



HAL
open science

Code voting: when simplicity meets security

Véronique Cortier, Alexandre Debant, Florian Moser

► **To cite this version:**

Véronique Cortier, Alexandre Debant, Florian Moser. Code voting: when simplicity meets security. ESORICS 2024, Sep 2024, Bydgoszcz, Poland. hal-04627733

HAL Id: hal-04627733

<https://inria.hal.science/hal-04627733>

Submitted on 27 Jun 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Code voting: when simplicity meets security

Véronique Cortier, Alexandre Debant, and Florian Moser

CNRS, Université de Lorraine, INRIA, France

`veronique.cortier,alexandre.debant,florian.moser@inria.fr`

Abstract. The Swiss Chancellery has developed highly demanding requirements in terms of security for electronic voting. A few protocols have been proposed that meet the requirements.

We propose a very simple protocol that satisfies the Swiss requirements while achieving an additional property: secrecy against a dishonest voting device, thanks to code voting. One key feature of our protocol is to use very short codes (typically one or two digits), so that it can be easily used by voters. Moreover, it only relies on standard primitives. Using the tool ProVerif, we formally prove vote privacy and end-to-end verifiability under the trust model of the Swiss Chancellery, assuming a dishonest voting device.

Keywords: Internet Voting · Code Voting · Privacy · Verifiability · Switzerland

1 Introduction

Electronic voting is used in several countries, such as Australia [8], Estonia [21], Switzerland [38], or France [16]. It aims at preserving the same properties as traditional paper-ballots elections and in particular two main properties, namely vote privacy and verifiability: the result of the election should reflect the votes of all legitimate voters. Verifiability is often divided into several sub-properties such as cast-as-intended, individual, universal, and eligibility verifiability [28].

All these properties come with trust assumptions on the parties involved in the protocol. Such assumptions vary a lot depending on the voting system. For example, some systems such as Helios [1], Belenios [15], or Civitas [12] consider a fully dishonest voting server but assume a public Bulletin Board that anyone can see and trust [22]. Another approach followed in Estonia [21] and Switzerland [38] is to maintain a private bulletin board, distributed over several online servers, among which only one is trusted. Another important element of a voting system is the voting device, which is assumed to be trusted or untrusted depending on the protocol.

This work benefited from funding managed by the French National Research Agency under the France 2030 program with the reference ANR-22-PECY-0006. It was also partly supported by the ANR Chair IA ASAP (ANR-20-CHIA-0024) with support from the region Grand Est, France.

Dishonest voting device. Securing Internet voting against a dishonest voting device is a challenging task. A versatile approach is the Benaloh’s challenge [2], which allows a voter to check that their vote has been properly encrypted. This approach is, however, hard to use in practice [32]. In the Estonian system [21] or with CAISED [35], voters are offered the possibility to use a second device to control the behavior of their main voting device. In Switzerland, the voters are given a voting sheet before the start of the election that contains a *return code* for each voting option [38]. Then, when a voter selects some voting option v on their voting device, they should be displayed the corresponding return code. This mechanism guarantees their vote has been correctly transmitted since the voting device cannot guess the return codes.

However, all these solutions only focus on cast-as-intended: even if the voting device is compromised, the voter’s *intended* vote should be correctly encrypted and sent to the voting server. None of these approaches protect against vote privacy: a dishonest voting device immediately learns the voter’s vote since the vote is entered in clear by the voter. This yields an uncomfortable situation where a device is trusted for one key property (vote privacy) but not the other one (verifiability). A few systems overcome this issue using *code voting* introduced by Chaum [10]: a voter no longer enters their vote in clear but transmits a voting code instead, so the voting device does not learn the vote.

Swiss context. The Swiss Chancellery [18] is highly demanding in terms of security for Internet voting. The specification should be fully public, as well as the code of the system. Public scrutiny is encouraged through bug bounty programs. In the Swiss approach, the voting sheets distributed to voters are generated by a trusted **Setup Component**, in collaboration with **Control Components**. When a voter votes, their **Voting Device** encrypts their vote and sends the ciphertext to the control components, which intuitively play the role of a distributed voting server. The control components collaboratively compute the corresponding return code, which is sent back to the voter, who will confirm their vote only if the return code is correct.

As defined by the Swiss Chancellery, verifiability should hold even if the attacker controls the network, all the online control components but one, and the voting device. For vote privacy, the trust assumptions are similar, except the voting device is now trusted. Our goal is to suppress this assumption.

Our contribution. Our main contribution is a voting protocol that satisfies the security requirements of the Swiss Chancellery (vote privacy and verifiability) without assuming a trusted voting device, even for vote privacy. A recent proposal [19] aims at the same goal, relying on BLS signatures [7] and long voting codes. Comparing to it, we show that it is possible to obtain the same security guarantees using a radically different approach, with very short voting codes of typically 1 or 2 digits. Interestingly, our protocol is very simple in terms of design and cryptographic primitives. Besides homomorphic encryption or verifiable mixnets for the tally, it only uses very standard cryptographic primitives, such

as signatures and hashes. We believe our protocol can easily be implemented and maintained, even by programmers with a light background in cryptography.

In addition to removing the trust in the voting device, we mitigate one of the strongest trust assumptions of the Swiss context: the trusted setup component. This component is trusted both for vote privacy and verifiability. We keep this assumption but our protocol is designed in such a way that the computations of SC are minimal and easy to audit. In particular, it does not need to generate any randomness. Hence, it is easy to check that the output corresponds to the input, with no additional information or bias introduced through the randomness. So, if the SC misbehaves, it can be caught through random audits. Moreover, a malicious setup could use well-chosen randomness to leak information. In our design, we remove the use of randomness in the core design of the setup, hence limiting the risk of covert channels. The use of randomness is limited to external components, e.g., to implement authenticated channels with signatures.

Finally, we provide a full security proof of both vote privacy and verifiability using the ProVerif [5] tool, for an arbitrary number of voters and an arbitrary number of elections. For verifiability, we rely on a recent framework [13], that allows to directly prove end-to-end verifiability, that is, proving that the final result accounts for all the votes of honest voters, plus at most one vote per dishonest voters. Directly verifying end-to-end verifiability avoids the need to consider several sub-properties (e.g., cast-as-intended, recorded-as-cast), at the risk of missing an important property [30].

