Short Voting Codes For Practical Code Voting

Florian Moser[0000−0003−2268−2367]

INRIA Nancy, France florian.moser@inria.fr

Abstract. To preserve voter secrecy on untrusted voter devices we propose to use short voting codes. This ensures voting codes remain practical even if the voter is able to select multiple voting choices. We embed the mechanism in a protocol that avoids complex cryptography in both the setup and the voting phase and relies only on standard cryptographic primitives. Trusting the setup, and one out of multiple server components, the protocol provides vote secrecy, cast-as-intended, recorded-as-cast, tallied-as-recorded, eligibility and universal verifiability.

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

1 Introduction

In an internet voting system, the voter's client is usually considered untrusted. Consequentially, cast-as-intended and recorded-as-cast mechanism typically ensure the device does not alter the voter's choice. However, many of these mechanisms require the voter to enter their plain vote. Consequentially, the voter's device learns the plain vote of the voter.

A solution to safe-guard vote secrecy, even if the voter's client is malicious, are voting codes [2]. The voter no longer enters their plain vote, but a voting code, with the voter's device unable to attribute the voting code to the plain vote it represents. However, voting codes often tend to be long, as they incorporate ciphertext of the plain vote or authentication secrets. This makes entering the codes tedious for the voter, even more so if multiple voting choices are chosen.

In this work, we introduce *short* voting codes (1 or 2 digits in reasonable scenarios). Only as many different voting codes are needed as plain voting choices are available. In elections with a reasonable set of plain voting choices, the voting codes are therefore also of reasonable length. For each voter, the voting codes are assigned differently to the plain voting choices, so the voter's device does not learn the plain vote.

We implement this mechanism in a protocol providing vote secrecy, cast-as-intended, recorded-as-cast, tallied-as-recorded, eligibility verifiability and universal verifiability. We grant the adversary control of the network, the voter's device and some of the multiple server components we call control components. To guard our security properties, we assume the setup and one out of multiple server components trusted. Notably, our adversary and trust model, as well as the achieved properties, conform to the requirements set forward by the Swiss

chancellery for Swiss political national elections [4].[1] Against an adversary only observing the exchange between the voter and the control components, the system achieves everlasting privacy.

The protocol relies on well-know cryptographic constructs, and tasks which users are familiar with from other applications (entering and comparing short codes, scanning QR-codes). The setup and voting phases need only few operations not more complex than signatures and hashes, and are efficient. The tally phase additionally needs a privacy-preserving tally mechanism such as a homomorphic tally or a verifiable shuffle, both well-understood mechanisms.

Related Work Voting codes to achieve privacy on an untrusted voters device's were first proposed in 2001 [2]. Previous work directly uses the ciphertext as the voting code (as in BeleniosVS [3]) or an identifier of the corresponding ciphertext (as in Pretty Understandable Democracy (PUD) [1]). Further, some schemes use the voting codes to deliver additional guarantees. In Pretty Good Democracy, the voting codes are hard to guess to achieve receipt-freeness [10]. Many schemes also use the voting codes for authentication (i.e. a proposed code voting extension to the Swiss Post System [12], an other code voting scheme proposed in the Swiss setting [5] and others [2,7,9,8,5]).

In our adversary and trust model, we observe that proposed protocols usually are complex, and involve purpose-specific cryptography. To extract the verification codes in the voting phase, the Swiss Post protocol requires two round trips between the control components involving multiple zero-knowledge proofs, encryptions and hashes [11]. For the same purpose, CHVote uses instead a novel oblivious transfer scheme [6]. A more recent proposal by Hänni et al. managed to simplify the voting phase using voting codes, however opted for a verifiable shuffle in the setup phase over all possible voting choices, with its associated complexity [5]. In this work, we avoid the verifiable shuffle at the cost of an authenticated (but public, and therefore auditable) channel out of the trusted setup component.[2]

Contributions We present a internet voting protocol using short voting codes. Only as many different voting codes are needed as voting choices are available.

We achieve vote secrecy without trust in the voter's device. Further, we provide cast-as-intended, registered-as-cast, tallied-as-recorded, eligibility verifiability and universal verifiability. Trust is required in the setup component, and a collection of control components, of which only one needs to be honest. The adversary may control the network and the untrusted components.

The protocol uses few, efficient and readily available cryptographic building blocks. Besides the verifiable tally mechanism, we need no more advanced cryptography than signatures, notably we do not need zero-knowledge proofs or homomorphic encryption in the setup and voting phases.

