Review of the Symbolic Proofs for the Swiss Post Voting System's Cryptographic Protocols

Saša Radomirović, Ioana Boureanu, and Steve Schneider
Surrey Centre for Cyber Security, University of Surrey, UK

17 October 2022

1 Scope and Methodology

We reviewed the symbolic models and proofs of the Swiss Post Voting System's cryptographic protocols. The reviewed material consists of 6 ProVerif files. Two of these files concern vote privacy, another two concern individual verifiability of votes, one concerns universal verifiability of votes and the final file demonstrates that the individual verifiability model of a previous version of the voting system is sufficiently faithful to capture an attack reported by Haines.

Our work falls into Scope 1 of the Federal Chancellery's (FCh's) Audit concept [AuC22] and is in fulfillment of Article 26.1.1 of the Annex of [OEV22], but *restricted to the Symbolic Proofs*.

We note that Scope 1 of [AuC22] also entails the evaluation of the protocols' *cryptographic proofs*. These are not in scope of this review, nor are computational cryptography concerns such as resilience to quantum computing.

Documents Reviewed and Supportive Material. Our report is based on the examination of the following versions of documents:

- The documents comprising the symbolic models and proofs of the Swiss Post Voting System's cryptographic protocols as published on 8th July 2022 [Mod22a] and [Mod22b].
- Version 1.0.0 of the Swiss Post Voting System Specification document [Sys22].
- Version 1.0.0 of the Swiss Post Voting System Verifier Specification document [Ver22].
- Version 1.0.0 of the Cryptographic Primitives of the Swiss Post Voting System [Cry22].
- The FCh's Ordinance on Electronic Voting [OEV22] that entered into force on 1st July 2022 and its explanatory report [ER22].

The symbolic models and documentation of the Swiss Post Voting System were downloaded from the public repository at the following URL: `https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation`. The links to the specific versions of the files we reviewed are given in the references.

Assessments Undertaken. Our work consisted of

1. Assessment of the **correspondence between the aforesaid symbolic models and the system specification in [Sys22]**.

   A symbolic formal model is necessarily an abstraction of the system specification. This abstraction limits the extent to which a proven security claim in the symbolic model provides assurance of the corresponding security goal in the specified system.

   To assess the faithfulness of the symbolic models, we carefully considered the system [Sys22] and verifier [Ver22] specifications and each of the symbolic models implemented in the ProVerif files. To this end, where necessary to understand the details of the cryptographic protocol, we also consulted the document specifying the Cryptographic Primitives of the Swiss Post Voting System [Cry22].

   Assessment of the faithfulness of the symbolic model entailed checking which protocol actors, execution flows and cryptographic terms in the system specification are retained by the symbolic model and what abstractions and simplifications are employed in the symbolic model.

2. **Examination of the symbolic proofs** of the protocols' privacy, individual verifiability and universal verifiability properties in the symbolic models with respect to the requirements in Article 2 of the Annex of [OEV22].

   In this examination, we did the following:

   - We verified that the symbolic models' trust assumptions with respect to system actors and channels are not stronger than the trust assumptions stated in Article 2 of [OEV22, Annex].
   - We verified that the symbolic models' adversary model is not weaker than the attacker assumptions in Article 2.3 of [OEV22, Annex].
   - We verified that the security requirements in Article 2 of [OEV22, Annex] are covered by the formalization of the privacy and verifiability security properties in the symbolic models.

   We used ProVerif version 2.04 to verify the correctness of the security claims and we consulted the FCh's explanatory report [ER22] for further clarification of the relevant Articles in [OEV22].

2 Summary of results

2.1 Correspondence between the symbolic model and the system specification

Both the symbolic models and the system specification are of a very high quality. The symbolic models are well commented and their notation generally matches the notation used in the system specification [Sys22].

The correspondence between the symbolic models and the system specification has several limitations. Some of the main limitations follow, whereas others are described in more specific contexts later in the document.

Limitations of the Correspondence between the Models and the Specifications:

- **Voter authentication is not specified** in [Sys22]. The symbolic models' initial state is after a voter is successfully authenticated.
- The symbolic models do **not consider write-in votes**.
- The symbolic models are significant abstractions of the system specification. They consider only the main protocol flow thus **excluding, for example, system behaviour in case of errors** and the voters' ability to stop and later resume execution of the protocol.

  The symbolic privacy model **excludes the audit phase of the protocol**. Neither the symbolic models for privacy nor for verifiability include the complete role of the auditor.

We elaborate on these limitations and their possible implications in Section 3.

2.2 Compliance with the requirements in [OEV22]

Under the limitations stated in Section 2.1 and detailed in Section 3, the symbolic proofs provide evidence that the specified system [Sys22] satisfies the requirements in Article 2 in the Annex of [OEV22].