Outline of the paper. In the next section, we further discuss other voting protocols based on code voting. Our protocol is presented in Section 3, and we explain its security model and proof in ProVerif in Section 4. Some concluding remarks are in Section 5.

2 Related work

In the Swiss context, two main protocols have been proposed, namely CHVote [20] and the one developed by Swiss Post [38]. However, both assume that the voting device is trusted for vote privacy. In the literature, a few systems have been proposed to achieve vote privacy against a dishonest voting device, using code voting, following the seminal work of Chaum [10]. Among them, only one, HKL [19], has been designed specifically for the Swiss context, i.e., with the aim to satisfy the Swiss trust assumptions. Others have been designed in a more general context. Hence, their different trust assumptions led to different design compromises. We discuss here the systems that aim at protecting vote privacy against a dishonest voting device.

Comparison to HKL protocol [19]. In the HKL protocol, voters receive a voting sheet in which each voting option is associated to a QR-code, that encodes the encrypted vote. Thanks to a verifiable mixnet during the setup, the relation between the ciphertext and its plain vote remains private for anyone without

access to the voting sheet. BLS signatures [7], based on pairing, are then used to prevent ballot stuffing.

One major drawback of this approach is its usability. Indeed, in Switzerland, a voter typically needs to answer about 6 questions, and for several questions, 3 options can be selected (yes, no, abstain). This would amount to selecting 6 QR-codes out of 18 for a typical election. But some elections have more questions (e.g., 13 questions in Zurich in autumn 2022 [26,37], therefore amounting to 39 QR-codes) and some elections even offer voters to select *several* options for each question [25]. Moreover, because the camera used to scan the QR-code(s) shall not see the other cryptographic data printed onto the voting sheet for security purposes, the voter is subject to additional constraints when manipulating the voting sheet. Recent studies [40,33] conclude that QR-codes are usable for electronic voting when voters select exactly *one* voting option in total, but it seems cumbersome to use such a technology in elections where voters have to select more than a few options.

Compared to HKL, our protocol is QR-code-free (exception, optionally, for the first step). Voters only need to enter a short code for each question: if a question has 15 possible answers, then the voter simply needs to enter a number between 1 and 15. The association between these small codes and the corresponding encrypted votes is committed by the setup component and audited by auditors before the start of the election. The return codes guarantee that the voting device cannot modify the voting code without being detected. Studies have also explored the usability of entering voting codes [33,29], and even though the voting codes were longer than in our proposal (8 numbers in [33], and 3 characters in [29]), reached acceptable usability scores.

Comparison to non-Swiss specific systems. Outside the Swiss context, BeleniosVS [14] follows the same approach as HKL: the code is simply the encrypted vote. However, BeleniosVS assumes in addition that voters can access a public Bulletin Board. Unfortunately, existing techniques to implement a secure public Bulletin Board [22] typically assume a signature mechanism. Since a voter cannot check the validity of a signature by themselves, they need to rely on their voting device to do so. Assuming the voting device dishonest as in our setting, it may simply lie to the voter. Therefore, an approach based on a public Bulletin Board does not seem to be a realistic option under our trust assumptions.

Pretty Good Democracy (PGD) [36] uses long, hard to guess, codes for each voting choice, that intuitively points to the right encrypted vote in the data prepared during setup. In PGD, if the attacker knows another code, it may swap votes. While long codes may be acceptable when only one vote is expressed, this does not fit well in the Swiss context. Furthermore, the control components (the trustees in the PGD setting) need to be trusted for verifiability, as they may lie on how they shifted the ciphertexts during the setup. Pretty Understandable Democracy (PUD) [9] also uses pointers to ciphertexts. The voter needs to enter several codes for each voting option, received from different voting sheets, one from each server (these servers intuitively correspond to the Swiss control components). This may again be very cumbersome in practice. The VeryVote

system [24] shifts the trust from the voting device to the voting server that (intuitively) encrypts the vote for the voter, but in a verifiable manner. The authors later extend their approach to remove the trust in the voting server [23], but at the cost of a secure hardware token on the voter side, hence trusting again some part of the voting device.

All these systems use either voting codes which need to be resistant to random guessing or encode directly the ciphertext or signatures. This results into long voting codes in the order of 128 bits or even more. To our knowledge, the only scheme that uses short voting codes, hence in the order of number of candidates, is D-Demos [11]. As in our proposal, the voting code simply points to the right ciphertext, and the number of needed voting codes is simply the number of voting options. D-DEMOS uses a double-ballot system, where the voter chooses which ballot to audit and which one to use for voting, which leads to a slightly more complex system. Moreover, D-Demos assumes a public Bulletin Board, which tends to discard it for real-world usages under our trust assumptions as discussed above. Finally, their setup component has to generate a lot of cryptographic, randomized, material, which may be hard to audit.

Table 1 displays a comparison of the existing code voting systems that do not assume any trust in the voting device. We first compare their security (privacy and verifiability) for different trust models. We highlight the trust model of the Swiss Chancellery and our trust model. For the sake of comparison, we also display the two main systems that have been designed for the Swiss context, namely Swiss Post [38] and CHVote [20]. None of them achieve privacy against an insecure voting device but in contrast, they achieve a better verifiability level than code voting when the setup component is dishonest but the voting device is trusted (a scenario outside the Swiss trust model).