---

[1] We note, and we have double-checked this fact with the Swiss Chancellery, that the current Swiss law and its derived ordinances do not forbid code voting.

[2] Consider the extensions proposed in section 3.4.

2   Voter's view

To introduce the protocol we 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 of how this might look like. First, the voter scans the QR code to login ①. Then, the voter enters the short voting codes of the voting choices they intend to vote for ②. The voting codes are sent to the voting system, which responds with a verification code each. If and only if these codes match to what is printed on the voting sheet ②, the voter is instructed to confirm the vote ③.

To perform their tasks as instructed, voters need to be able to scan QR codes and enter and compare short codes; all mechanisms already in use by existing large-scale systems (e.g. certificate scanning and second factor applications). To our knowledge, the usability of entering short voting codes in the voting context has not yet been explored. An extension of SPS incorporating (long) voting codes has however been shown to not reduce general usability [12].

The voter application needs to essentially only forward values to the voting system and back to the user, and needs not to encrypt or sign the vote. The application may guide the user with basic validation (e.g. ensuring the voting code entered for the first question is between 2 and 4).

Fig. 1. An example of a voting sheet using realistic parameters. ① contains a QR code with *Id*. ② contains the voting codes C and their respective vote verifications VV_C. ③ contains the CA (upper string) and the CV (lower string).

3   Protocol

The protocol is divided in three phases. In the setup phase, all parties receive required cryptographic material to run the voting and tally phase. In the voting phase, the voter submits their vote and performs their individual verifiability check. In the final tally phase, the votes are decrypted and counted, depending on the chosen tally mechanism.

Roles In the setup phase, the trusted *Administrator* decides on the parameters of the election (participating voters, voting choices, length of keys, etc). The trusted *Setup Component* then generates the corresponding key material and distributes it over secure channels to the other participants.

In the voting phase, the *Voter* casts and confirms their vote using their voting device over insecure channels. Voters can verify casting and confirmation is successful without trust in their device neither for privacy nor verifiability.

In all phases of the protocol, the *Control Components* guarantee correctness and privacy of the election. We require only a single control component to be honest for the properties to be preserved. The *Adversary* learns the data of untrusted protocol participants and can act on their behalf. Further, it can read, drop or add messages exchanged over untrustworthy channels.

Notation We choose random with $\overset{r}{\leftarrow}$. We denote modular addition with $\oplus_n$ for n the modulo value. If n is obvious from the context, we may omit it.

We use boldface for lists, for example $\mathbf{l} = [a_1, a_2,...]$. We call a list composed out of pairs a dictionary, for example $\mathbf{d} = [(a_1, b_1), (a_2, b_2)]$. We look for a match in a dictionary with $\leftharpoonup$ (i.e. $x \leftharpoonup (2, .) \in [(1, a), (2, b)]$ results in $x = b$). If no match is found, or more than one, the process terminates.

We denote the group of permutations of integers up to s as $\mathbb{P}_s$ (e.g. $\mathbb{P}_3$ includes $[1, 2, 3]$ or $[2, 3, 1]$). Using $*$, we apply permutations to themselves (e.g. $[1, 2, 3] * [2, 3, 1] = [3, 1, 2]$), and to the right value of the pairs in a dictionary (e.g. $[(1, a), (2, b), (3, c)] * [3, 1, 2] = [(1, b), (2, c), (3, a)]$).

When a party encounters an *assert* with a falsy expression (like *assert false* or *assert 0*), then the party aborts processing. We use $\overrightarrow{}_A$ and $\overrightarrow{}_S$ to denote communication sent over the authenticated and secure channel, respectively.

Parameters The authorities decide on the following parameters and algorithms:

- $\mathbb{Z}_{n_a}$ to pick authentication secrets from. The adversary is given their hashes, so n_a must be big enough for picked values to be hard to brute-force.
- $\mathbb{Z}_{n_v}$ to pick verification codes from. The adversary has a single try to convince the voter, so n_v must be big enough for picked values to be hard to guess.
- ***Id*** is a list of identifiers such that there is one for each eligible voter.[3]

---

[3] If an adversary guesses such an identifier, it may vote on the voter's behalf. While the voter will detect if a wrong vote has been cast, and consequentially does not confirm (which prevents tallying), they cannot use the system to cast their own vote anymore. Therefore, for availability, hard to guess identifiers are a good idea.