3 Detailed Results

3.1 System Specification

The system specification is generally sufficiently precise to be able to understand the logical flow of the cryptographic protocol necessary for a symbolic model. It also provides precise pointers to the detailed specification of the employed cryptographic algorithms. Its notation follows a logical and consistent naming structure which greatly helps the reader to cope with the complexity of the protocol and cryptographic keys infrastructure.

We list below the few exceptions to the rule and discuss their significance afterwards.

3.1.1 Voter Authentication

No voter authentication protocol is specified in the System Specification.

On page 10 of [Sys22], it is said that the voter enters the start voting key (`SVK`) to authenticate herself, while the symbolic model's `Readme` file mentions that the voter authentication protocol is a challenge-response protocol. This is insufficient information for a precise understanding of the cryptographic protocol used for voter authentication.

We note that the `SVK` is said to be a shared secret between the Setup Component and the Voter Client [Sys22, page 44] and that it decrypts the voter's verification card keystore ($\texttt{VCks}_{\texttt{id}}$) which provides the necessary cryptographic key to submit a vote.

3.1.2 Voter Identification

Throughout the cryptographic protocol specification, the key to identify the voter's credentials is the verification card identifier, $\texttt{vc}_{\texttt{id}}$. This is an identifier that the Setup Component randomly generates. It is used as input to several cryptographic algorithms, but the specification does not state how this identifier is learned or communicated at the beginning of the voting protocol, where it is necessary for the voting server to send the correct verification card keystore ($\texttt{VCks}_{\texttt{id}}$) to the voter's voting client.

3.1.3 Channel Security

In the configuration phase of the voting protocol, the Control Components generate cryptographic keys that are communicated to the Setup Component. The communication between Setup and Control Components (CCR/CCM) runs via the untrusted system (Voting Server) and is considered to be an untrustworthy communication channel. This conforms with the channel provisions and trust assumptions specified by the FCh Ordinance [OEV22, Annex Article 2.2 and 2.10].

To protect the communication between the Setup and Control components the communicated messages are cryptographically signed. This necessitates a prior exchange of public key certificates. The certificate generation and verification is only specified informally in the final Chapter of [Sys22]. It states that "*Each trustworthy party generates a signature key pair and distributes the corresponding certificate to the other trustworthy parties and the auditors.*" and that "*All messages exchanged between the trustworthy parties are signed [...]*".

More details are given in the Cryptographic Primitives Specification [Cry22, Section 6.2], where it is stated that "*The distribution of certificates must rely on an existing authenticated channel and the process for the distribution must be documented in sufficient detail.*" It also states that the verification of certificates is a "*human-led*" process but no further details of this process are provided.

The ordinance [OEV22, Appendix 2.13.3] allows for trustworthy channels to distribute electronic certificates among system participants. The established procedures appear to comply with the ordinance and we do not see any security

issues. However, we also cannot exclude the potential for security issues, since the informal description leaves room for different interpretations.

3.1.4 Why the highlighted issues matter

We highlighted the above issues in the System specification for two reasons.

1. The system specification is the central reference document for the verification of the voting system. A complete unambiguous system specification is necessary to ensure that the implementation of the system corresponds to the cryptographic and symbolic models in which the system's security is verified.

   In contrast, ambiguities or incompleteness in a system specification bear the risk that security problems are missed due to different interpretations by those who formally model the specification and those who implement the specification. An assessment of the correspondence between the system specification and implementation is not in scope of this review.

2. While the initial state assumptions made in the symbolic models look reasonable to us, the highlighted issues in the system specification limit our ability to assess the justification of the initial state assumptions made in the symbolic models. We give two examples of what could go wrong in Section 3.3.

3.2 Symbolic Models and Proofs

The symbolic models combined implement most of the functionality of the Swiss e-voting system as specified in [Sys22] and their proofs satisfy the required security properties *within the model*. The models are well made, both in terms of documentation of their functionality and limitations and in terms of modeling techniques used. They employ standard formalization techniques for the required security properties. The discussion in this section focuses on the limitations of the models with respect to their faithfulness to the System Specification and their potential implications.

While the models combined cover most of the system specification's functionality, each model is tailored to the specific security property being proven and excludes functionality which was deemed to be unrelated. This results in three different models which we discuss separately when necessary below. Several comments pertain to all three models and there are consequently some issues that are raised repeatedly as a general comment on all models and as a comment for a particular model.

3.2.1 Absence of write-in votes

Write-in votes are not formally modeled in any of the three symbolic models. The justification (in the `Readme` file accompanying the symbolic models) is that

"*most election events have a fixed set of voting options, and even if write-in options are allowed, only a small percentage of voters use write-ins.*"

We also observe that write-ins are dealt with differently than the fixed set of vote options in Algorithm 5.2, `CreateVote`, of the system specifications and in Algorithm 7.5 of the cryptographic specification.