We also compare the systems in terms of usability and auditability. We pinpoint which systems use short voting codes and which are “vote and go”; that is, a voter simply needs to interact with the system during the voting phase, and does not need to come back afterward, e.g., to check that their vote has been counted. In terms of auditability and simplicity, we highlight systems that use a deterministic setup and do not use cryptography on the voting device. We also remark that most papers focus on the design of the system and do not accompany it with a security proof of verifiability, vote privacy, or both.

| | Swiss Setting | | | | Voting Codes Systems | | | | |
|--------------------------------|---------------|----------------|-------------|----------|----------------------|------------|-----------------|--------------------|------------------|
| | SPP [38] | CHVote [20] | HKL [19] | Proposal | PGD [36] | PUD [9] | D-DEMOS [11] | BeleniosVS [14] | VeryVote [24] |
| Trust for Vote privacy | | | | | | | | | |
| <i>1/n CC, SC, VD</i> | • | • | • | • | • | • | • | • | |
| <i>1/n CC, SC</i> | | | • | • | • | • | • | • | |
| 1/n CC, VD | | | | | | | | • | |
| 1/n CC | | | | | | | | | |
| Trust for Verifiability | | | | | | | | | |
| 1/n CC, SC, VD | • | • | • | • | | • | • | • | • |
| <i>1/n CC, SC</i> | • | • | • | • | | • | • | • | • |
| 1/n CC, VD | • | • | | | | | | | • |
| 1/n CC | | | | | | | | | • |
| Security Proof | | | | | | | | | |
| Privacy | • | | | • | | | • | • | |
| Verifiability | • | • | | • | | | • | • | |
| Construction | | | | | | | | | |
| Short Voting Codes | | | | • | | | • | | |
| Deterministic SC | | • | • | • | • | | | | |
| Vote and go | • | • | • | • | • | • | | | |
| No Crypto on VD | | | • | • | • | • | • | | |

Table 1. Comparison of code voting systems.

Block one and two describe scenarios with different trust assumptions for vote privacy and verifiability corresponding to the control components CC, the setup component SC and the voting device VD. The more components need to be trusted, the less secure is the protocol. In *italic* the Swiss trust model, and in gray our stronger trust model (we no longer assume a trusted voting device for privacy). **•**: these protocols assume a public bulletin board that voters can securely access. This implicitly assumes some trusted device. In PGD, all control components need to be honest for verifiability, as they may lie on how they shifted the ciphertexts during the setup. In VeryVote, the server is trusted to encrypt the candidate for the voter, which is a stronger assumption than the ones we consider here.

3 Protocol

We first recall the voting infrastructure of the Swiss context. We then describe our protocol and its main security properties.

3.1 The Swiss context

The Swiss Chancellery has precisely defined the main actors of a voting system in the Swiss context, with associated communication channels. We summarize the given model as follows.

The **Setup Component** is active during the setup. It computes the voting material and distributes it to the voters through postal mail. It also prepares the material used to tally the votes.

The **Control Components** are entities that are active during all the voting phases. They collectively generate the randomness used during the setting, register the votes and compute return codes during the voting phase, and gather the encrypted ballots for the tally phase.

The **Decryption Authorities** are in charge of generating the public key of the election and decrypting during the tally. Typically, a threshold of t out of n authorities is necessary to decrypt.

The **Voting Device** interacts with the **Voter** to help them cast a vote over the Internet.

The voting device does not communicate directly with the control components. Instead, all communications go through a **Voting Server** that is fully untrusted and simply sends the messages back and forth from the voting devices and the control components. Therefore, we will omit its description in what follows.

Trust assumptions. We consider the trust model as defined by the Swiss Chancellery [18].

The network is fully under the control of the attacker. In particular, the security of the protocol cannot rely on TLS. However, postal mail from the setup component to the voter is assumed to be trusted: the attacker can neither read nor tamper with the voting sheets sent to the voters.

The setup component is assumed to be fully trusted. It is only active during the setup phase and does not need to be online. The voting material (the voting sheets) is sent over secure channels to the voters (in practice, by Post). The control components are typically a set of 4 entities, among which only one is trusted for verifiability and vote privacy. The decryption authorities are trusted as a whole, with a threshold of trusted authorities. Finally, the voting device is untrusted for verifiability but trusted for privacy in the trust model of the Chancellery. This is, however, the point where our trust assumptions differ: we no longer assume the voting device to be honest for privacy.

Moreover, the Chancellery's model places a lot of trust in the setup component since it is trusted both for privacy and verifiability. This is something that we mitigate in our protocol by proposing a fully deterministic process, easy to audit.


3.2 Voter's view

To introduce the protocol, we first present the voter's view. The voter is given a ballot sheet with a QR code to login, and codes to cast and confirm their vote. See Figure 1 for an example using realistic parameters (e.g., length of codes). First, the voter scans the QR code to login ① with her identifier Id . The QR code contains the URL of the voting server, reducing the chance that the voter ends up on the wrong page. Then, the voter enters the short voting codes c_1, \dots, c_n of the voting choices they intend to vote for ②. The voting codes are sent to the voting system, which responds with return codes rc_1, \dots, rc_n , one for each selected voting option (codes for blank votes shall be provided). If and only if these codes match what is printed on the voting sheet ②, the voter is instructed to confirm the vote ③ by sending the authentication code ac . This is the only long secret value that authenticates the voter. The voter finally receives a finalization code fc , that guarantees them that their vote has been counted.

The voting client essentially needs to forward values to the voting system and back to the user. It does not need to encrypt or sign the vote. The voting client may guide the user with basic validation (e.g., ensuring the voting code entered for the first question is between 1 and 3).

① **Start**

Scan to open the election platform and login:



Then you can cast your Vote ②.

② **Vote**

Do you want to accept the popular initiative
"Lower Voting Age to Sixteen Years"

| | Vote | Verify |
|----------------|------|--------|
| Yes | 3 | ADEH |
| No | 1 | KYSO |
| Abstain | 2 | KKHA |

Do you want to accept the federal law
"Electronic Identification (eID Act)"

| | Vote | Verify |
|----------------|------|--------|
| Yes | 4 | FSUM |
| No | 6 | PINQ |
| Abstain | 5 | UAQM |

Enter Vote numbers to cast the vote.
Afterwards, check Verify letters are shown.

If Verify letters match, then Confirm ③.
If Verify letters do not match, abort.

③ **Confirm**

If Verify letters match, enter Authentication:

F28Q nAJcj 6W3f C5CXt vwfa

Afterwards, check Finalization is shown:

JQHS UMKQ

If Finalization shown, your vote is stored.
If Finalization is not shown, call 114.

Fig. 1. An example of a voting sheet using realistic parameters. ① contains a QR code with the voter id. ② contains the voting codes and their respective return code. ③ contains the authentication code and the respective finalization.

