Choice Return Codes for more than one ballot. Importantly, if \(\text{VCC}_id^*\) is the target voter, then the entry \((\text{VCC}_id^*, b) \in \text{bb}\) must be a valid confirmed ballot.*

49
In Game C the Challenger proceeds as follows:

**Configuration:**

**Challenger:**
- Compute $EL_{pk} = EL^{(1)}_{pk} \cdot EL^{(2)}_{pk}$
- Compute $(C_{ax}, VCCs_{pk}, VCCs_{ax}) \leftarrow \text{Setup}.SDM(1^\lambda)$
- Initialize the empty voter lists $ID_h, ID_c, ID = (ID_h \cup ID_c)$
- Post $(EL_{pk}, VCCs_{pk}, ID)$ in BB
- Keep $(EL^{(1)}_{ax}, C_{ax}, VCCs_{ax})$

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

- **OChallenger_RegisterHonest(id):**
  - If generated id $\in ID$, stop and return ⊥. Otherwise:
  - Add id to $ID_h$
  - Select $x_{id} \leftarrow \{v_1, \ldots, v_n\}$
  - Generate $(SVK_{id}, VC_{id}, K_{id}, VCC_{id}, BCK_{id}, VCC_{id}, \{v_j, CC_{id}^j\}_{j=1}^n)$ in interaction with the Attacker
  - Post (id, $VC_{id}, K_{id}, VCC_{id}, map_{id}$) in BB
  - Provide $SVK_{id}, x_{id}$ to the Attacker
  - Keep $\{v_i, CC_{id}^i\}_{i=1}^n, VCC_{id}$

- **OChallenger_RegisterCorrupt(id):**
  - If generated id $\in ID$, stop and return ⊥. Otherwise:
  - Add id to $ID_c$
  - Generate $(SVK_{id}, VC_{id}, K_{id}, VCC_{id}, BCK_{id}, VCC_{id}, \{v_j, CC_{id}^j\}_{j=1}^n)$ in interaction with the Attacker
  - Post (id, $VC_{id}, K_{id}, VCC_{id}, map_{id}$) in BB
  - Provide $(SVK_{id}, VC_{id}, BCK_{id}, VCC_{id}, \{v_i, CC_{id}^i\}_{i=1}^n)$ to the Attacker

**Voting:** The Attacker obtains replies to vote casting and vote confirmation queries by asking the Challenger to run the CreateVote and Confirm protocols in interaction with the Attacker:

- **OChallenger_Vote(id, b):**
  - run $\text{ProcessVote}(pk^{CCR}, VC_{id}, b)$. If result is 1:
    - compute $\sigma \leftarrow \text{CreateCC}(b, C_{ax}, map_{id})$
      - in interaction with the Attacker
    - add $(VC_{id}, b)$ to $bb^{CCR}_C$
    - send $\sigma$
  - Else send ⊥

- **OChallenger_Confirm(id, CM_{id}):**
  - Run $(\mu, S_{\mu}) \leftarrow \text{ProcessConfirm}(bb, VC_{id}, CM_{id}, C_{ax}, VCCs_{pk})$, in interaction with the Attacker
- add (VC_id, (CM_id)kc_id) entry to BBCCR
- if (µ, S, µ) ̸= ⊥: add (VC_id, µ, S, µ) entry in BBCCR
Send µ

Attacker: Is in control or has additional knowledge of the following:

- Computes key pairs (EL(2)pk, EL(2)sk) and (K′2, k′2); (K2id, k2id) and (Kc2id, kc2id)
for every voter id ∈ ID
- Receives Csk and CMtable
- Controls boxes bb, bbCCR

At the end of the game, the Attacker provides (VC∗id, µ∗).

6.3.1 Security reduction of game C

The development of the game C starts with a reduction from the ability of the Attacker to confirm a vote, to be able to guess the right confirmation message.
After that, it proceeds similarly than in the case of games A and B, showing that the information the Attacker gets is independent from the confirmation message it has to guess. Finally, we will see that the best chance for the Attacker is to try to guess the valid Ballot Casting Key from which the confirmation message is constructed.
Throughout the hybrids we denote with S_Ci, in game C.i as the event that ∃(VC∗id, µ∗, S, µ) ∈ BB for VC_id ∈ IDh.