Omitting the modeling of write-ins is acceptable for elections where the system does not allow for write-ins. However, voting with a ballot that allows for write-ins does not achieve Individual Verifiability, because the Voter cannot verify that the write-in was cast as intended and the untrusted Voting Client can change the write-in. [OEV22, Article 16] makes a provision for exceptions to be applied for individual requirements and the FCh's Explanatory Report [ER22] states waiving the individual verifiability of write-in votes as an example of a possible exception that could be applied for.

Why the modelling of write-ins should be considered. Even if individual verifiability of write-ins were to be exempt as in the example of [OEV22, Article 16], the write-ins should then still be proven to be private and universally verifiable and it should be proven that they do not affect the verifiability and privacy of the other voting options.

The fact that write-ins are not treated the same by the cryptographic algorithms as the fixed voting options indicates that the existing proofs may not trivially extend to the write-ins.

3.2.2 Privacy models

The privacy model assumes that one of the four mixing control components (CCMs) is trustworthy, while all the return codes control components (CCRs) are only considered trustworthy during the setup phase. Two variants of the model are provided, one where the first CCM is trustworthy and one where one of the last three CCMs is trustworthy.

Vote privacy is established with a standard technique that demonstrates the adversary's inability to tell for a pair of honest voters and their votes which voter has cast which vote.

The vote privacy property is also used to argue in the accompanying documentation (`Readme` file) that no premature partial results are released "*provided that the electoral authorities perform the final decryption of the votes only after the election event ended*".

Limitations and why they matter. The model abstracts away the functionality of many protocol components by declaring them untrustworthy (e.g., control components), simplifying their roles (e.g., auditors), or focusing on the main protocol flow and ignoring the possibility of alternative behaviour (e.g., a voter stopping and resuming the voting process). This simplifies the complexity and to a certain degree improves readability of the model.

Such a very selective modeling can be necessary to ensure that the verification tool is able to complete its proof search. However, it also carries the risk that possible attacks are abstracted away.

One concrete example of loss of functionality due to the abstractions and simplifications is that in the two given models the honest agents cannot reach the end of their voting process.[1]

We give a further illustration of potential risks in Section 3.3.

3.2.3 Fixed voting options and simplifications of control components

Voters and voting options. The models fix the numbers of voting options and voters that are considered.

- The verifiability models consider 4 voting options. Two separate files are provided for individual verifiability that differ in the voter's ability to select 1 or 2 of these options. For universal verifiability the voter selects 2 options. In the privacy model the voter selects 1 out of an undetermined number of options.
- The universal verifiability model limits the number of honest voters to 2 and the individual verifiability to 1 voter. There are no fixed limits for the number of dishonest voters in the verifiability models. The privacy models consider 2 honest and 1 dishonest voter.

While a more general setting is preferable, such choices are not unusual and in some cases supported by theorems.

Control components. The models also simplify the control components, by leaving out the modeling of the Tally Control Component and for verifiability reducing the number of control components to 2 instead of 4. Here a justification for the simplification is desirable, since the modeling of such control components is not a common feature in cryptographic protocols.

The omission of the Tally Control Component is justified in the accompanying `Readme` file by considering the Tally Control component as an untrustworthy element that verifies proofs and shuffles. However, in [Sys22], the Tally Control Component also performs the final decryption of votes which the abovementioned justification does not seem to consider.

3.2.4 Modelling of the auditors' role

Auditors are modeled in the universal verifiability proof model to the extent that is necessary to verify correct decryptions.

They are not modeled in the individual verifiability nor in the vote privacy models. Moreover, the role of the auditors in the voting system, described in the Verifier Specification [Ver22], is also to verify the configuration phase prior to the voting phase. This is not modeled in any of the three proof models.

[1] The required input in line 283 in both of the privacy files [Mod22a] is not received.

3.2.5 Formal modelling of corruptions

In all three models, corruption is often modelled by dropping a part of a process or even a full process of the "main process" specification, and by leaking the "missing" processes' long-term key(s) to the attacker. This is standard and the most general (i.e., strongest) attacker that symbolic/Dolev-Yao verification can offer.

Positives on modelling corruptions. This type of corruption is generally modelled in a sensible way and in keeping with the threat model of the specification. For instance, when a voter is dishonest, the attacker can manipulate its inputs (i.e., gets all the details out of voter's process via a sub-process called `AliceData(·)`), but the attacker does not get the id of the voter linked to this data. This modelling is in line with the Setup/registration being trusted.

Similarly, in e.g., universal verifiability, the private key of one CCM (i.e., CCM1) is made available to the attacker, and then this process has most parts as "free contexts", i.e., the attacker is able to let it perform an almost arbitrary execution instead of an honest CCM run.