3.3 Protocol description

The protocol is divided into three phases. In the setup phase, all parties receive the required cryptographic material to run the voting and tally phase. In the voting phase, the voter submits their vote and performs their individual verifiability checks. In the final tally phase, the votes are decrypted and counted. For simplicity, we present our protocol for an election with one question only, but it can easily be extended to several questions (e.g., the voting sheet presented in Figure 1 has two questions).

Notations. We denote \mathbb{Z}_n or $[1, n]$ the set of integers between 1 and n and \oplus_n the addition modulo n . If n is obvious from the context, we may omit it. The set of all possible permutations of \mathbb{Z}_n is denoted S_n . The composition of two permutations π_1 and π_2 is denoted $\pi_1 \circ \pi_2$.

The set of voting options is denoted \mathcal{V} and is of size n_v . The correspondence between voting options and voting codes will be denoted by a substitution (or a table) σ from \mathcal{V} to \mathbb{Z}_{n_v} . The application of a substitution σ to an element $v \in \mathcal{V}$ is denoted $\sigma[v]$. We assume an *initial correspondence* σ_0 , chosen publicly by the authorities, that maps each voting option to some integer in \mathbb{Z}_n . This substitution is fixed for all voters. For example, if $\mathcal{V} = \{a, b, c\}$, the initial substitution could be $\sigma_0 = \{a \rightarrow 1, b \rightarrow 2, c \rightarrow 3\}$.

Cryptographic primitives. Our protocol relies on three standard primitives.

- a hash function `hash`,
- a signature scheme. We denote $\sigma = \text{sign}(sk, m, r)$ the signature with signing key sk , message m and randomness r . Similarly, we denote $\text{verify}(pk, m, \sigma)$ the verification applied to public key pk , message m and signature σ . We write $\text{sign}(sk, m)$ when the source of randomness r is clear from the context.
- an encryption scheme compatible with a homomorphic tally or a mixnet, such as the one used in Helios [1]. We denote $E = \text{enc}(pk, m, r)$ for the encryption of message m with public key pk and randomness r .

Setup Before the setup, we assume that the authorities have agreed on a list of identifiers \mathbf{Id} , one for each eligible voter. The identifiers are used to keep track of the cryptographic material per voter, and they are not assumed to be secret.

We also assume given n_{rc} the number of return codes, n_a the number of authentication codes, and n_f the number of finalization codes. For return and finalization codes, the adversary has a single try to convince the voter, hence the values do not need to be hard to guess. Typically, n_{rc} and n_f will be of medium size (e.g., 4 digits, that is $n_{rc} = n_f = 10^4$). Regarding authentication codes, corrupted parties will have access to their hashes, so this set must be large enough for picked values to be hard to brute-force. Hence typically $n_a = 2^{128}$. Lastly, we assume given \mathbb{Z}_{n_r} , the set from which encryption randomness is drawn.

Key generation. The decryption authorities jointly generate the election public key pk_E , using a Distributed Key Generation, for example, the one used in Belenios [15] or ElectionGuard [3]. They each have a share of the decryption key such that k out of n authorities can decrypt.

We also assume that each control component CC_i has a signing key sk_i , with associated public verification key vk_i .

Preparation of the voting sheets. During the setup, the setup component is in charge of sending to each voter:

- their identifier Id ,
- a correspondence σ between voting options and voting codes,
- a table t that maps each voting option to a return code,
- an authentication code ac ,
- a finalization code fc .

To obtain a deterministic setup component, each control component generates for each $Id \in \mathbf{Id}$:

- a randomly chosen correspondence π_i between voting options and voting codes,
- a table t_i that maps each voting option to a randomly chosen return code,
- a table r_i that maps each voting code to a randomly chosen encryption randomness,
- a random authentication code ac_i ,
- a random finalization code fc_i .

Then, the setup component simply combines the contributions of all m control components:

- $\sigma = \sigma_0 \circ \pi$ where $\pi = \circ_{i=1}^m \pi_i$: the initial correspondence σ_0 is permuted using the combination π of the permutations π_i chosen by the CC_i ,
- $t = \Sigma_{i=1}^m t_i$,
- $r = \oplus_{i=1}^m r_i$,
- $ac = \Sigma_{i=1}^m ac_i$,
- $fc = \Sigma_{i=1}^m fc_i$,

where \oplus denotes the bitwise exclusive-OR operator, and $\Sigma_{i=1}^m x_i = x_1 + \dots + x_m \bmod N$ with N defined from the context (e.g., $N = n_{rc}$ when computing t , $N = n_a$ for ac , and $N = n_f$ for fc). By abuse of notation, when operands are tables, the operator is applied for each index.

Additionally, the setup returns to the control components:

- $h = \text{hash}(ac)$, the hashed authentication code,
- a table te from the voting codes to the corresponding (encrypted) voting options

$$\{c \rightarrow \text{enc}(pk_E, \sigma^{-1}[c], r[c]) \mid c \in \mathbb{Z}_{n_v}\}$$

using the XORed randomness received from the CC_i .

The setup phase is depicted in Figure 2, as well as the remainder of the protocol.