- *PtC* (*Plain to Codes*) is a dictionary which maps each plain vote $P \in \mathcal{P}$ to a unique code $C \in \mathcal{C}$ (e.g. $[('Yes', A), ('No', B), ('Abstain', C)]$).
- H is a pre-image resistant hash function (e.g. SHA256). The scheme needs to provide a hash function $\mathsf{Hash}(m)$ for message m.
- S is an EUF-CMA secure signature scheme (e.g. ECDSA). The scheme needs to provide a sign function $\sigma \leftarrow \mathsf{Sign}(sk, r, m)$ for secret key sk, randomness r and message m. Further, the scheme needs to provide a verification function $\mathsf{Verify}(pk, m, \sigma) \in \{0, 1\}$ for public key pk, message m and signature σ. For a valid key pair (pk, sk) it holds that $\forall r, m.\mathsf{Verify}(pk, m, \mathsf{Sign}(sk, r, m)) = 1$. We write $\mathsf{Sign}(m)$ and $\mathsf{Verify}(m, \sigma)$ when the other arguments are clear from the context.
- E is an IND-CPA secure public key encryption scheme (e.g. ElGamal) to produce ciphertext suitable as input for the privacy-preserving tally. The scheme needs to provide an encryption function $E \leftarrow \mathsf{Enc}(pk, r, m)$ for pk the public key, r the encryption randomness and m the message. Likely, Enc operates on an aggregated public key of which a private key share is at each control component. We write $\mathsf{Enc}(m)$ when the other arguments are clear from the context.

3.1 Setup

In the setup phase, the setup component generates key material and sends the result to the control components and to the voters.

Alg.: $\mathsf{GenPartialBallot}()$

for $C \in \mathcal{C}$ **do**
  $vv \xleftarrow{r} \mathbb{Z}_{n_v}$
  $Ctvv \leftarrow Ctvv \cup (C, vv)$

$ca \xleftarrow{r} \mathbb{Z}_{n_a}$
$cv \xleftarrow{r} \mathbb{Z}_{n_v}$

$p \xleftarrow{r} \mathbb{P}_{|\mathcal{C}|}$

return $(Ctvv, ca, cv, p)$

Algorithm 1: Generates a partial ballot.

Alg.: $\mathsf{MergePartialBallots}(\forall i \in [1, m].b^{(i)})$

$\forall i \in [1, m].(Ctvv^{(i)}, ca^{(i)}, cv^{(i)}, p^{(i)}) \leftarrow b^{(i)}$

for $C \in \mathcal{C}$ **do**
  $\forall i \in [1, m].vv^{(i)} \xleftarrow{} (C, .) \in Ctvv^{(i)}$
  $CtVV \leftarrow CtVV \cup (C, \oplus_{i=1}^{m} vv^{(i)})$

$CA \leftarrow \oplus_{i=1}^{m} ca^{(i)}$
$CV \leftarrow \oplus_{i=1}^{m} cv^{(i)}$

$PtC \leftarrow PtC * \prod_{i=1}^{m} p^{(i)}$

return $(CtVV, CA, CV, PtC)$

Algorithm 2: Merges partial ballots generated by Algorithm 1.
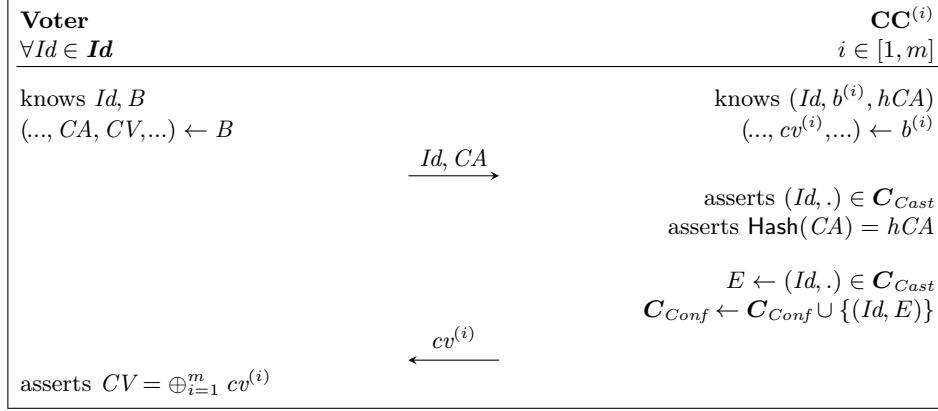

3.2 Voting phase

In the voting phase, the voter casts and confirms their vote together with the control components (see Protocol 2 (cast vote) and Protocol 3 (confirm vote)).

