Private Internet Voting on Untrusted Voting Devices

Rolf Haenni and Reto E. Koenig and Philipp Locher

BFH, Bern University of Applied Sciences, Biel, Switzerland
{rolf.haenni,reto.koenig,philipp.locher}@bfh.ch

Abstract. This paper introduces a new cryptographic Internet voting protocol, which offers individual verifiability and vote privacy even on completely untrustworthy voting devices. The core idea is to minimize the voting client to a simple device capable of scanning a QR code, sending its content to the web server of the included URL, and displaying a response message to the voter. Today, QR code scanners are pre-installed into mobile devices, and users are familiar to using them for many different purposes. By reducing the voting client to an existing functionality of the voters' personal device, the implementation of the protocol is simplified significantly compared to other protocols. The protocol itself can be seen as a variant of Chaum's code voting scheme with an elegant solution to the problem of distributing the trust to multiple authorities. The approach is based on BLS signatures and verifiable mix-nets. It relies on trustworthy printing and mailing services during the election setup.

1 Introduction

A major challenge in Internet voting is the untrusted voting client. Current protocol proposals and implementations, for example the ones used in Switzerland, provide a mechanism for achieving cast-as-intended and recorded-as-cast verifiability simultaneously by returning verification codes to the voter for each submitted vote. Since the malicious voting client cannot predict these codes, manipulation attempts can be detected by the voter with high probability. However, vote privacy remains an unsolved problem on the client, because from performing the encryption of the submitted votes, the malicious voting client learns both the selected voting options and the encryption randomization. The latter serves as a receipt to prove the selected voting options to a third party. With respect to vote privacy, such systems are therefore only secure under the strong assumption of a fully trustworthy voting client.

In this paper, we introduce a fundamentally different Internet voting protocol, which offers vote privacy even in the presence of a completely untrustworthy voting client. The difference comes from minimizing the role assigned to the voting client as far as possible, in such a way that it not even learns the selected voting options. We only require the voting client of being capable of scanning a QR code, sending its content to the web server of the included URL, and displaying a response message to the voter. Since this is a pre-installed functionality of today's mobile devices, and because users are familiar to scanning QR code from different applications, we can almost entirely eliminate the complexity of designing and implementing a reliable voting web client.

Our proposal is motivated by the current situation in Switzerland, where legislation is very strict with respect to E2E-verifiability, but quite lax with respect to the privacy problem on the voting client. We interpret this unbalanced regulation as an unwanted compromise in the absence of a better solution. Our paper aims at providing a compelling solution for solving this conflict. Another motivation for our protocol is the usability study in [6], which showed that voters are able to use code voting with QR codes in an E2E-verifiable system. The paper ended with an invitation to the community for presenting corresponding cryptographic protocols. This paper is a first response to the this invitation.

1.1 Election Model and Voting Procedure

If we take the Swiss context as a reference point for designing the protocol, we cannot assume that voters possess any type of electronic credentials, which they could possibly use for authentication when submitting the ballot. On the other hand, we can assume that reliable printing and postal services are available for sending such credentials to the voters prior to an election, possibly together with other voting materials. We can also assume that with multiple simultaneous m_i-out-of-n_i elections, all possible election uses cases can be covered adequately (see discussion in [4, Section 2.2]). For reason of simplicity, we restrict ourselves in this paper to a single m-out-of-n election, but our protocol is flexible enough to support the general case with just a few additional steps.

From the voter's perspective, the election procedure starts with receiving a letter from the electoral office over postal mail. In an m-out-of-n election, this letter includes n different *voting cards*, one for each voting option, and a single *confirmation card*. Note that the optimal design of these cards is still an open question [6], but for security reasons, it is important that not all elements printed on these cards are visible at the same time. In the example shown in Figure 1, we consider the case of a referendum (1-out-of-2 election) with two voting cards and one confirmation card. The QR codes are printed on the back of these cards to avoid making them visible together with the verification and participation codes.

Given these cards, the voting procedure is now relatively simple and efficient. It only assumes that the voter is capable of scanning QR codes and sending the scanned data to the included URL, for example using a mobile phone with a working Internet connection. Then the voting procedure consists of five steps:

1. Select m voting cards (or less to submit blank votes).
2. Scan the *voting QR codes* on the back of each selected voting card.
3. Compare the displayed codes with the *verification codes* shown on the voting cards (below the selected voting options).
4. If all codes match, scan the *confirmation QR code* on the back of the confirmation card.
5. Compare the displayed code with the *participation code* shown on the confirmation card.

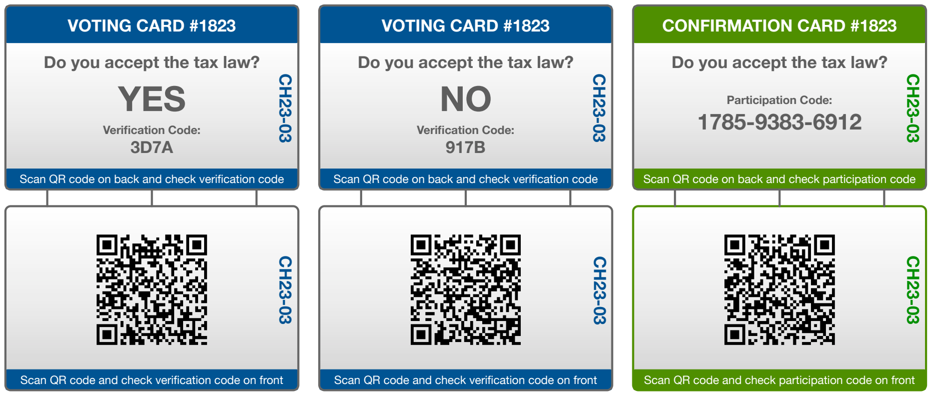

VOTING CARD #1823
Do you accept the tax law?
YES
Verification Code:
3D7A
CH23-03
Scan QR code on back and check verification code
Scan QR code and check verification code on front
VOTING CARD #1823
Do you accept the tax law?
NO
Verification Code:
917B
CH23-03
Scan QR code on back and check verification code
Scan QR code and check verification code on front
CONFIRMATION CARD #1823
Do you accept the tax law?
Participation Code:
1785-9383-6912
CH23-03
Scan QR code on back and check participation code
Scan QR code and check participation code on front

Fig. 1: Example of voting and participation cards for a referendum with two voting options `"YES"` and `"NO"`. The front sides of the cards are shown on top and the back sides at the bottom. The sizes of the QR codes are realistic.

For the verification codes to match in Step 3, the votes must have been cast as intended with high probability, and for the participation code to match in Step 5, the vote must have been confirmed as intended with high probability. This mechanism for achieving cast-as-intended and recorded-as-cast verifiability is exactly the same as in current approaches used in Switzerland [4,8], but in our new protocol, the voting client learns nothing about the selected voting options. Note that the general idea and the voting procedure in our approach are very similar to Chaum's *code voting* [3,7], but our new protocol requires much weaker trust assumptions for generating the codes and performing the tally.

In case the verification codes do not match in Step 3 of the above procedure, voters in Switzerland are instructed to submit the ballot on paper using the existing traditional voting channels. We adopt this convention here to handle such exceptional cases properly, i.e., without creating new security problems. In case the participation code does not match in Step 5, vote confirmation can be repeated—possibly on different devices—until the code matches. If the problem persists, voters are instructed to report the problem to the electoral office.