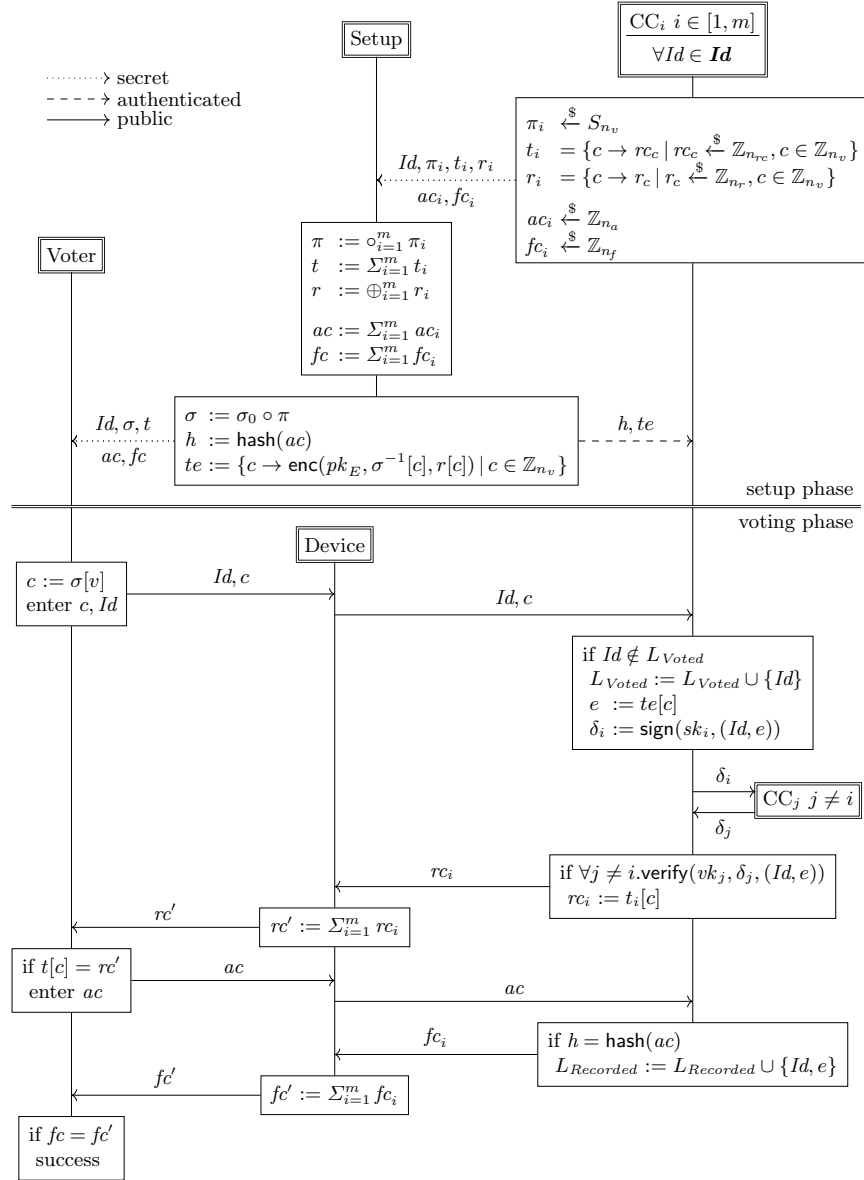


Fig. 2. Protocol.

Audit of the setup component The setup component needs to be fully trusted. Notably, it could switch the vote and verification codes of plain votes, such that a different voting option is voted for than the one intended. We introduce an audit procedure to improve the detection of a misbehaving setup component.

To make an audit of the setup component practical, we ensure it only runs deterministic algorithms. Note that if at least one control component performs honestly, the adversary has no advantage in guessing the ballots.

To audit the setup component, we first note that the identifiers in \mathbf{Id} are not tied to voters. They are just numbers (possibly randomly chosen). Hence, when preparing the election, the authorities simply add n additional entries to \mathbf{Id} , for n chosen in such a way that if n voters are audited, the risk is sufficiently minimized. After the setup component finishes execution, each control component i chooses some subset of voters to audit $\mathbf{Id}_A^{(i)} \subset \mathbf{Id}$ such that $|\cup_i \mathbf{Id}_A^{(i)}| = n$. Any corresponding Id can no longer be used to cast votes, but instead, the control components publish all values they generated for this Id . The control components then assert that the voting sheet, h , and te have been generated correctly. Finally, the remaining voting sheets are randomly assigned to voters, to prevent targeted attacks on specific Ids from the setup and control components.

Voting phase The voter cast their vote by interacting with the control components.

- The voters first select the voting code c corresponding to their vote v and they simply enter c together with their identifier Id to their voting device.
- Each control component checks that they did not receive Id yet, signs Id together with the ciphertext e corresponding to c in their table te , and sends this signature to the other control components. Once the control component has received corresponding signatures back from all other control components, it sends their share rc_i of the return code.
- The voting device combines the return codes and displays $rc = \sum_{i=1}^m rc_i$ to the voter. If the voter agrees, they send their authentication code ac .
- Each control component checks that the code ac is valid by comparing it to its hashed value h . They now record the ciphertext e and Id in the list $L_{Recorded}$ of recorded votes and they each send their share fc_i of the finalization code.
- The voting device displays the combination of the fc_i to the voter, which ends the voting phase.

At any time, if the voter detects some inconsistency, they should contact the voting authorities, who then offer other means of voting.

Multiple voting options. For simplicity, we considered the selection of one voting option only. In case the voter has to select k out of n options, then the voter simply sends the k corresponding voting codes, the control components return k return codes and the protocol runs accordingly. In case the voter may select up to k out of n options, we use the same approach as for other Swiss systems:

we add k blank options so that the voter always selects k options, possibly with some blank ones. The voter hence has to send k voting codes and check k return codes, even if they vote for less than k actual candidates.

Several questions can easily be supported using different code voting names. For example, the voting sheet provided in Figure 1 corresponds to an election with two questions, of 3 voting options each. How to adapt our protocol to cover more complex electoral rules, where a ballot may be further constrained (e.g., when at least one non-blank candidate needs to be selected) is left as future work.

Tally phase At the end of the voting phase, each control component has a list of recorded votes $L_{Recorded}$ which need to be tallied. Ideally, all control components should have the same $L_{Recorded}$ in their local state. However, dishonest control components might have added, modified, or dropped entries in their local $L_{Recorded}$, hence the control components need to agree on their list $L_{Recorded}$ of encrypted votes.

Agreement procedure. To establish L_{Agreed} , each control component CC_i sends $L_{Recorded}$ with the “proof” for each entry (that is, ac and signatures over (Id, e)) to all other control components CC_j . Each control component then defines L_{Agreed} to be $L_{Recorded}$, in which they add any e such that

- they received a valid signature $\text{sign}(sk_j, (Id, e))$ from each CC_j ,
- h is associated to Id and $h = \text{hash}(ac)$.

After this step, all control components agree, or they can detect who is misbehaving.

Tally procedure. After establishing L_{Agreed} , the decryption authorities execute the tally procedure with the ciphertext given by L_{Agreed} as input. We leave this step abstract as our protocol can support both a homomorphic tally [1] or a mixnet-based tally [17]. In both cases, the decryption authorities provide verifiable proof of correct tally.