Voter	**Setup component**	**CC**$^{(i)}$
$\forall Id \in \boldsymbol{Id}$	$\forall Id \in \boldsymbol{Id}$	$i \in [1, m]$

$$\forall i \in [1, m].b^{(i)} \xleftarrow{r} \mathsf{GenPartialBallot}()$$
$$B \leftarrow \mathsf{MergePartialBallots}(\{ b^{(i)} \mid i \in [1, m]\})$$

$$\xleftarrow{\quad Id, B \quad}_{\text{S}} \qquad\qquad \xrightarrow{\quad Id, b^{(i)} \quad}_{\text{S}}$$

$$(\ldots, CA, \ldots, PtC) \leftarrow B$$
$$hCA \leftarrow \mathsf{Hash}(CA)$$
$$CtE \leftarrow \{(C, E) \mid C \in \mathcal{C}; E \leftarrow \mathsf{Enc}(C * PtC^{-1})\}$$

$$\xrightarrow{\quad hCA, CtE \quad}_{\text{A}}$$

Protocol 1: Setup phase where the setup component establishes key material for the control components (CC) and the voters.

Voter	**CC**$^{(i)}$	**CC**$^{(j)}$
$\forall Id \in \boldsymbol{Id}$	$i \in [1, m]$	$j \in [1, m]\backslash i$

knows Id, B knows $(Id, b^{(i)}, CtE)$
$(CtVV, ..., PtC) \leftarrow B$ $(Ctvv, ...) \leftarrow b^{(i)}$

decides plain vote P
$C \leftarrow (P, .) \in PtC$

$$\xrightarrow{\quad Id, C \quad}$$

asserts $C \in \mathcal{C}$
asserts $Id \notin \boldsymbol{C}_{Sync}$

$\boldsymbol{C}_{Sync} \leftarrow \boldsymbol{C}_{Sync} \cup \{Id\}$
$E \leftarrow (C, .) \in CtE$
$\sigma^{(i)} \leftarrow \mathsf{Sign}(Id, E)$

$$\xleftarrow[\text{A}]{\sigma^{(i)}} \xrightarrow[\sigma^{(j)}\ \text{A}]{}$$

assert $\forall j \in [1, m]\backslash i.\mathsf{Verify}((Id, E), \sigma^{(j)})$

$\boldsymbol{C}_{Cast} \leftarrow \boldsymbol{C}_{Cast} \cup \{(Id, E)\}$
$vv^{(i)} \leftarrow (C, .) \in Ctvv$

$$\xleftarrow{\quad vv^{(i)} \quad}$$

$VV \leftarrow (C, .) \in CtVV$
asserts $VV = \oplus_{i=1}^{m} vv^{(i)}$

Protocol 2: Voting phase (1/2) where the voter casts their vote. The control components (CC) check the authentication, synchronize state and then respond with the verification specific to the received vote. Finally, the voter checks the verification.

Voter	**CC**$^{(i)}$
$\forall Id \in \boldsymbol{Id}$	$i \in [1, m]$

knows Id, B	knows $(Id, b^{(i)}, hCA)$
$(..., CA, CV, ...) \leftarrow B$	$(..., cv^{(i)}, ...) \leftarrow b^{(i)}$

$$\xrightarrow{\quad Id, CA \quad}$$

asserts $(Id, .) \in \boldsymbol{C}_{Cast}$
asserts $\mathsf{Hash}(CA) = hCA$

$E \leftarrow (Id, .) \in \boldsymbol{C}_{Cast}$
$\boldsymbol{C}_{Conf} \leftarrow \boldsymbol{C}_{Conf} \cup \{(Id, E)\}$

$$\xleftarrow{\quad cv^{(i)} \quad}$$

asserts $CV = \oplus_{i=1}^{m} cv^{(i)}$

Protocol 3: Voting phase (2/2) where the voter confirms their vote. The control components (CC) check the authentication and then respond with a verification that the confirmation has been received. Finally, the voter checks the verification.

If the voter reaches the end of both protocols, the voting procedure was successful and no other actions are necessary. If otherwise the voter aborts, they are instructed to use a different voting channel.