In the referendum example of Figure 1, the voter simply selects one of the two voting cards with ID $i = 1823$, let's say the second one for voting option `"NO"`, scans the QR code on the back of the card using a mobile device, checks if the verification code displayed on the device is `"917B"`, scans the QR code on the back of the confirmation card, and finally checks if the participation code displayed on the device is `"1785-9383-6912"`. Note that this procedure can be completed very quickly, for a single referendum possibly within a few seconds.

1.2 Contribution and Overview

This paper presents a new cryptographic voting protocol, which primarily aims at achieving vote privacy on untrustworthy voting clients. It has similarities to existing approaches [5,10], but it is quite unique in its elegant way of combining recent but well-established cryptographic primitives (BLS signatures, verifiable mix-nets, zero-knowledge proofs) with modern but wide-spread and accepted technologies (scanning of QR codes on mobile phones). Given this unique combination, we can achieve an additional important security goal simultaneously with improving the usability for the voters. Because security and usability are usually in conflict with each other, this is quite remarkable.

By limiting the responsibility of the voting client in the protocol, we also reduce the set of attack vectors and the overall complexity of the whole system. In an actual implementation, for example, since there is no need for implementing cryptography in JavaScript, developers can optimize their efforts into a single reliable back-end technology. At the same time, attacks based on injecting malicious JavaScript code become less likely and less powerful.

In Section 2, we start with an overview of the cryptographic primitives used in our protocol. The protocol itself is described in Section 3, which is the main section of this paper. The protocol description defines the involved protocol parties, the communication channels, the adversary model, and the three main protocol phases. In Section 4, we discuss the security properties of the proposed protocol, and Section 5 concludes the paper.

2 Cryptographic Background

Our protocol is based on various cryptographic primitives, which are commonly used in e-voting protocols. We briefly introduce them in the following subsections, but we refer to the literature for more details. Our protocol also relies on BLS signatures, which have not been used in e-voting very often, but which have become an established tool in other areas such as crypto-currencies. We briefly introduce BLS signatures in Section 2.2.

2.1 ElGamal Encryptions

The protocol as presented in this paper depends on a homomorphic asymmetric encryption scheme such as ElGamal. The choice of ElGamal is not mandatory, but we propose using it for its simplicity and maximal convenience. ElGamal is IND-CPA secure in a group, in which the DDH problem is hard. We propose the usual prime-order subgroup $\mathbb{G}_q \subset \mathbb{Z}_p^*$ of quadratic residues modulo a safe prime $p = 2q + 1$ and assume that $\mathbb{G}_q$ and a generator $g \in \mathbb{G}_q \backslash \{1\}$ are publicly known.

In this setting, we denote the generation of an ElGamal key pair consisting of a randomly selected private decryption key $dk \in \mathbb{Z}_q$ and a public encryption key $ek = g^{dk}$ by $(dk, ek) \xleftarrow{\$} \mathsf{KeyGenEnc}()$. For a given encryption key $ek \in \mathbb{G}_q$ and message $m \in \mathbb{G}_q$, we use $e \leftarrow \mathsf{Enc}_{ek}(m, r)$ to denote the encryption of m with randomization $r \xleftarrow{\$} \mathbb{Z}_q$. The result is a ciphertext pair $e = (g^r, m \cdot ek^r) \in \mathbb{G}_q \times \mathbb{G}_q$ consisting of two group elements. For the decryption $m \leftarrow \mathsf{Dec}_{dk}(e)$ of a ciphertext $e = (a, b)$, the decryption key dk can be used to compute $m = b/a^{dk}$.

ElGamal keys can be generated in a distributed setting, in which the private decryption key of a common encryption key is shared among multiple parties. In the simplest construction of $ek = \prod_{k=1}^{s} ek_k$, all s holders of a private key share dk_k need to cooperate to perform the decryption. We use $a'_k \leftarrow \mathsf{pDec}_{dk_k}(e)$ to denote the partial decryption of a ciphertext $e = (a, b)$. The resulting values $a'_k = a^{dk_k}$ can then be used to retrieve the plaintext $m = b / \prod_{k=1}^{s} a'_k$. More involved constructions exist to enable a threshold number $t \leq s$ of key share holders to perform the decryption. This is an optional extension for our protocol.

In applications of ElGamal in e-voting, where up to m voting options can be selected from n available voting options, it is possible to encode a set of selections $S \subset [1, n]$, $|S| \leq m$, into an ElGamal message by selecting n small prime numbers $p_1, \ldots, p_n$ from $\mathbb{G}_q \cap \mathbb{P}$ and by computing the product $\Gamma(S) = \prod_{s \in S} p_s$ over all selections. Under the condition that $\Gamma(S) < p$, such a product can be decoded into S using integer factorization. Therefore, to guarantee the uniqueness of the decoding, the maximal size of the parameters m and n is limited by p. In most practical election use cases, however, this limitation is not a real problem.

2.2 BLS Signatures

BLS signatures are defined over three groups G_1, G_2, and G_T of a bilinear map $e : G_1 \times G_2 \rightarrow G_T$. If we use the popular pairing-friendly elliptic curves from the Barreto-Lynn-Scott family, for example, we achieve approximately 128 bits of security for BLS12-381 and 256 bits of security for BLS48-581 (both curves are included in a current IETF draft [11]). In the particular case of BLS12-381, $G_1 \subset E_1[\mathbb{F}_p]$ is a prime-order subgroup of the elliptic curve $E_1 : y^2 = x^3 + 4$ over the prime-order field $\mathbb{F}_p$ for $\|p\| = 381$, $G_2 \subset E_2[\mathbb{F}_{p^2}]$ is a subgroup of the curve $E_2 : y^2 = x^3 + 4(1 + i)$ over $\mathbb{F}_{p^2}$, and $G_T \subset \mathbb{F}_{p^{12}}$ is a subgroup of the integers modulo p^{12}. The common group order $q = |G_1| = |G_2| = |G_T|$ is 256 bits long, which corresponds to 128 bits of security. Note that BLS12-381 also defines generators $g_1 \in G_1$ and $g_2 \in G_2$ for both groups.

In the BLS signature scheme, we use $(sk, vk) \overset{\$}{\leftarrow} \mathsf{KeyGenSig}()$ to denote the generation of a key pair. The private signature key $sk \in \mathbb{Z}_q$ is picked at random and the public verification key $vk = g_2^{sk}$ is computed in the group G_2.[1] For a collision-resistant hash function $\mathsf{hash} : \{0,1\}^* \rightarrow G_1$, the BLS signature of a message $m \in \{0,1\}^*$ is a single deterministic group element $\sigma = \mathsf{hash}(m)^{sk} \in G_1$. We use $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$ to denote this operation. Note that such signatures are points $\sigma = (x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ on the curve $E_1[\mathbb{F}_p]$, which can be represented using $\|p\| + 1$ bits ($\|p\|$ bits for x and 1 bit for the sign of $\pm y$). For BLS12-381, the signature size is therefore 382 bits (48 bytes). A BLS signature $\sigma \in G_1$ is valid if $e(\sigma, g_2) = e(\mathsf{hash}(m), vk)$, i.e., for performing the verification $\mathsf{Verify}_{vk}(\sigma, m) \in \{0,1\}$, the pairing function needs to be computed twice.