Game C.1 First we proceed with a game transition based on the soundness of one of the NIZK proof systems. Let this be the game in which the adversary is restricted to submit to the ProcessVote algorithm from game A.1 ballots whose encrypted options and encrypted partial Choice Return Codes have been computed over the same voting option x.

We claim that |Pr{SC1} − Pr{SC0}| = ε_{sound_{EqEnc}},
where ε_{sound_{EqEnc}} is the advantage of an efficient algorithm against the soundness of the NIZKP scheme for the equality of encryptions (which is negligible assuming the properties of NIZKPs).

Game C.2 This transition is also based on indistinguishability: let this be the game in which for honest voters id ∈ IDh, the Challenger changes the computation of k1id = δ(k′1, VC_id) and kc1id := δ(k′1||VC_id||ConfirmStr), where (k′1, k′1) ← Gen_e(G) and (Kc1, k′1) ← Gen_e(G), to k1id, kc1id ← Z_q. Still, the Challenger will keep a list {(id, k1id, kc1id)}id∈IDh of created keys internally, so that they can be provided whenever needed.

We claim that |Pr{SC2} − Pr{SC1}| = ε_{dl-kdf},
where $\epsilon_{dl-kdf}$ is the KDF-advantage of an efficient KDF distinguisher algorithm, that tries to distinguish $K, Kc \xrightarrow{\$} K_{sp}$ from $K \xleftarrow{\$} k_{id} = \delta(k', \underline{VC}_{id}), Kc \xleftarrow{\$} kc_{id} := \delta(k'_{1}||\underline{VC}_{id}||\underline{ConfirmStr})$, with $k'_{1} = (g)^{k_{1}}$ known.

**Game C.3** This transition is again based on indistinguishability: let this be the game in which for honest voters $id \in ID_{h}$, the Challenger changes for $j = 1, \ldots, n$ the computation of $(v_{j})^{k_{id}}$ in $\text{Setup}.SDM(\cdot)$ and $\text{CreateCC}(\cdot)$ by replacing them with $R_{j} \xrightarrow{\$} G$, and the computation of $(CM^{id})^{kc_{id}}$ by replacing it with $S^{id} \xrightarrow{\$} G$. Still, the Challenger will keep the list of $\{R_{j}^{id}\}_{j=1}^{n}$ and $S^{id}$ for every $id \in ID_{h}$, so that they can be used whenever needed.

We claim that

$$\left| \Pr\{S^{C}_{3}\} - \Pr\{S^{C}_{2}\} \right| = \epsilon_{ddh},$$

where $\epsilon_{ddh}$ is the DDH-advantage of an efficient distinguisher algorithm of Diffie-Hellman tuples in $G$, that tries to distinguish tuples $(g, K_{id}^{1}, v_{j}, (v_{j})^{k_{id}})$ from $(g, K_{id}^{1}, R_{j}^{id})$ for $R_{j}^{id} \xrightarrow{\$} G$ and $j = 1, \ldots, n$ or $(g, K_{id}^{1}, CM^{id}, (CM^{id})^{k_{id}})$ from $(g, K_{id}^{1}, CM^{id}, S^{id})$ for $S^{id} \xrightarrow{\$} G$.

**Game C.4** This transition is based on failure: let this be the game in which for honest voters $id \in ID_{h}$ for which the Attacker confirms a vote, namely there exists a unique entry $(\underline{VC}_{id}, b, \underline{VCC}_{id}, S_{VCC}_{id}) \in bb$ is a valid confirmed ballot. In particular this implies that $\underline{VCC}_{id} = H(\underline{VC}_{id}||\underline{CM^{id}}^{id} \cdot (CM^{id})^{k_{id}})$.

We claim that

$$\left| \Pr\{S^{C}_{4}\} - \Pr\{S^{C}_{3}\} \right| = \epsilon_{cm},$$

where $\epsilon_{cm}$ is the probability of breaking the collision resistance of the hash function $H$.