3.4 Security claims

Our protocol satisfies all the security requirements of the Swiss Chancellery, even if the voting device is dishonest. More precisely, our protocol guarantees vote privacy, even when the attacker controls the voting device, all but one control component, and less than t decryption authorities. Our protocol also ensures verifiability, even when the attacker controls the voting device and all but one control component. In particular, it guarantees cast-as-intended and vote privacy against a dishonest voting device.

3.5 Design Rationale

In this section, we give an intuitive explanation why our protocol guarantees the verifiability and privacy in our trust model. We provide formal proofs in Section 4.

Verifiability A correct return code rc in the first roundtrip guarantees that the voting code c cast by the voter Id has been received by all control components. rc can only be reconstructed if all control components publish their rc_i . The honest control component will only publish the corresponding rc_i if c is indeed the first vote of that voter, and it has received signatures over (Id, e) for $e = te[c]$ of all other control components. We assume the adversary unable to guess rc_i or forge signatures.

A correct finalization code fc in the second roundtrip guarantees that only votes are tallied for which the first roundtrip was successful. fc can only be reconstructed if all control components publish their fc_i . The honest control component will only publish fc_i if it has seen ac , which is in turn only published by the voter when rc of the previous roundtrip was correct. We assume the adversary unable to guess fc_i or ac .

Finally, for the vote e to enter the tally, there are two conditions: there must exist a signature over (Id, e) from all control components, and ac must be known. If the voter has seen a correct finalization code, the honest control component is in possession of these values, therefore guarantees inclusion of e in the tally. Furthermore, the fact that ac is known implies that rc was valid, which in turn implies that the corresponding c has been received by all control components. The verifiable tally procedure then ensures the vote is not modified during the tally.

Privacy We first collect which values learned by the adversary depend on the selected voting option. When the voter has chosen their plain vote v , they then map it to the voting code $c := \sigma[v]$, hence c clearly depends on v . The other values that depend on v are the corresponding partial return codes $rc_i := t_i[c]$, and the encrypted vote $e := te[c]$. All the other values, including those from other voters, are independent (i.e. remain the same if the voter would chose a different v').

The adversary cannot relate c to its corresponding v , as it does not know σ , and cannot reconstruct π (as π_i of the honest control component is unknown to the adversary). The adversary cannot learn additional information from rc_i , as they were randomly chosen. Further, the adversary is unable to decrypt e as at least one decryption authority is assumed honest. The privacy-preserving tally procedure then ensures that the adversary does not learn anything about v , apart from the result of the election.

4 Security analysis

In order to study the security of our protocol, we conducted a formal security analysis using the automatic prover ProVerif.

4.1 ProVerif in a nutshell

ProVerif [5,6] is an automatic tool designed to conduct security analyses of cryptographic protocols. Protocols are modeled in a symbolic model: messages are abstracted with terms modeling an idealized cryptography. For instance, as soon as the attacker does not know the decryption key, a ciphertext is perfectly hiding its plain message. Protocol roles are modeled with processes that define how messages are exchanged between the participants and which checks are done upon receiving a message. Processes are made of commands that describe the different actions of a participant: $\text{in}(c, x)$ models the input of a message x on a channel c , $\text{out}(c, u)$ the output of message u on channel c , $\text{if } b \text{ then } P \text{ else } Q$ models a test. Other commands exist to model protocol states, parallel execution or replication of roles, fresh term generation, and more. Finally, ProVerif models a Dolev-Yao attacker controlling the network: the attacker learns all messages exchanged between participants on public channels and can modify them, block them, or forge new ones. This attacker usually over-approximates the behavior of a concrete attacker. It can also behave as a dishonest participant of the protocol as soon as it is given the corresponding secret material.

Regarding security properties, ProVerif supports both reachability and equivalence properties. In the context of electronic voting, reachability properties are used in particular to model verifiability: for all executions, the outcome of the election corresponds to the intended votes. Equivalence properties model that an attacker cannot distinguish between two different scenarios. This is typically used to model vote privacy.

Other tools, such as Tamarin [34] or CPSA [31], have also been developed for the analysis of security protocols. We chose ProVerif because it has already been successfully applied to voting protocols in the Swiss setting [39] and because it offers a framework [13] for proving end-to-end verifiability. Unfortunately, such a framework does not exist to prove vote privacy. Hence, we defined our own model, combining the protocol roles previously defined in the verifiability framework.

4.2 Modeling the protocol and security properties

Protocol. The first step to prove the security of the protocol is to describe its different roles (voter, setup component and control component). Because the voting device is always assumed dishonest in the scenarios we want to consider, we do not model it. Instead, this role will be under the control of the implicit attacker of the symbolic model. In the same way, we model only one control component, the others being assumed dishonest, too. Moreover, for the sake of simplicity, we model a setup component that interacts with only one dishonest control component, which mimics the behavior of all the dishonest ones.


```

1      (* voter who casts and verifies *)
2      let Voter(id: id_t, voting_choice: voting_choice_t) =
3
4          (* read out voting material *)
5          get VoteState(=id, =voting_choice, vote_code, return_code) in
6          get AuthenticateState(=id, authentication_code, finalization_code) in
7
8          (* cast the vote and raise Voted event *)
9          out(c, (id, vote_code));
10         event Voted(id, voting_choice);
11
12         (* if the return code matches, then authenticate the vote *)
13         in(c, return_code': xor_t);
14         if return_code' = return_code then (
15             out(c, authentication_code);
16
17             (* if the finalization code matches, then raise Verified event *)
18             in(c, finalization_code': xor_t);
19             if finalization_code' = finalization_code then (
20                 event Verified(id, voting_choice)
21             )
22         )
23     .

```

Fig. 3. The voters' process.

We provide an excerpt of the voter role in Figure 3. We model the voting sheet as a table filled by the setup component, which the voter can read from using `get` in order to retrieve the appropriate voting code. Then, the voter sends the voting code together with their id. The voter expects back a return code, and if this return code is as expected, the voter sends the authentication code. The voter again expects back a finalization code, and again checks if this finalization code is as expected. After voting and after verification, the voter raises events which we use in the queries of our verifiability proof.