A useful property of the BLS signature scheme is the aggregation of signatures $\sigma_k \leftarrow \mathsf{Sign}_{sk_k}(m)$ generated under multiple private keys into a single signature $\sigma = \prod_{k=1}^{s} \sigma_k$. The aggregated signature can then be verified using the combined verification key $vk = \prod_{k=1}^{s} vk_k$ of all s key holders. As in the case of ElGamal,

[1] In our protocol, we minimize the size of the signatures by using G_1 for the signatures and G_2 for the public keys, but the roles of the groups are interchangeable.

there are more involved methods to allow the signature to be generated by a threshold number of key holders, but we will not need this in our protocol.

2.3 Non-Interactive Zero-Knowledge Proofs

We use different non-interactive zero-knowledge proofs to ensure that the involved election authorities follow the protocol faithfully. First, to avoid the possibility of so-called *rogue key attacks* [9], they need to prove knowledge of the generated private ElGamal and BLS keys dk and sk, respectively. We will denote the generation of such a proof as

$$\pi^{\mathsf{key}} \xleftarrow{\$} \mathsf{NIZKP}[(dk, sk) : ek = g^{dk} \wedge vk = g_2^{sk}],$$

and the verification as $\mathsf{Verify}(\pi^{\mathsf{key}}, ek, vk) \in \{0,1\}$. Note that π^{key} is a conjunctive composition of two non-interactive Schnorr proofs. Second, authorities need to prove the correctness of the partial decryptions $a'_i = a_i^{dk}$ for a given list $\mathbf{e} = (e_1, \ldots, e_N)$ of ElGamal ciphertexts $e_i = (a_i, b_i)$. We denote this proof as

$$\pi^{\mathsf{dec}} \xleftarrow{\$} \mathsf{NIZKP}[(dk) : ek = g^{dk} \wedge \left(\bigwedge_{i=1}^{N} a'_i = a_i^{dk}\right)],$$

which is a conjunctive composition of $N+1$ non-interactive Schnorr proofs. For $\mathbf{a} = (a_1, \ldots, a_N)$ and $\mathbf{a}' = (a'_1, \ldots, a'_N)$, the proof verification is denoted by $\mathsf{Verify}(\pi^{\mathsf{dec}}, ek, \mathbf{a}, \mathbf{a}') \in \{0,1\}$. For the details of corresponding proof generation and verification algorithms, we refer to the literature [4, Section 5.4].

2.4 Verifiable Mix-Nets

Other important cryptographic tools in our protocol are verifiable re-encryption mix-nets. The idea is to apply a series of cryptographic shuffles to an input list $\mathbf{e} = (e_1, \ldots, e_N)$ of ElGamal ciphertexts. In each shuffle, a random permutation is applied to the list, and every ciphertext of the permuted list is re-encrypted using a fresh randomization. If Ψ_N denotes the set of all permutations of size N, then $\psi \xleftarrow{\$} \Psi$ denotes picking a permutation uniformly at random, and $j = \psi(i)$ denotes the application of $\psi : [1,N] \rightarrow [1,N]$ to an input index i. The re-encryption $(\tilde{a}, \tilde{b}) \leftarrow \mathsf{ReEnc}_{ek}(e, r)$ of an ElGamal ciphertext $e = (a,b)$ is a new ciphertext $(\tilde{a}, \tilde{b}) = (a,b) \cdot \mathsf{Enc}_{ek}(1, r) = (a \cdot g^r, b \cdot ek^r)$ for a fresh randomization $r \xleftarrow{\$} \mathbb{Z}_q$.

By putting everything together, cryptographic shuffling can be defined by $\tilde{\mathbf{e}} \leftarrow \mathsf{Shuffle}_{ek}(\mathbf{e}, \psi, \mathbf{r})$, where $\tilde{\mathbf{e}} = (\tilde{e}_1, \ldots, \tilde{e}_N)$ denotes the shuffled list of re-encrypted ciphertexts $\tilde{e}_i = \mathsf{ReEnc}_{ek}(e_j, r_j)$ for $j = \psi(i)$. To prove the correctness of the shuffle, authorities need to provide a non-interactive *shuffle proof*,

$$\pi^{\mathsf{shuffle}} \xleftarrow{\$} \mathsf{NIZKP}[(\psi, \mathbf{r}) : \tilde{\mathbf{e}} = \mathsf{Shuffle}_{ek}(\mathbf{e}, \psi, \mathbf{r})],$$

which can be verified by $\mathsf{Verify}(\pi^{\mathsf{shuffle}}, ek, \mathbf{e}, \tilde{\mathbf{e}}) \in \{0,1\}$. We require such shuffle proofs twice in our protocol. In one of the two cases, we will need to ensure that the permutation used in the shuffle corresponds to an existing *permutation commitment* $c \leftarrow \mathsf{Commit}(\psi, r)$. In existing constructions [2, 13], such commitments are part of the proof generation and included in π^{shuffle}. Therefore, by assuming that the same algorithm generates the permutation commitment and the shuffle proof simultaneously, we can use a slightly abusive notation,

$$(\pi^{\mathsf{shuffle}}, c, r) \xleftarrow{\$} \mathsf{NIZKP}[(\psi, \mathbf{r}) : \tilde{\mathbf{e}} = \mathsf{Shuffle}_{ek}(\mathbf{e}, \psi, \mathbf{r})],$$

for the proof generation and include c as an additional argument for the proof verification. For further details on theses proofs, we refer again to the literature or to the pseudocode algorithms given in [4, Sections 5.5 and 8.4].

3 Protocol Description

Our new Internet voting protocol consists of three consecutive phases called *pre-election*, *election*, and *post-election*. To specify the technical details of the protocol, we provide separate subsections for each of them. As a preparatory step at the beginning of this section, we define the set of election parameters, the protocol parties and communication channels, and the protocol's general idea.

3.1 Election Parameters

In the discussion of the election model in Section 1.1, we already decided to restrict our approach to single m-out-of-n elections, where n denotes the number of *candidates* (or *voting options*) and $m < n$ the maximal number of candidates to choose. For each election, we assign a *unique election identifier* U and an *election description* ED. Furthermore, we assign a unique *candidate description* CD_j to each candidate, and for an electorate of size N, we assign a unique *voter description* VD_i to each voter i. If $\mathbf{c} = (CD_1, \ldots, CD_n)$ and $\mathbf{v} = (VD_1, \ldots, VD_N)$ denote the vectors of all candidate and voter descriptions, respectively, then

$$EP = (U, ED, m, n, N, \mathbf{c}, \mathbf{v})$$

denotes the complete set of election parameters (note that $n = |\mathbf{c}|$ and $N = |\mathbf{v}|$ are redundant, but we prefer to list them explicitly). Without loss of generality, we assume that U, ED, CD_j, and VD_i are strings from a given alphabet. We also assume that these strings contain sufficient information for their specific purposes (for example, VD_i may contain the voter's name and address to enable the production of address labels by the printing service). In Figure 1, for example, we have $U =$ `"CH23-03"`, $ED =$ `"Do you accept the tax law?"`, $m = 1$, $n = 2$, $CD_1 =$ "`YES`", $CD_2 =$ "`NO`", and $i = 1823$ (N and $\mathbf{v}$ are unspecified).