**Game C.5** This transition is also based on indistinguishability: let this be the game in which for honest voters $id \in ID_{h}$, the Challenger associates a random Choice Return Code $CC^{id}_{id}$ and a random Vote Cast Code $VCC^{id}_{id}$ that are independent of the values appearing in the ciphertexts of the mapping table $\text{map}^{id}$. Still, the Challenger will keep the list of Choice Return Codes and the Vote Cast Code (and corresponding signature) internally, so that they can be provided whenever needed.

We claim that

$$\left| \Pr\{S^{C}_{5}\} - \Pr\{S^{C}_{4}\} \right| = \epsilon_{enc},$$

where $\epsilon_{enc}$ is the advantage of an efficient algorithm that distinguishes the output of $\text{Enc}^{\circ}$ from a random function, which is negligible considering that $\text{Enc}^{\circ}$ can be seen as a pseudo-random function.
We can see that all the information the Attacker can get during the game is independent from the expected value \( \text{CM}^{id} \) it has to generate in order to successfully confirm a vote, and therefore \( \Pr(S_C^7) \) is the probability of guessing it. Given that the Attacker knows the Verification Card private key \( k_{id} \), \( \text{CM}^{id} = ((\text{BCK}^{id})^2)^{k_{id}} \) and \( A_{\text{bck}} \subset G \), the best chance for the Attacker is to guess the valid Ballot Casting Key, for which the probability is \( \frac{1}{A_{\text{bck}}} \). We can conclude that

\[
\text{BCK}_{\text{adv}}[A] = \epsilon_{\text{cm}} + \epsilon_{\text{enc}} + \epsilon_{\text{dl-kdf}} + \epsilon_{\text{ddh}} + \epsilon_{\text{sound}}\text{EqEnc},
\]

which is negligible given the assumption that the voters’ keys are uniformly sampled from the key space, the order of the group \( G \) is larger than the number of voters, cryptographic hash functions are collision-resistant, the used symmetric encryption scheme is indistinguishable from a random distribution without knowledge of the required key, the underlying hash function of the pseudorandom function, and given the properties of hash functions in the random oracle.

6.4 Game D: incorrect tally

The objective of the Attacker in this game is to obtain that, for any registered honest voter \( id \), her ballot \( (\text{VC}_{id}^*, b^*) \in bb \) is tallied and contributes to the election result with a voting choice \( v^* \neq x_{id}^* \) different from her intended voting choice \( x_{id}^* \).

Importantly, an admissible Attacker must store in the ballot box the ballots that originate from honest voters and which have been confirmed. Additionally, an admissible Attacker cannot delete ballots from the ballot box that originate from honest voters and which have been confirmed. Any of these two situations this would be detected by an Auditor by running Tests 1-5 of Definition 3.

In Game 0, the Challenger proceeds as follows:

**Configuration:**

**Challenger:**

Compute \( \text{EL}_{pk} = \text{EL}_{pk}^{(1)} \cdot \text{EL}_{pk}^{(2)} \)

Compute \( (c_{sk}, \text{VCC}_{pk}, \text{VCC}_{sk}) \leftarrow \text{SetupSDM}(1^A) \)

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

Post \( (\text{EL}_{pk}, \text{VCC}_{pk}, ID) \) in BB

Keep \( (\text{EL}_{sk}^{(1)}, c_{sk}, \text{VCC}_{sk}) \)

**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 \( \bot \). Otherwise:

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

Select \( x_{id} \leftarrow \{v_1, \ldots, v_n\} \)
Generate (SVK\textsubscript{id}, VC\textsubscript{id}, K\textsubscript{id}, VCks\textsubscript{id}, BCK\textsubscript{id}, VCC\textsubscript{id}, \{v_j, CC\textsubscript{j}\}_{j=1}^n) in interaction with the Attacker

Post (id, VC\textsubscript{id}, K\textsubscript{id}, VCks\textsubscript{id}, map\textsubscript{id}) in BB

Provide SVK\textsubscript{id}, x\textsubscript{id} to the Attacker

Keep \{v_i, CC\textsubscript{i}\}_{i=1}^n, VCC\textsubscript{id}

\textbf{OChallenger\_RegisterCorrupt}(id):