The system includes the vote for tallying as soon as the voter entered the confirm authentication CA. But the voter may also participate over a different voting channel (e.g. because they did not receive a correct confirm verification), so duplicates over the different voting channels need to be dealt with.

3.3   Tally phase

At the end of the voting phase, each control component has a list of confirmed votes $\boldsymbol{C}_{Conf}$ which need to be tallied. All correctly processing control components have the same $\boldsymbol{C}_{Conf}$ in their local state. However, dishonest control components might have added, modified or dropped entries in their local $\boldsymbol{C}_{Conf}$, so we need to clearly define what list of ciphertext is to be tallied.

Definition 1. $\boldsymbol{C}_{Agre}$ *contains all ciphertext entries E for which it holds that for some voter Id, the CA is known and from all control components there exists a signature over (Id, E) (i.e. $\forall i \in [1, m].\exists \sigma^{(i)}.\mathsf{Verify}((Id, E), \sigma^{(i)})$).*

To establish $\boldsymbol{C}_{Agre}$, each control component sends $\boldsymbol{C}_{Conf}$ with the proof for each entry (CA and signatures over (Id, E)) to all other control components. Each control component then adds all E to $\boldsymbol{C}_{Agre}$ for which the definition holds, considering their own $\boldsymbol{C}_{Conf}$ and proofs, as well as the $\boldsymbol{C}'_{Conf}$ and proofs received from the other control components.

After establishing $\boldsymbol{C}_{Agre}$, the control components then execute the privacy-preserving verifiable tally-mechanism, with the ciphertext given by $\boldsymbol{C}_{Agre}$ as input. The correct execution of all the steps are provable to third parties.

3.4   Extensions

With the core protocol given, we now improve on functionality and security.

Practical availability As presented here, the voter directly communicates with the control components, hence an adversary may try to flood the control components with bogus requests. As soon as a single control component is unable to process more requests, honest voters can no longer cast and confirm votes. However, verification of authentication and validation of votes is efficient: Only hash and set membership checks are needed (but e.g. no asymmetric cryptography). Further, the checks operate solely on public data.

To improve availability, an untrusted server component can be introduced specifically hardened to resist such flooding attacks. The server component is placed in between the voter and the control components. If said system then operates correctly, any request which reaches the control components will pass authentication and validation, minimizing load on the control components' side.

Supporting multiple elections, voting choices and eligibility So far, we only described how a single popular vote or election will proceed for a single voting choice. However, votes are usually held over multiple issues at the same time. Further, depending on the issue voted on, multiple voting choices are possible. Also, voters might not all be of the same eligibility. To ensure the protocol is applicable to many voting scenarios, we aim to support k-out-of-n elections with varying eligibility per voter, which for example corresponds to elections in Switzerland [6, Chapter 2.2.2/2.2.3].

To support multiple issues at the same time, additional sets of voting codes $\mathcal{C}$ are chosen. Consequentially, the setup phase handles additional permutations corresponding to each additional set of voting codes. To support k-out-of-n issues, the voter has to submit exactly k voting options out of $|\mathcal{C}| = n$. To support different eligibilities, voters are restricted in the sets of voting codes they can submit a vote for. Consequentially, the vote validation (see Protocol 2) is extended to ensure the sets of voting codes submitted, and the number of submitted voting options per set of voting codes, is valid.

To improve privacy of voters with restricted eligibility, the tally mechanism may tally each issue (i.e. each set of voting options) separately.

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

To make an audit of the setup component practical, we first ensure it only runs deterministic algorithms. We implement this by running GenPartialBallot() at each control component instead of the setup component, and consequentially reverse the corresponding secure channel (now from the control components to the setup component). Note how MergePartialBallots() ensures that if at least one $b^{(i)}$ is generated honestly, the adversary has no advantage to guess values in B.

To audit the setup component, we add n additional entries to $\boldsymbol{Id}$, 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 $\boldsymbol{Id}_A^{(i)} \subset \boldsymbol{Id}$ for $|\cup \boldsymbol{Id}_A^{(i)}| = n$. Any corresponding Id can no longer be used to cast votes, but instead the control components publish $b^{(i)}$. The control components then assert that B, hCA and (C, E) have been generated correctly.

Making the protocol robust outside the adversary model As presented, the protocol is secure within the adversary model. However, we may also want to preserve security properties outside the adversary model. So even if a presumed honest party is controlled by the adversary, given some additional assumptions, we still want to deliver some guarantees.