3.2 Protocol Parties and Communication

We distinguish five different types of protocol parties. In an implementation of the protocol, the *administrator*, the *election authorities*, and the *printing service* form the system's core infrastructure, which can be used in the same constellation for multiple elections. The *voters* are members of the electorate of a given election and the *voting clients* are the personal devices used by the voters for casting the vote. The following list describes the roles of the five party types.

Administrator. To run an election, the administrator specifies the election parameters EP, launches the three protocol phases, and finally assembles and announces the election result. The administrator does not generate or possess any cryptographic secrets other than a private signature key.

Election Authorities. In the pre-election phase, the election authorities generate a common encryption key, a common signature key, and the contents of the voting and confirmation cards in a distributed manner. During the election, they respond to submitted ballots and confirmations, and in the post-election phase, they shuffle and decrypt the list of encrypted votes. To ensure the correctness of the election and the privacy of the votes, it is assumed that at least one election authority is honest. They all possess a private signature key.

Printing Service. The printing service is responsible for printing the voting and confirmation cards and sending them to the voters. For this, it receives shares of the values to be printed from the election authorities. The printing service is trusted to assemble these shares in a deterministic manner, print them on paper, and protect the confidentiality of the received data. Otherwise, the printing service does not generate or possess any cryptographic secrets.

Voters. The voters are the main actors in the protocol. Using the voting and confirmation cards obtained from the printing service over postal mail, honest voters submit their votes according to the procedure described in Section 1.1. Dishonest voters may deviate from the protocol, but not if this affects the validity of the submitted vote. This implies that they keep the voting and confirmation cards secret, even in the presence of a vote buyer or coercer.

Voting Clients. The functionality of the voting clients is restricted to scanning and submitting the QR codes to the election authorities and displaying their responses to the voters. Since they are potentially dishonest, they do not generate or possess any cryptographic secrets.

To authenticate the communication channels between the infrastructure parties, we assume that the administrator and the election authorities use their private signature keys to sign all outgoing messages, and that all infrastructure parties accept incoming messages only if they contain a valid signature. Furthermore, we assume that the channels from the election authorities to the printing service and from the printing service to the voters are confidential.

Note that we do not explicitly impose the existence a public bulletin board. As we will see, every election authority in our protocol keeps a full record of all the public election data. By considering the union of their records as input for the verification, it is sufficient to have at least one honest authority for reaching a consensus about the relevant election data. This approach is consistent with the current practice and legislation in Switzerland.

3.3 General Protocol Idea

From a cryptographic point of view, the main protocol idea is to prepare for each voter a list of ciphertexts for all n possible voting options. For this, the election authorities apply a verifiable mix-net to an initial list of trivial ElGamal ciphertexts $e_j = \mathsf{Enc}_{ek}(\mathit{\Gamma}(j), 0) = (1, \mathit{\Gamma}(j))$ for all candidates $j \in [1, n]$. This leads to a matrix $\tilde{\mathbf{E}} = (\tilde{e}_{ij})_{N \times n}$ of encrypted votes, in which each row i contains encryptions of all candidates in permuted order ψ_i. The correctness of this matrix can be publicly verified by checking the shuffle proofs from the mix-net.

To submit a vote for candidate $s \in [1, n]$, the voter with voting card index i needs to know the right column index $j = \psi_i(s)$ in row i of the matrix $\tilde{\mathbf{E}}$ and submit it to the election authorities. In other words, by submitting a ballot $b = (U, i, j)$ to the authorities, the voter expresses the intention to vote in election U for candidate $s = \psi_i^{-1}(j)$. The authorities can then pick the encrypted vote $\tilde{e}_{ij}$ from $\tilde{\mathbf{E}}$ and add it to the ballot box. While this is the basic idea of the protocol, it is clear that additional security measures need to be taken.

To prevent ballot stuffing, the election authorities generate BLS signatures $\sigma_{ij} = \mathsf{Sign}_{sk}(U, i, j)$ in a distributed manner for all pairs $(i, j) \in [1, N] \times [1, n]$. To cast a valid vote, voters must then submit an extended ballot $b = (U, i, j, \sigma_{ij})$, which includes a valid signature σ_{ij} for (U, i, j). This is exactly the information that needs to be included in the QR code assigned to the voting option s. If we assume that the two voting options have been swapped in the mix-net of the example given in Figure 1, then the QR code assigned to the second voting option NO $(s = 2)$ could be an encoding of the URL string

```
https://www.qrvote.ch?U=CH23-03&i=1823&j=1&sigma=7BM8qdOu08gwVKpjmZ9
        hf1u0+KSotcq4DWeFgcXgn2vloWJbYLHUcl+wpUaZJgoR,
```

which directs the browser to the specified web server and submits the ballot $b = (\texttt{"CH22-12"}, 1823, 1, \sigma)$ included in the query string to the election authorities. In this particular example, σ is a Base64-encoded BLS signature of length $64 * 6 = 384$ bits (48 bytes), which corresponds to an element of G_1 in BLS12-381. If σ is a valid signature, the authorities respond with a share of the corresponding verification code. The aggregation of these shares is displayed to the voter.

In an election with $m > 1$ possible selections, the authorities accept such incoming ballots until the size of the voter's ballot box reaches m. To confirm the vote, the voter submits a similar tuple $c = (U, i, \sigma_i)$, where $\sigma_i = \mathsf{Sign}_{sk}(U, i)$ is an aggregated BLS signature of (U, i), which represents the voter's intention to confirm the vote in election U. Again, if c contains a valid signature, the authorities respond with a share of the corresponding participation code, and an aggregation of these shares is displayed to the voter.

3.4 Pre-Election Phase

The pre-election phase begins with an initial message EP from the administrator to the election authorities. Upon receiving this message, the election authorities generate a common ElGamal encryption key ek and a common BLS signature key vk by exchanging corresponding key shares. In Figure 2, the key shares are generated in Step2a and the key aggregation takes place in Step2b.

In the next step of the pre-election phase, the election authorities generate the ciphertext matrix $\tilde{\mathbf{E}}$, in which each row contains ciphertexts for all candidates in permuted order. In Figure 2, the initialization of this list is shown in Step3a,