If generated id ∈ ID, stop and return ⊥. Otherwise:

Add id to ID

Generate (SVK\textsubscript{id}, VC\textsubscript{id}, K\textsubscript{id}, VCks\textsubscript{id}, BCK\textsubscript{id}, VCC\textsubscript{id}, \{v_j, CC\textsubscript{j}\}_{j=1}^n) in interaction with the Attacker

Post (id, VC\textsubscript{id}, K\textsubscript{id}, VCks\textsubscript{id}, map\textsubscript{id}) in BB

Provide (SVK\textsubscript{id}, VC\textsubscript{id}, BCK\textsubscript{id}, VCC\textsubscript{id}, \{v_i, CC\textsubscript{i}\}_{i=1}^n) to the Attacker

\textbf{Voting:} The Attacker generates votes running the CreateVote algorithm, or confirm votes running ProcessConfirm, and can ask the Challenger to run the following algorithms several times:

\textbf{OChallenger\_Vote}(id, b):

- run ProcessVote(pk\textsubscript{CCR}, VC\textsubscript{id}, b). If result is 1:

  - compute \(σ ← \text{CreateCC}(b, C\textsubscript{sk}, \text{map}^\text{id})\)

  in interaction with the Attacker

  - add (VC\textsubscript{id}, b) to bb

  - send \(σ\textsubscript{id}\)

Else send ⊥

\textbf{OChallenger\_Confirm}(id, CM\textsubscript{id})

- Run \((μ, S_μ) ← \text{ProcessConfirm}(bb, VC\textsubscript{id}, CM\textsubscript{id}, C\textsubscript{sk}, VCC\textsubscript{pk}), \text{in interaction with the Attacker}\)

- add (VC\textsubscript{id}, (CM\textsubscript{id}, C\textsubscript{sk})) entry to BB\textsubscript{CCR}

- if \((μ, S_μ) \neq ⊥\): add (VC\textsubscript{id}, μ, S_μ) entry in BB,

  Send \(μ\)

\textbf{Tally:} The Attacker obtains from the Challenger the result of sequentially running Cleansing followed by MixDec\textsubscript{1} on input the adversarial controlled box BB. Notwithstanding, the admissible attacker is restricted in how it handles these boxes, since their content needs to be consistent with BB\textsubscript{CCR}, which are maintained by the honest component CCR\textsubscript{1}. More precisely the attacker obtains the output of

\[
\left(L^s, L_1, P_s, P_{dec}\right) := \text{MixDec}_1\left(EL_{\text{sk}}^{(1)}, \text{Cleansing}(EL_{\text{pk}}, \{pk_j^{(j)}\}_{j=1}^\psi, VCC\textsubscript{pk}, BB)\right)
\]

\textbf{Attacker:} Is in control or has additional knowledge of the following:

\textbf{Attacker:}

- Computes key pairs (EL\textsubscript{pk}, EL\textsubscript{sk}) and (K_2, K'_2); key pairs (K_1^2, K_id) and (Kc_1^2, Kc_id) for every voter id ∈ ID

  - Receives C\textsubscript{sk} and C\textsubscript{table}

  - Controls boxes BB, BB\textsubscript{CCR}
At the end of the game, the Attacker outputs \((BB, r, \Pi)\) with \(\Pi = (L, L_1, P_m, P_{\text{dec}}, L_1, L_v, P_B, P_{\text{dec}})\).

Define \(S_0^A\) as the event that \(\exists (VC_{id}^*, b^*) \in bb\) for \(VC_{id}^* \in ID_h\) such that:

**Property 1** \(\sigma_{id}^* = CC_{id}^*\)

**Property 2** \(\exists (VC_{id}^*, \mu, S_{\mu}) \in BB\) s.t. Verif\((VCC_{\text{pk}}, \mu, S_{\mu}) = 1\)
- let \(L_B := \{b_1, \ldots, b_\ell\}\) the list of ballots accepted by Cleansing except for the target ballot \(b^*\), i.e. \(b^* \notin L_B\)
- let \(L^v_B := \{\text{dec}(b_1), \ldots, \text{dec}(b_\ell)\}\)

**Property 3** \(L_v \neq L^v_B \cup \{x_{id}^*\}\)

If event \(S_0^A\) occurs that implies that the attacker has managed to change the contribution to the final result by voter with Verification Card identifier \(VC_{id}^*\) from \(x_{id}^*\) to \(v^*\), where \(v^* := L_v \setminus L^v_B\).