Note how the mechanisms presented here cannot prevent all attacks of a presumed honest party. For example, no mechanism prevents a dishonest setup component to person-in-the-middle the individual verifiability check, and if all control components are dishonest, they can easily drop votes. But the mechanisms might still manage to increase security in practical scenarios.

To prevent a dishonest setup component to insert votes for abstaining voters, the control components can inform the user directly about whether their vote is considered in the tally (i.e. is part of $\boldsymbol{C}_{Conf}$) after the voting phase has ended. This assumes an authenticated channel to the voter for that purpose.

To prevent a dishonest setup component to target specific voters (i.e. switching voting and verification codes for voters likely voting in a certain way), the secure channel to the voter can obfuscate to the setup component which actual voter (name, address) is assigned which Id. For example, if the secure channel to the voter is implemented over postal mail, the voting sheets can be printed first, physically shuffled, and only then be put into addressed envelopes.

To prevent jointly dishonest control components to insert or modify votes, the setup component can generate a signature key pair per voter, of which the public key is published. The voter device scans the private key together with the identification, and signs the submitted vote.[4] Only votes which have a valid signature are included in the tally, to be checked by an auditor. This assumes an honest setup component, an honest voter device and an honest auditor.

To prevent jointly dishonest control components to learn the vote of a voter, an additional tally component can be introduced which contributes part of the decryption key and participates in the tally. The tally component is therefore included in the group of control components of which only one needs to be honest for the security properties of the tally mechanism to be preserved. From a practical point of view, the tally component may indeed be easier to secure, as it does not need to participate in the voting phase (hence needs not be "online"). If this same tally component is involved in proving the participation of voters, it further ensures that dishonest control components cannot drop or add votes.

---

[4] Note how this does not change the voter interaction.

References

[1] Budurushi, J., Neumann, S., Olembo, M.M., Volkamer, M.: Pretty understandable democracy-a secure and understandable internet voting scheme. In: 2013 International Conference on Availability, Reliability and Security. pp. 198–207. IEEE (2013)

[2] Chaum, D.: Sure Vote: Technical Overview. In: Proceedings of the workshop on trustworthy elections (WOTE 2001) (2001), http://www.vote.caltech.edu/wote01/pdfs/surevote.pdf

[3] Cortier, V., Filipiak, A., Lallemand, J.: BeleniosVS: Secrecy and verifiability against a corrupted voting device. In: 2019 IEEE 32nd Computer Security Foundations Symposium (CSF). pp. 367–36714. IEEE (2019)

[4] Federal Chancellery: Federal Chancellery Ordinance on Electronic Voting. https://www.fedlex.admin.ch/eli/cc/2022/336/en (December 2013), https://www.fedlex.admin.ch/eli/cc/2022/336/en, accessed at 2022-11-06

[5] Haenni, R., Koenig, R.E., Dubuis, E.: Private Internet Voting on Untrusted Voting Devices. In: International Conference on Financial Cryptography and Data Security (2023)

[6] Haenni, R., Koenig, R.E., Locher, P., Dubuis, E.: CHVote System Specification. IACR Cryptol. ePrint Arch. **2017**, 325 (2017)

[7] Helbach, J., Schwenk, J.: Secure internet voting with code sheets. In: International Conference on E-Voting and Identity. pp. 166–177. Springer (2007)

[8] Joaquim, R., Ferreira, P., Ribeiro, C.: EVIV: An end-to-end verifiable Internet voting system. Computers & Security **32**, 170–191 (2013)

[9] Joaquim, R., Ribeiro, C., Ferreira, P.: Veryvote: A voter verifiable code voting system. In: International Conference on E-Voting and Identity. pp. 106–121. Springer (2009)

[10] Ryan, P.Y., Teague, V.: Pretty good democracy. In: International Workshop on Security Protocols. pp. 111–130. Springer (2009)

[11] Schweizerische Post: Swiss Post Voting System: System Specification. Version 1.1.1. report, Schweizerische Post (October 2022), https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/afbd7555b3f87092596a5203f022c00dcc0fd390/System/System_Specification.pdf, accessed at 2023-11-21

[12] Volkamer, M., Kulyk, O., Ludwig, J., Fuhrberg, N.: Increasing security without decreasing usability: A comparison of various verifiable voting systems. In: Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022). pp. 233–252 (2022)