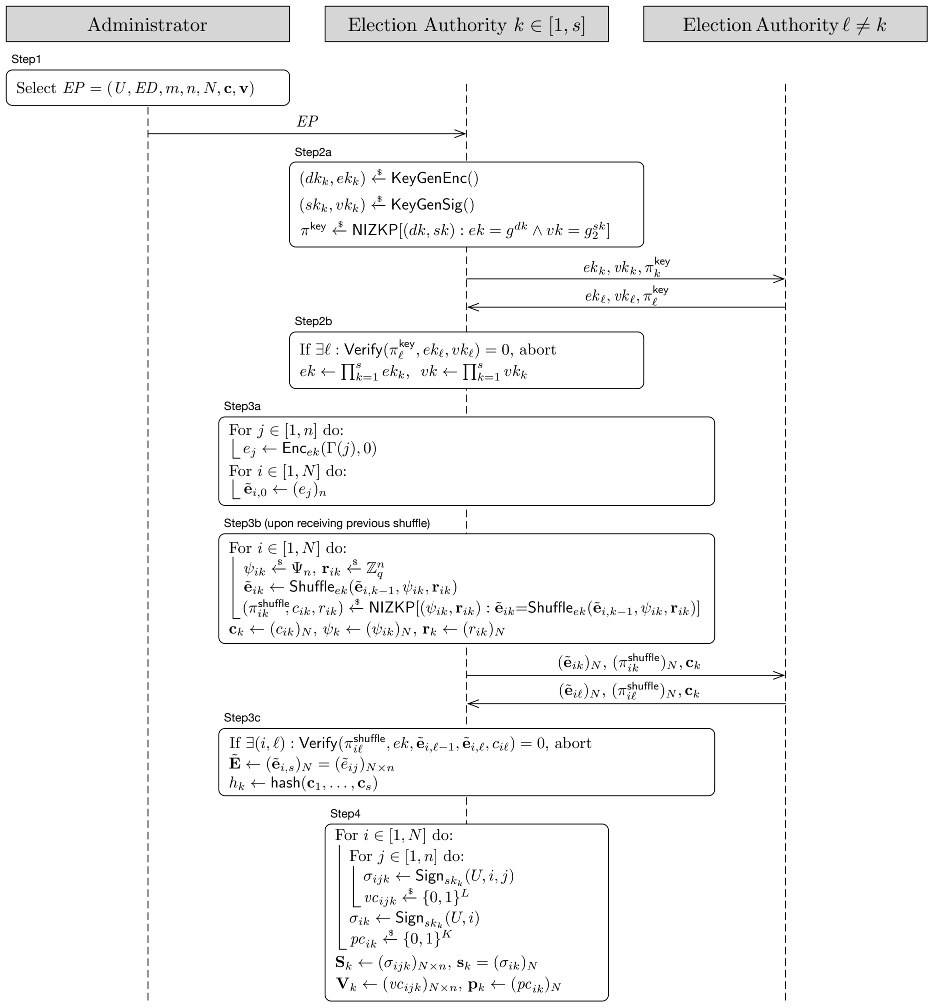
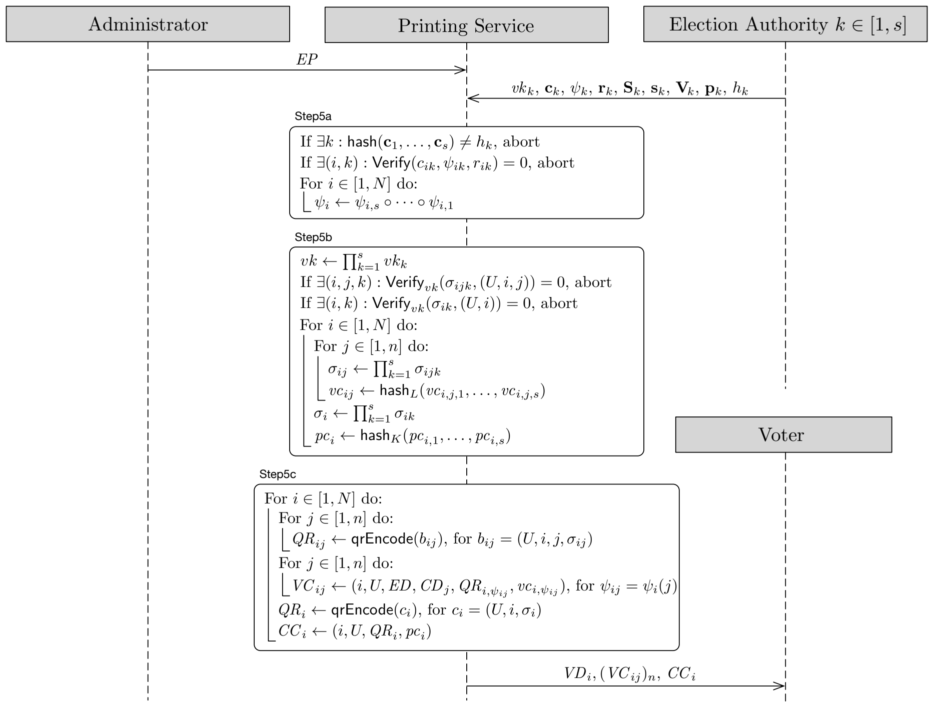

Administrator
Election Authority $k \in [1,s]$
Election Authority $\ell \neq k$
Step1
Select $EP = (U, ED, m, n, N, \mathbf{c}, \mathbf{v})$
EP
Step2a
$(dk_k, ek_k) \xleftarrow{\$} \mathsf{KeyGenEnc}()$
$(sk_k, vk_k) \xleftarrow{\$} \mathsf{KeyGenSig}()$
$\pi^{\mathsf{key}} \xleftarrow{\$} \mathsf{NIZKP}[(dk, sk) : ek = g^{dk} \wedge vk = g_2^{sk}]$
$ek_k, vk_k, \pi_k^{\mathsf{key}}$
$ek_\ell, vk_\ell, \pi_\ell^{\mathsf{key}}$
Step2b
If $\exists \ell : \mathsf{Verify}(\pi_\ell^{\mathsf{key}}, ek_\ell, vk_\ell) = 0$, abort
$ek \leftarrow \prod_{k=1}^{s} ek_k,\ vk \leftarrow \prod_{k=1}^{s} vk_k$
Step3a
For $j \in [1,n]$ do:
$e_j \leftarrow \mathsf{Enc}_{ek}(\Gamma(j), 0)$
For $i \in [1,N]$ do:
$\tilde{\mathbf{e}}_{i,0} \leftarrow (e_j)_n$
Step3b (upon receiving previous shuffle)
For $i \in [1,N]$ do:
$\psi_{ik} \xleftarrow{\$} \Psi_n$, $\mathbf{r}_{ik} \xleftarrow{\$} \mathbb{Z}_q^n$
$\tilde{\mathbf{e}}_{ik} \leftarrow \mathsf{Shuffle}_{ek}(\tilde{\mathbf{e}}_{i,k-1}, \psi_{ik}, \mathbf{r}_{ik})$
$(\pi_{ik}^{\mathsf{shuffle}}, c_{ik}, r_{ik}) \xleftarrow{\$} \mathsf{NIZKP}[(\psi_{ik}, \mathbf{r}_{ik}) : \tilde{\mathbf{e}}_{ik} = \mathsf{Shuffle}_{ek}(\tilde{\mathbf{e}}_{i,k-1}, \psi_{ik}, \mathbf{r}_{ik})]$
$\mathbf{c}_k \leftarrow (c_{ik})_N$, $\psi_k \leftarrow (\psi_{ik})_N$, $\mathbf{r}_k \leftarrow (r_{ik})_N$
$(\tilde{\mathbf{e}}_{ik})_N$, $(\pi_{ik}^{\mathsf{shuffle}})_N, \mathbf{c}_k$
$(\tilde{\mathbf{e}}_{i\ell})_N$, $(\pi_{i\ell}^{\mathsf{shuffle}})_N, \mathbf{c}_k$
Step3c
If $\exists (i,\ell) : \mathsf{Verify}(\pi_{i\ell}^{\mathsf{shuffle}}, ek, \tilde{\mathbf{e}}_{i,\ell-1}, \tilde{\mathbf{e}}_{i,\ell}, c_{i\ell}) = 0$, abort
$\tilde{\mathbf{E}} \leftarrow (\tilde{\mathbf{e}}_{i,s})_N = (\tilde{e}_{ij})_{N \times n}$
$h_k \leftarrow \mathsf{hash}(\mathbf{c}_1, \ldots, \mathbf{c}_s)$
Step4
For $i \in [1,N]$ do:
For $j \in [1,n]$ do:
$\sigma_{ijk} \leftarrow \mathsf{Sign}_{sk_k}(U, i, j)$
$vc_{ijk} \xleftarrow{\$} \{0,1\}^L$
$\sigma_{ik} \leftarrow \mathsf{Sign}_{sk_k}(U, i)$
$pc_{ik} \xleftarrow{\$} \{0,1\}^K$
$\mathbf{S}_k \leftarrow (\sigma_{ijk})_{N \times n}$, $\mathbf{s}_k = (\sigma_{ik})_N$
$\mathbf{V}_k \leftarrow (vc_{ijk})_{N \times n}$, $\mathbf{p}_k \leftarrow (pc_{ik})_N$