We define the universal verifiability advantage of an adversary \(A\) as:

\[
UV_{\text{adv}}[A] = \Pr\{S_0^A\}
\]

### 6.4.1 Security reduction of game D

The security reduction starts by using the fact that sVote enjoys security against type 1-3 attacks, and therefore an admissible attacker cannot fool a voter into confirming a ballot that does not contain the voter’s intended voting choice. It proceeds by showing that since cleansing is deterministic, and that mixing and partial decryption proofs are sound, every confirmed ballot will contribute to the election result with their corresponding encrypted voting choice. In the case of honest voters, that is their intended voting choice.

**GAME D.1** The adversary is not allowed to change the choice \(x_{id}\) for any honest voter \(id \in ID_h\).

Define \(S_1^A\) as the event that \(\exists (VC_{id}^*, b^*) \in bb\) for \(id \in ID_h\) for which the Attacker succeeds to undetectably change \(id\)’s contribution to the final result, namely the attacker’s output and the of the game satisfies Properties 1,2,3. We claim that

\[
|\Pr\{S_0^A\} - \Pr\{S_1^A\}| = CC_{\text{adv}}[A] + VCC_{\text{adv}}[A] + BCK_{\text{adv}}[A],
\]

which is negligible given the analysis of the previous games.

---

\(^{17}\)For ease of notation, we obviate the fact that Dec works over the ciphertext \(c\) inside \(b\).
GAME D.2  Then we proceed with a game transition based on the soundness of one of the NIZK proof systems. Let this be the game in which the adversary is restricted to output a correctly computed shuffled list of ciphertexts $L_1$.

Define $S^A_2$ as the event that $\exists (V_{C_{id}}, b^*) \in bb$ for $id \in ID_h$ for which the Attacker succeeds to undetectably change $id$’s contribution to the final result, namely the attacker’s output and the of the game satisfies Properties 1,2,3. We claim that

$$|\Pr\{S^A_1\} - \Pr\{S^A_2\}| = \epsilon_{\text{sound}_\text{VMix}},$$

where $\epsilon_{\text{sound}_\text{VMix}}$ is the advantage of an efficient algorithm against the soundness of the NIZKP scheme for verifiable mixing (which is negligible assuming the properties of NIZKPs).

GAME D.3  Then we proceed with a game transition based on the soundness of one of the NIZK proof systems. Let this be the game in which the adversary is restricted to output a correctly computed list of partial decryptions $L_v$.

Define $S^A_3$ as the event that $\exists (V_{C_{id}}, b^*) \in bb$ for $id \in ID_h$ for which the Attacker succeeds to undetectably change $id$’s contribution to the final result, namely the attacker’s output and the of the game satisfies Properties 1,2,3. We claim that

$$|\Pr\{S_3^A\} - \Pr\{S_2^A\}| = \epsilon_{\text{sound}_\text{EqDL}},$$

where $\epsilon_{\text{sound}_\text{EqDL}}$ is the advantage of an efficient algorithm against the soundness of the NIZKP scheme for correct decryption (which is negligible assuming the properties of NIZKPs).

At this stage we can claim that $\Pr\{S_3^A\} = \epsilon$, where $\epsilon$ is negligible, since the Attacker has not modified the honest voters’ choices before they are stored in the form of ballots in $bb$, the Attacker is not removing any honest voters’ confirmed ballots, and the Attacker is publishing in $L_v$ the correct decryption of the stored and confirmed ballots.