Verifiability. The framework developed by Cheval et al. in [13] aims at proving end-to-end verifiability: an electronic voting protocol satisfies end-to-end verifiability if for all execution traces,

$$result = V_{HV} \uplus V_{HNV} \uplus D$$

where *result* is the multiset of tallied votes, V_{HV} is the multiset of votes of honest voters who verify, V_{HNV} is a sub-multiset of the votes cast by honest voters who do not verify, and D contains at most one vote per dishonest voter.

Instantiating this definition to return code protocols, we say that a voter has voted if they have cast their voting code and received the expected return code. Moreover, we say that a voter has verified if they have sent the confirmation code and received the expected finalization code. Informally, end-to-end verifiability guarantees that the attacker cannot modify the plain vote cast by the voter, nor confirm a vote on their behalf. Moreover, the attacker cannot stuff the ballot box by impersonating voters or casting multiple ballots in their name.

Vote privacy. In [27], Kremer et al. define vote privacy as an equivalence property: an electronic voting protocol satisfies vote privacy if an attacker is not able to distinguish whether Alice votes for 0 and Bob for 1, or conversely, i.e.

$$\text{Voter}(\text{alice}, 0) \mid \text{Voter}(\text{bob}, 1) \mid P \approx \text{Voter}(\text{alice}, 1) \mid \text{Voter}(\text{bob}, 0) \mid P$$

where P denotes the roles of the protocol other than Alice’s and Bob’s, such as the setup component or the control component. For the sake of generality, we prove vote privacy assuming that P models an arbitrary number of dishonest voters.

Modeling choices. Even if our modeling is close to the protocol description provided in Section 3, the ProVerif model differs in a few aspects.

Exclusive-OR and sum. ProVerif does not support the exclusive-or operator. It does not support addition between variables either¹ Therefore, we had to abstract these two operators to make the analysis possible. We chose to abstract it through a unique uninterpreted function of symbol `xor_combine(·, ·)` that combines two shares into a unique element. We then provide to the attacker some minimal deduction capabilities, common to both exclusive-OR and sum, using the two following reduction rules:

$$\begin{aligned} \text{getL}(\text{xor_combine}(x, y), y) &= x \\ \text{getR}(\text{xor_combine}(x, y), x) &= y. \end{aligned}$$

Unfortunately, this modeling does not reflect the algebraic properties of the operators. Going beyond this limitation in ProVerif is still an open problem and out of scope of this paper. Despite this limitation, our analysis provides a substantial level of confidence in the security of the protocol.

Permutations. In the protocol, permutations are used to keep the voter’s choice private. Permutations are encountered in other contexts, such as mixnets, but unfortunately, no faithful model exists at the moment for mixnets, they are instead abstracted away in existing analysis (see e.g., [14,4]). Hence we cannot rely on these approaches. In our protocol, we use the fact that permutations are independent from a voter to another. They can therefore be soundly abstracted as follows to prove vote privacy: first, do not apply any permutation for voting material associated to dishonest voters. Then, when forging Alice’s and Bob’s voting material, apply the following permutation: assuming that the permutation σ is applied to the plain votes 0 and 1 on the left side of the equivalence, apply the permutation $\sigma' = \{0 \mapsto 1, 1 \mapsto 0\} \circ \sigma$ on the right side. If σ is a randomly sampled permutation, then σ' is a randomly sampled permutation too. We can thus soundly apply the first transformation and abstract away the exact permutation σ . It makes ProVerif succeed in proving equivalence in the scenarios we consider.

¹ Since v2.01, ProVerif supports natural numbers. However, the operator $+$ can only be used between a variable and a constant or two constants, which is too restrictive to model our protocol.

4.3 Results

The ProVerif tool confirmed the expected security of the protocol: both privacy and verifiability hold as soon as the setup component and one control component are trusted. The analysis took about 224s for verifiability and about 11s for vote privacy. The corresponding ProVerif files are available as supplementary material. Experiments have been conducted on a standard laptop (MacBook Pro - M2 Pro - 32GB RAM) using the ProVerif development branch `improved_scope_lemma`² as requested by the framework used to prove verifiability [13].

5 Conclusion

We have provided a simple voting protocol that satisfies both verifiability and privacy, even against a dishonest device. Its simplicity makes it easy to implement, with no scalability issue related to cryptographic computations. Our proposal also demonstrates that the Swiss requirements can be fulfilled with a minimal protocol, providing some conceptual answer on the “minimal” building blocks needed in this setting. It also opens the discussion on which other properties are also needed. For example, we noticed that the SwissPost protocol still guarantees privacy when only the setup component and one auditor are dishonest, and all other components, including the communication channels and the voting server, are honest. This is due to the fact that the voting device encrypts the vote. In contrast, our code-voting protocol does not offer vote privacy against an honest-but-curious setup component that has access to the ballot box. This highlights the fact that the Swiss Chancellery, in charge of defining the required properties, may want to further strengthen its requirements. For example, it may want to require that an honest-but-curious setup component may not, alone, break vote privacy. On the other hand, this will make the design of code voting schemes much more complex. Hence, on the contrary, another desirable requirement could be vote privacy against a voting device.

References

1. Adida, B.: Helios: Web-based Open-Audit Voting. In: USENIX (2008)
2. Benaloh, J.: Simple verifiable elections. EVT (2006)
3. Benaloh, J., Naehrig, M.: ElectionGuard. Design Specification Version 2.0.0. Tech. rep., Microsoft Research (2023), https://github.com/microsoft/electionguard/releases/download/v2.0/EG_Spec_2_0.pdf
4. Bernhard, D., Cortier, V., Gaudry, P., Turuani, M., Warinschi, B.: Verifiability Analysis of CHVote. Cryptology ePrint Archive (2018), <https://eprint.iacr.org/2018/1052>
5. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: Computer Security Foundations Workshop (CSFW) (2001)

² https://gitlab.inria.fr/bblanche/proverif/-/tree/improved_scope_lemma, with the most recent commit at the time of writing being a7ff60b1.