Fig. 2: First part of the pre-election phase: Election setup (Step1), key generation (Step2), ciphertext preparation (Step3), signature and code generation (Step4).

the shuffling in Step3b, and the assembling of the matrix $\tilde{\mathbf{E}}$ in Step3c. As a side-product of this procedure, each election authorities receives a list of permutation commitments $\mathbf{c}_\ell$ from all other authorities $\ell \neq k$. The hash $h_k = (\mathbf{c}_1, \ldots, \mathbf{c}_s)$ of these commitments is used to demonstrate to the printing service that a consensus have been reached among the election authorities about the outcome of the mixing. Another side-product are the commitment randomizations $\mathbf{r}_k$, which are given to the printing service for opening the commitments.

In Step4 of the pre-election phase, the election authorities prepare the shares σ_{ijk} of the BLS signatures for the ballots $b_{ij} = (U, i, j, \sigma_{ij})$ and the shares vc_{ijk} of corresponding verification codes vc_{ij}. Similarly, they prepare the shares σ_{ik} of the BLS signatures for the confirmations $c_i = (U, i, \sigma_i)$ and the shares pc_{ik}


Administrator
Printing Service
Election Authority $k \in [1,s]$
EP
$vk_k,\ \mathbf{c}_k,\ \psi_k,\ \mathbf{r}_k,\ \mathbf{S}_k,\ \mathbf{s}_k,\ \mathbf{V}_k,\ \mathbf{p}_k,\ h_k$
Voter
$VD_i, (VC_{ij})_n,\ CC_i$

```
Step5a
If $\exists k : \mathsf{hash}(\mathbf{c}_1,\dots,\mathbf{c}_s) \neq h_k$, abort
If $\exists (i,k) : \mathsf{Verify}(c_{ik},\psi_{ik},r_{ik}) = 0$, abort
For $i \in [1,N]$ do:
    $\psi_i \leftarrow \psi_{i,s} \circ \cdots \circ \psi_{i,1}$

Step5b
$vk \leftarrow \prod_{k=1}^{s} vk_k$
If $\exists (i,j,k) : \mathsf{Verify}_{vk}(\sigma_{ijk},(U,i,j)) = 0$, abort
If $\exists (i,k) : \mathsf{Verify}_{vk}(\sigma_{ik},(U,i)) = 0$, abort
For $i \in [1,N]$ do:
    For $j \in [1,n]$ do:
        $\sigma_{ij} \leftarrow \prod_{k=1}^{s} \sigma_{ijk}$
        $vc_{ij} \leftarrow \mathsf{hash}_L(vc_{i,j,1},\dots,vc_{i,j,s})$
    $\sigma_i \leftarrow \prod_{k=1}^{s} \sigma_{ik}$
    $pc_i \leftarrow \mathsf{hash}_K(pc_{i,1},\dots,pc_{i,s})$

Step5c
For $i \in [1,N]$ do:
    For $j \in [1,n]$ do:
        $QR_{ij} \leftarrow \mathsf{qrEncode}(b_{ij})$, for $b_{ij} = (U,i,j,\sigma_{ij})$
    For $j \in [1,n]$ do:
        $VC_{ij} \leftarrow (i,U,ED,CD_j,QR_{i,\psi_{ij}},vc_{i,\psi_{ij}})$, for $\psi_{ij} = \psi_i(j)$
    $QR_i \leftarrow \mathsf{qrEncode}(c_i)$, for $c_i = (U,i,\sigma_i)$
    $CC_i \leftarrow (i,U,QR_i,pc_i)$
```

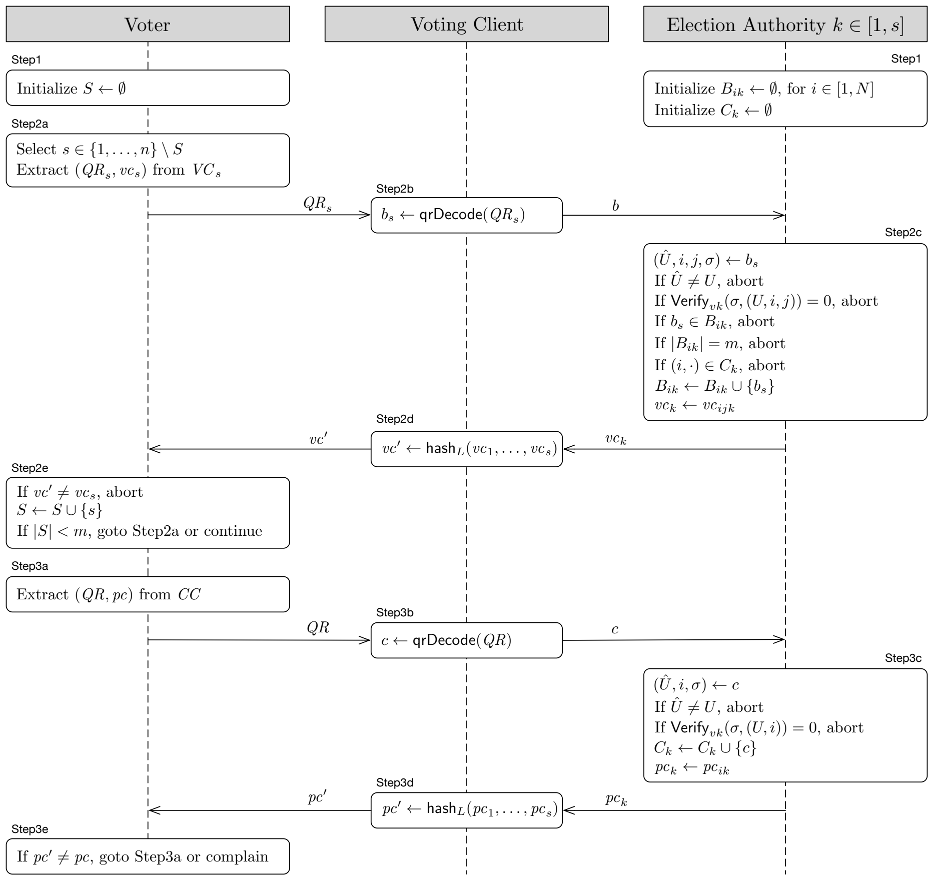
Fig. 3: Second part of the pre-election phase: Permutation composition (Step5a), signature and code aggregation (Step5b), voting card preparation (Step5c).

of corresponding participation codes pc_i. These codes are random bit strings of length L and K, respectively (in the example from Figure 1, we have $L = 16$ and $K = 40$). Finally, the share vk_k of the verification key, the commitments $\mathbf{c}_k$, the random permutation ψ_k, the randomizations $\mathbf{r}_k$, the shares $\mathbf{S}_k$ and $\mathbf{s}_k$ of the BLS signatures, the shares $\mathbf{V}_k$ and $\mathbf{p}_k$ of the verifications and confirmation codes, and the hash value h_k of the commitments are sent to the printer.