Finally the advantage of an admissible Attacker against the Universal Verifiability of sVote is negligible, since

$$\text{UVAdv}[A] = \text{IVAdv}[A] + \epsilon_{\text{sound}_\text{VMix}} + \epsilon_{\text{sound}_\text{EqDL}} + \epsilon$$

7 Conclusions

7.1 Final remarks and improvements

Our analysis shows that the four attack types described in Section 5.4 are unlikely to occur. Security against the first three type of attacks model cast-as-intended and recorded-as-cast properties. The last type of attack models counted-as-recorded property.
Regarding the first and third types of attacks, leveraging on the entropy of the output of the return codes control component allowed us to argue that the (encrypted) mapping table does not leak meaningful information on the choice codes corresponding to the selected voting options, nor in the vote cast code of the voter.

Security against the second type of attacks tells us that an adversary even with the knowledge of the ballot casting key cannot discard the vote of an honest user. Without this guarantee it is not possible to tell apart the case where the voter types incorrectly the ballot casting key from the case where the adversary tampers with it. We have not considered the case in which the adversary tampers with the start voting key because this would be detected (either the content of the encrypted ballot will not be well-formed or we are in a similar situation as for type 1 attacks).

The fourth type of attacks leverages on the difficulty of carrying out the first three attacks so that it can be assumed the ballot box after the voting phase is in good state. From this, an argument for correctness of the result is given due to the presence of verifiable mixnets.

We point out that the assurance stemming from our security definitions does not rule out the existence of other attacks. However, it rules out those considered in the ordinance, as argued in Section 7.2. There are several alternatives to further strengthen sVote, including distributing the mapping table across control components (avoiding a centralized setup), and augmenting the entropy of the return control components with respect to the computation of the long choice codes. These improvements may allow to move checks from auditors to honest voters, among other advantages.

7.2 VEleS security objectives

We believe that our interpretation of the security objectives fits with what we have proven. To see this, we review the ordinance, and in Table 8 provide a many-to-one relationship between the security objectives of the ordinance and our defined cryptographic properties of Section 5.

7.2.1 Individual verifiability

The ordinance states the following requirements for authorizations for more than 30 percent, but less than 50 percent, of the cantonal electorate.

**VEleS Art. 4.2:** For the purpose of individual verification, voters must receive proof that the server system has registered the vote as it was entered by the voter on the user platform as being in conformity with the system. [...] 

It follows that only the voter can detect these type of attacks. Therefore, neither the voting client nor the system should be able to forge a proof to the voter that convinces him that his vote was registered correctly by the trustworthy part of the system. Since the voter has a central role in individual verifiability, we need to formulate a number of assumptions about a honest voters behavior.

ASSUMPTIONS ON HONEST VOTERS BEHAVIOR. Our assumption is that the voter receives a voting card containing the following information over a trusted
<table>
<thead>
<tr>
<th>Security Objective</th>
<th>Technical annex of VEleS</th>
<th>Our interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1 attack (Section 5.1)</td>
<td>misappropriating a vote before its registration by the trustworthy part of the system</td>
<td>The voter must receive the short vote cast code that corresponds to his ballot casting key and that was derived by the trustworthy part of the system</td>
</tr>
<tr>
<td>Type 2 attack (Section 5.2)</td>
<td>misappropriating a vote before its registration by the trustworthy part of the system</td>
<td>The voter must receive the short vote cast code that corresponds to his ballot casting key and that was derived by the trustworthy part of the system</td>
</tr>
<tr>
<td>Type 3 attack (Section 5.3)</td>
<td>casting a vote</td>
<td>It shall not be possible to cast a vote in a voters name without knowledge of the start voting key and the ballot casting key</td>
</tr>
<tr>
<td>Type 4 attack (Section 5.4)</td>
<td>Auditors receive proof that the result has been ascertained correctly</td>
<td>Correctness of tally</td>
</tr>
<tr>
<td>Type 1 attack (Section 5.1)</td>
<td>changing a vote cast in conformity with the system, the casting of which has been registered by the trustworthy part of the system</td>
<td>The auditor must be able to check that a vote was not altered after its registration by the trustworthy part of the system</td>
</tr>
<tr>
<td>Type 2 attack (Section 5.2)</td>
<td>misappropriating a vote cast in conformity with the system, the casting of which has been registered by the trustworthy part of the system</td>
<td>The auditor must be able to check that all votes that were correctly confirmed are included in the final tally.</td>
</tr>
<tr>
<td>Type 3 attack (Section 5.3)</td>
<td>inserting a vote</td>
<td>The auditor must be able to check that no votes that were cast that were not confirmed using a valid votes ballot casting key</td>
</tr>
</tbody>
</table>

Table 8: Identifying VEleS security objectives with our defined cryptographic properties

channel (postal mail). All codes are different for each voter and the attacker cannot learn these codes in advance from an honest voter. However, it is possible that dishonest voters collude with the attacker and divulge their codes to the attacker.