6. Blanchet, B., Cheval, V., Cortier, V.: ProVerif with Lemmas, Induction, Fast Subsumption, and Much More. In: Symposium on Security and Privacy (SP) (2022)
7. Boneh, D., Lynn, B., Shacham, H.: Short Signatures from the Weil Pairing. In: Advances in Cryptology (2001)
8. Brightwell, I., Cucurull, J., Galindo, D., Guasch, S.: An Overview of the iVote 2015 Voting System. New South Wales Electoral Commission, Australia (2015), <https://elections.nsw.gov.au/getmedia/4279ab0e-5db0-451e-9e3d-87d81ed82d9c/overview-of-the-ivote-2015-voting-system.pdf>
9. Budurushi, J., Neumann, S., Olembo, M.M., Volkamer, M.: Pretty Understandable Democracy - A Secure and Understandable Internet Voting Scheme. In: International Conference on Availability, Reliability and Security (ARES) (2013)
10. Chaum, D.: Sure Vote: Technical Overview. In: Workshop on trustworthy elections (WOTE) (2001), <http://www.vote.caltech.edu/wote01/pdfs/surevote.pdf>
11. Chondros, N., Zhang, B., Zacharias, T., Diamantopoulos, P., Maneas, S., Patsonakis, C., Delis, A., Kiayias, A., Roussopoulos, M.: D-DEMOS: A distributed, end-to-end verifiable, internet voting system. In: International Conference on Distributed Computing Systems (ICDCS) (2016)
12. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: Symposium on Security and Privacy (SP) (2008)
13. Cortier, V., Debant, A., Cheval, V.: Election Verifiability with ProVerif. In: Computer Security Foundations Symposium (CSF) (2023)
14. Cortier, V., Filipiak, A., Lallemand, J.: BeleniosVS: Secrecy and verifiability against a corrupted voting device. In: Computer Security Foundations Symposium (CSF) (2019)
15. Cortier, V., Gaudry, P., Glondu, S.: Belenios: a simple private and verifiable electronic voting system. Foundations of Security, Protocols, and Equational Reasoning: Essays Dedicated to Catherine A. Meadows (2019)
16. Debant, A., Hirschi, L.: Reversing, breaking, and fixing the french legislative election e-voting protocol. In: USENIX Security Symposium (2023)
17. Douglas Wikström: Open Verificatum. <https://www.verificatum.org/>
18. Federal Chancellery: Federal Chancellery Ordinance on Electronic Voting. <https://www.fedlex.admin.ch/eli/cc/2022/336/en> (December 2013)
19. Haenni, R., Koenig, R.E., Dubuis, E.: Private Internet Voting on Untrusted Voting Devices. In: International Conference on Financial Cryptography and Data Security (2023)
20. Haenni, R., Koenig, R.E., Locher, P., Dubuis, E.: CHVote Protocol Specification. Cryptology ePrint Archive (2017), <https://eprint.iacr.org/2017/325.pdf>
21. Heiberg, S., Willemsen, J.: Verifiable internet voting in Estonia. In: International Conference on Electronic Voting (EVOTE) (2014)
22. Hirschi, L., Schmid, L., Basin, D.A.: Fixing the Achilles Heel of E-Voting: The Bulletin Board. In: Computer Security Foundations Symposium (CSF) (2021)
23. Joaquim, R., Ferreira, P., Ribeiro, C.: EVIV: An end-to-end verifiable Internet voting system. Computers & Security (2013)
24. Joaquim, R., Ribeiro, C., Ferreira, P.: Veryvote: A voter verifiable code voting system. In: International Conference on E-Voting and Identity (VOTE-ID) (2009)
25. Kanton Zürich: National- und Ständeratswahlen. <https://www.zh.ch/de/politik-staat/wahlen-abstimmungen/national-staenderatswahlen.html>
26. Kanton Zürich: Abstimmungsvorlagen vom 25. September 2022. <https://www.zh.ch/de/news-uebersicht/medienmitteilungen/2022/06/abstimmungsvorlagen-vom-25-september-2022.html> (September 2022)

27. Kremer, S., Ryan, M.: Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. In: European Symposium on Programming (ESOP) (2005)
28. Kremer, S., Ryan, M.D., Smyth, B.: Election verifiability in electronic voting protocols. In: Symposium on Research in Computer Security (2010)
29. Kulyk, O., Neumann, S., Budurushi, J., Volkamer, M.: Nothing comes for free: How much usability can you sacrifice for security? IEEE Security & Privacy (2017)
30. Küsters, R., Truderung, T., Vogt, A.: Clash attacks on the verifiability of e-voting systems. In: Symposium on Security and Privacy (SP) (2012)
31. Liskov, M.D., Ramsdell, J.D., Guttman, J.D., Rowe, P.D.: The Cryptographic Protocol Shapes Analyzer: A Manual. The MITRE Corporation (2016)
32. Marky, K., Kulyk, O., Renaud, K., Volkamer, M.: What Did I Really Vote For? On the Usability of Verifiable E-Voting Schemes. In: Conference on human factors in computing systems (CHI) (2018)
33. Marky, K., Schmitz, M., Lange, F., Mühlhäuser, M.: Usability of code voting modalities. In: Conference on Human Factors in Computing Systems, CHI 2019. ACM (2019)
34. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The Tamarin Prover for the Symbolic Analysis of Security Protocols. In: International Conference on Computer Aided Verification (CAV) (2013)
35. Müller, J., Truderung, T.: A protocol for cast-as-intended verifiability with a second device. In: International Joint Conference on Electronic Voting (EVOTE) (2023)
36. Ryan, P.Y., Teague, V.: Pretty good democracy. In: International Workshop on Security Protocols (2009)
37. Stadt Zürich: 25. September 2022: fünf kommunale Sachvorlagen . https://www.stadt-zuerich.ch/portal/de/index/politik_u_recht/abstimmungen_u_wahlen/archiv_abstimmungen/vergangene_termine/220925.html (September 2022)
38. Swiss Post: Swiss Post Voting System: System Specification. Version 1.1.1. Tech. rep., Swiss Post (October 2022), https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/afbd7555b3f87092596a5203f022c00dcc0fd390/System/System_Specification.pdf
39. Swiss Post: Symbolic Analysis of the Swiss Post Voting System. Version 1.6.0.0. <https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/20d8d707/Symbolic-models> (2024)
40. Volkamer, M., Kulyk, O., Ludwig, J., Fuhrberg, N.: Increasing security without decreasing usability: A comparison of various verifiable voting systems. In: Symposium on Usable Privacy and Security (SOUPS) (2022)