The above messages launch the printing process. In Step5a of Figure 3, the printing service first checks the consistency and validity of the permutation commitments and then combines the permutations into $\psi_i = \psi_{i,s} \circ \cdots \circ \psi_{i,1}$ for every $i \in [1,N]$. Similarly, in Step5b, the printing serving checks the validity of the received signatures and then computes their aggregations σ_{ij} and σ_i. The same happens for the verification codes vc_{ij} and participation codes pc_i. From these values, the printing service assembles the voting cards VC_{ij} and confirmation cards CC_i, such that everything from Figure 1 is included at the right place. The cards are printed and sent to the voters over a confidential channel.

3.5 Election Phase

The election phase is shown in Figure 4. It consists of three consecutive steps. In Step1, the election authorities initialize their ballot boxes B_{ik} (one for each voter)

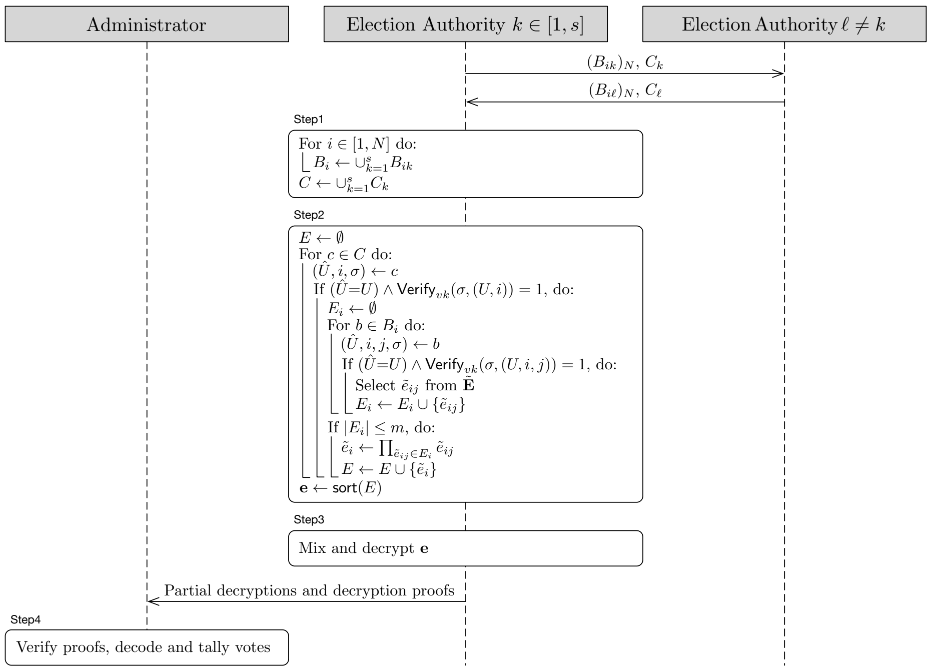


Voter
Voting Client
Election Authority $k \in [1, s]$
Step1
Initialize $S \leftarrow \emptyset$
Step1
Initialize $B_{ik} \leftarrow \emptyset$, for $i \in [1, N]$
Initialize $C_k \leftarrow \emptyset$
Step2a
Select $s \in \{1, \dots, n\} \setminus S$
Extract (QR_s, vc_s) from VC_s
QR_s
Step2b
$b_s \leftarrow \mathsf{qrDecode}(QR_s)$
b
Step2c
$(\hat{U}, i, j, \sigma) \leftarrow b_s$
If $\hat{U} \neq U$, abort
If $\mathsf{Verify}_{vk}(\sigma, (U, i, j)) = 0$, abort
If $b_s \in B_{ik}$, abort
If $|B_{ik}| = m$, abort
If $(i, \cdot) \in C_k$, abort
$B_{ik} \leftarrow B_{ik} \cup \{b_s\}$
$vc_k \leftarrow vc_{ijk}$
Step2d
vc'
$vc' \leftarrow \mathsf{hash}_L(vc_1, \dots, vc_s)$
vc_k
Step2e
If $vc' \neq vc_s$, abort
$S \leftarrow S \cup \{s\}$
If $|S| < m$, goto Step2a or continue
Step3a
Extract (QR, pc) from CC
Step3b
QR
$c \leftarrow \mathsf{qrDecode}(QR)$
c
Step3c
$(\hat{U}, i, \sigma) \leftarrow c$
If $\hat{U} \neq U$, abort
If $\mathsf{Verify}_{vk}(\sigma, (U, i)) = 0$, abort
$C_k \leftarrow C_k \cup \{c\}$
$pc_k \leftarrow pc_{ik}$
Step3d
pc'
$pc' \leftarrow \mathsf{hash}_L(pc_1, \dots, pc_s)$
pc_k
Step3e
If $pc' \neq pc$, goto Step3a or complain

Fig. 4: Overview of the election phase: Initialization (Step1), vote casting (Step2), vote confirmation (Step3).

and their confirmation box C_k. A voter, who has selected voting option $s \in [1, n]$ and is ready to vote, uses the personal device to scan QR_s on the voting card, decode QR_s into b_s, and send the ballot to the election authorities (Step2a–2b). If the ballot is valid and fresh, and if the size of the ballot box is less than m and the vote has not been confirmed yet, it is accepted into the ballot box and the share of the verification code is returned (Step2c). The voting client assembles the shares and displays the code to the voter (Step2d). The voter checks the verification code and then decides to submit further ballots or confirm (Step2e).

The procedure for the vote confirmation is almost identical, i.e., the voter uses the personal device to scan QR, decode it into c, and send the confirmation to the election authorities (Step3a–3b). They check its validity and return their share of the participation code (Step3c). The voting client assembles the shares and displays the code to the voter (Step3d). Finally, the voter terminates the vote casting procedure by checking the participation code (Step3e). Note that the confirmation can be repeated multiple times, possibly on different devices.


Administrator
Election Authority $k \in [1, s]$
Election Authority $\ell \neq k$
$(B_{ik})_N, C_k$
$(B_{i\ell})_N, C_\ell$
Step1
Step2
Step3
Mix and decrypt $\mathbf{e}$
Partial decryptions and decryption proofs
Step4
Verify proofs, decode and tally votes

```
Step1
For $i \in [1, N]$ do:
    $B_i \leftarrow \cup_{k=1}^{s} B_{ik}$
$C \leftarrow \cup_{k=1}^{s} C_k$

Step2
$E \leftarrow \emptyset$
For $c \in C$ do:
    $(\hat{U}, i, \sigma) \leftarrow c$
    If $(\hat{U}{=}U) \wedge \mathsf{Verify}_{vk}(\sigma, (U, i)) = 1$, do:
        $E_i \leftarrow \emptyset$
        For $b \in B_i$ do:
            $(\hat{U}, i, j, \sigma) \leftarrow b$
            If $(\hat{U}{=}U) \wedge \mathsf{Verify}_{vk}(\sigma, (U, i, j)) = 1$, do:
                Select $\tilde{e}_{ij}$ from $\tilde{\mathbf{E}}$
                $E_i \leftarrow E_i \cup \{\tilde{e}_{ij}\}$
        If $|E_i| \leq m$, do:
            $\tilde{e}_i \leftarrow \prod_{\tilde{e}_{ij} \in E_i} \tilde{e}_{ij}$
            $E \leftarrow E \cup \{\tilde{e}_i\}$
$\mathbf{e} \leftarrow \mathsf{sort}(E)$
```

Fig. 5: Overview of the post-election phase: ballot and confirmation box synchronization (Step1), extraction of valid ciphertexts (Step2), mixing and decryption (Step3), decoding and tallying (Step4).