Based on an honest voters behavior, our security objectives closely follow the technical annex of the ordinance.
VEleS Annex, Chapter 4.3: The attacker is unable to achieve the following objectives under the trust assumptions for the complete verifiability of the protocol without a voter or a trustworthy auditor being highly likely to recognise that an attack has been carried out:

- changing a vote before its registration by the trustworthy part of the system
- misappropriating a vote before its registration by the trustworthy part of the system
- casting a vote
- [...]

Further extending the requirements of Art 4., Art. 5.3 states additional requirements for authorizations up to a 100 per cent of the cantonal electorate.

VEleS Art 5.3: For individual verification the following requirements must be met in addition to those in Article 4:

a. The proof must also confirm to the voters that the data relevant to universal verification has reached the trustworthy part of the system (para. 6).

b. Voters must be able to request proof after the electronic voting system is closed that the trustworthy part of the system has not already registered a vote cast using their client-sided authentication measure.

c. The substantiveness of the proof must not depend on the trustworthiness of the entire server system. It may however be based on the trustworthiness of the trustworthy part of the system.

Based on our interpretation, Art 5.3.a is covered by the security against type 1 attacks. We understand art 5.3.b as an extension of security against type 3 attacks, for instance when the voter is not sure that somebody used his ballot casting key in his name in case he did not want to participate in the vote. This requirement is covered procedurally, since the voter can ask the auditor to check if the trustworthy part of the system registered a vote for his specific voting card id. This is in line with the technical annex which states the following:

VEleS Annex. Supplementary provision 4.4.2: At the end the voting process, the public board displays the result and the proof that the result is correct, which has been issued in the context of universal verifiability. The voters could accordingly assume the role of "auditors" in the spirit of maximum possible transparency. Various risk considerations, connected not least with the practice orientated assumption that user platforms need to be regarded as un-trustworthy, can be used as arguments that the data for the trustworthy part of the system that is relevant to verifiability should not be published without restriction. It is therefore permitted to make the data available to a restricted group of auditors.

Art 5.3.c follows naturally from the trust assumptions of universal verifiability in the technical annex.
7.2.2 Universal verifiability

In contrast to individual verifiability, universal verifiability puts the role of the verifier into the hands of an auditor. Other approaches in academia go even further and would allow everyone to assume the role of this auditor. However, as stated in the supplementary provision 4.4.2, the group of auditors can be restricted.

**VEleS, Annex. Chapter 4.3:** The attacker is unable to achieve the following objectives under the trust assumptions for the complete verifiability of the protocol without a voter or a trustworthy auditor being highly likely to recognise that an attack has been carried out:

- [...]  
- changing a vote cast in conformity with the system, the casting of which has been registered by the trustworthy part of the system
- misappropriating a vote cast in conformity with the system, the casting of which has been registered by the trustworthy part of the system
- inserting a vote

**Assumptions on Auditors.** We explicitly do not cover the process of selecting and designating auditors and their technical aids. However, we assume that at least one honest auditor is verifying the results using a trustworthy technical aid.

**VEleS Chapter 4.3:** In addition, it is assumed that at least one trustworthy auditor will review the proof with the assistance of a trustworthy technical aid.

**VEleS Annex: Supplementary provision 4.4.1. (On Art. 5 para. 1):** The use of auditors serves transparency. Voters should be able to assume that auditors in the event of any doubt would point out irregularities. However the groups which people who act as auditors should be recruited from is deliberately left open.

Of course, the result of the tally should be also correct.

**VEleS Art. 5.4:** For universal verification, the auditors receive proof that the result has been ascertained correctly.

**References**


© Copyright notice:

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.