3.6 Post-Election Phase

At the end of the voting period, the election authorities first need to synchronize their ballot and confirmation boxes. This step might not be necessary in a normal protocol run, but there is no general guarantee that all submitted ballots and confirmations have reached all authorities. In Figure 5, the synchronization takes place in Step1. To ensure that the resulting sets B_i and C only contain valid and confirmed votes, it is necessary to verify the included BLS signatures and to check that the total number of ballots does not exceed m. If this is the case for the ballots of a given voter, we use the homomorphic property of ElGamal to merge the included ciphertexts. Therefore, the resulting set E of encrypted votes contains at most one entry for each voter. To obtain an unambiguous mix-net input from Step2 of Figure 5, we consider the list $\mathbf{e}$ obtained from sorting E.

For the two remaining steps of the post-election phase, space constraints in Figure 5 do not allow us to give further details. In Step3, the list $\mathbf{e}$ of encrypted votes is mixed in a procedure similar to Step3 from Figure 2, and the resulting shuffled list $\tilde{\mathbf{e}}$ is decrypted in a distributed manner as explained in Section 2.1. In Step4, the administrator assembles the plaintext votes from the partial decryptions and obtains the election result from applying Γ^{-1} to the plaintext votes. The correctness of the obtained election result is guaranteed by corresponding non-interactive zero-knowledge proofs π_k^{dec} and π_k^{shuffle}.

4 Security Discussion

We consider the usual active polynomial-time adversary, who might want to break vote privacy or manipulate the election result. The adversary may therefore try to corrupt as many protocol parties as needed, but we assume that the printing service and at least one election authority remain honest under all circumstances, i.e., they follow the protocol faithfully and do no disclose any of their secrets to another party. In this model, our protocol tries to achieve *covert security* [1,12], which means that corrupt parties are only cheating as long as the cheating remains undetectable and therefore has no consequences. Within this model, we also assume that voters keep their voting and confirmation cards secret from vote buyers and coercers, possibly because they fear legal consequences (see Section 3.2).

Below, we will briefly discuss the security properties of our protocol in the covert adversary model by giving some informal arguments relative to vote privacy and vote integrity. Verifiability arguments are the same as in other protocols based on verification codes.

Vote Privacy: Let the adversary know the assignment of the voting card indices i to the voters. A first possibility for breaking vote privacy would be to learn the combined permutation ψ_i or randomization $\mathbf{r}_i$ from the pre-election mix-net, but both ψ_i and $\mathbf{r}_i$ are only known to the trusted printing service and to the trusted group of election authorities (of which at least one is assumed trustworthy). The second possibility would be to learn the permutation ψ or randomization $\mathbf{r}$ of the post-election mix-net, and the third possibility would be to learn the aggregated private decryption key dk. In both cases cases, the necessary secrets are only known to the trusted group of election authorities.

Vote Integrity: For a confirmed ballot b_{ij} to appear in the final tally, both b_{ij} and c_i it must contain a valid BLS signature. Thus, a first possibility for ballot stuffing would be to generate extra signatures, but this requires the aggregated private signing key sk, which is known only to the trusted group of election authorities (of which at least one is assumed trustworthy). The second possibility would be to use the unused ballots from abstaining voters, but these ballots are only known to the trusted printing authority and to the abstaining voters themselves, who are trusted for keeping corresponding voting cards secret.

For a submitted and confirmed ballot b_{ij} to drop out from the final tally, there are three possibilities: first, by preventing either b_{ij} or c_i from reaching the election authorities (but this would be detected by the voter performing the cast-as-intended and recorded-as-cast verification), second, by removing b_{ij} or c_i from the ballot boxes of all election authorities (which contradicts the assumption that at least one election authority is honest), and third, by adding additional ballots to the ballot box of at least one election authority, such that the total number of ballots exceeds m (this case can be excluded for the same reasons as ballot stuffing).

5 Conclusion

The new Internet voting protocol proposed in this paper can be seen as a modern version of Chaum's code voting scheme, in which usability concerns have been solved using the QR code scanning functionality of mobile devices and trust assumptions have been distributed to a group of election authorities, of which only one needs to be honest to prevent or detect privacy and integrity attacks. In this way, we achieve security in the covert adversary model, which is often sufficient. From the voter's perspective, the vote casting procedure in referendums or elections with a small number of candidates is intuitive, efficient, and secure. Whether this statement still holds for more complex elections is an open question. Another open issue is the current lack of a formal security proof.

References

1. Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. Journal of Cryptology **23**(2), 281–343 (2010)
2. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: EUROCRYPT'12, 31st Annual International Conference on Theory and Applications of Cryptographic Techniques. pp. 263–280. Cambridge, UK (2012)
3. Chaum, D.: Surevote: Technical overview. In: WOTE'01, 1st Workshop On Trustworthy Elections. Tomales Bay, USA (2001)
4. Haenni, R., Koenig, R.E., Locher, P., Dubuis, E.: CHVote protocol specification – version 3.4. IACR Cryptology ePrint Archive **2017/325** (2022)
5. Joaquim, R., Ribeiro, C., Ferreira, P.: VeryVote: A voter verifiable code voting system. In: VoteID'09, 2nd International Conference on E-Voting and Identity, pp. 106–121, Luxembourg (2009)
6. Kulyk, O., Ludwig, J., Volkamer, M., Koenig, R.E., Locher, P.: Usable verifiable secrecy-preserving e-voting. In: E-Vote-ID'21, 6th International Joint Conference on Electronic Voting. pp. 337–353. Bregenz, Austria (2021)
7. Oppliger, R.: How to address the secure platform problem for remote internet voting. In: SIS'02, 5th Conference on "Sicherheit in Informationssystemen". pp. 153–173. Vienna, Austria (2002)
8. Renold, H., Esseiva, O., Hofer, T.: Swiss Post Voting System – System Specification – Version 1.2.0. Tech. rep., Swiss Post Ltd., Bern, Switzerland (2022)
9. Ristenpart, T., Yilek, S.: The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In: EUROCRYPT'07, 26th International Conference on the Theory and Applications of Cryptographic Techniques. pp. 228–245. Barcelona, Spain (2007)
10. Ryan, P. Y. A., Teague, V.: Pretty good democracy. In: SPW'09, 17th International Workshop on Security Protocols. pp. 111–130. Cambridge, U.K. (2009)
11. Sakemi, Y., Kobayashi, T., Saito, T., Wahby, R.S.: Pairing-friendly curves. Internet-draft, Internet Engineering Task Force (IETF) (2022)
12. Scholl, P., Simkin, M., Siniscalchi, L.: Multiparty computation with covert security and public verifiability. IACR Cryptology ePrint Archive **2021/366** (2021)
13. Terelius, B., Wikström, D.: Proofs of restricted shuffles. In: AFRICACRYPT'10, 3rd International Conference on Cryptology in Africa. pp. 100–113. LNCS 6055, Stellenbosch, South Africa (2010)