Aspects left unclear with respect to corruptions. In these somewhat fixed models (i.e., 2 CCMs, 4 votes), there are mixtures of generic corruption (by long-term key leakage) with normal executions of the processes done by means of events that are at times prescriptive.

For instance, in the universal verifiability model, CCM1 is fully corrupt (i.e., its private key have leaked), but events are used to link parts of a correct execution of CCM1 with the execution of CCM2.

It is not always clear how much this type of "weaving in" of dishonest and honest processes restricts the arbitrary corruption and if it is still in-keeping with the threat-model intended here.

Possible Shortcomings. Since parts of the systems are not formalised, including missing the audit phase of the voting in the privacy models, the current modelling of corruptions eludes a malicious server possibly manipulating (encrypted) write-ins and an auditor "colluding" with a malicious server in reading the modified write-ins.

Should the above be modelled, it is possible that (extended) versions of the vote-privacy query could fail. We detail a hypothetical scenario in Section 3.3.

3.2.6 Formal model of universal verifiability

The Universal Verifiability Requirement [OEV22, Article 2.6] is that *"The auditors receive a proof in accordance with Article 5 paragraph 3 letter a in conjunction with Article 6 letters a and c to confirm that no attacker:*
(a) after the votes were registered as cast in conformity with the system, has altered or misappropriated any partial votes before the result was determined;

(b) has inserted any votes or partial votes not cast in conformity with the system which were taken into account in determining the result."

[OEV22, Article 5.3] states: *"The requirements for universal verifiability are as follows:*
(a). The auditors receive proof that the result has been established correctly; the proof confirms that the result ascertained includes the following votes:
1. all votes cast in conformity with the system that have been registered by the trustworthy part of the system,
2. only votes cast in conformity with the system,
3. all partial votes in accordance with the proof generated in the individual verification process
(b). The auditors evaluate the proof in an observable procedure; to do so, they must use technical aids that are independent of and isolated from the rest of the system."

The formalisation of this requirement in the file containing the universal verifiability model [Mod22b] is a correspondence query. Expressed in natural language, this query states: if an auditor successfully ends their role (i.e., reaches the `HappyAuditor`(.) event) then the CCMs have correctly performed the mixing beforehand. The auditor's process reaches the end if decrypted votes are successfully checked. Since CCM2 is honest and CCM1 is dishonest, this query checks that correct mixing is attested by auditors even in the presence of corrupt control components.

We believe that the aforesaid query encodes a weaker version of points (a) and (b) of the requirements in [OEV22, Article 2.6] and, together with the model, weaker versions of [OEV22, Article 5.3] and of [OEV22, Article 5.1]. That is, it is not that adversarially inserted, changed, or removed votes are explicitly detected by the model/query in Proverif (i.e., this is not proven). What is proven is that correct mixing and decryption occurs in CCM2, even if CCM1's private key is leaked (which entails an arbitrary CCM1). In other words, vote manipulation inside the corrupt CCM1 is not modelled explicitly (which is fine), however, the detection of any such manipulation by an auditor or a query establishing absence of such a manipulation is not modelled either.

3.3 Potential vote-privacy issues due to under-specification

To illustrate the significance of our points raised in the preceding subsections, we give sketches of two attacks on vote privacy that could be missed.

The first attack shown in Section 3.3.1, observed by Oliver Spycher, directly demonstrates the importance of formally specifying all used protocols in the System Specification.

The second attack shown in Section 3.3.2 additionally demonstrates the importance of modeling all protocol features.

3.3.1 An attack on vote-privacy (if SVK leaks)

The attack below demonstrates that system flaws could be missed due to the omission of the authentication protocol in the System Specification [Sys22].

We interpret the System Specification's statement that voters are authenticated by sending SVK to the Voting Server to mean that the Voting Server learns the key SVK. The Voting Server can then compute the voter's secret key $\mathtt{k}_{\mathtt{id}}$.

If the Voting Server knows $\mathtt{k}_{\mathtt{id}}$ then it can learn how the particular voter voted from the $\mathtt{pCC}_{\mathtt{id}}$ term. This term is a list of prime numbers each corresponding to a particular choice made by the voter and each raised to the power $\mathtt{k}_{\mathtt{id}}$. Since the primes are the same for all voters and publicly known, the Voting Server can test whether the voter selected a particular voting option.

The Voting Server learns pCC either by receiving it from an untrustworthy CCR component or by decrypting the term E2 with the information in $\mathtt{d}_{\mathtt{j}}$ that the CCR components send to the Voting Server as part of the regular protocol flow. The information flow can be seen in Fig. 8 and the necessary computational steps can be found in Algorithms $\mathtt{PartialDecryptPCC}_{\mathtt{j}}$ and $\mathtt{DecryptPCC}_{\mathtt{j}}$ in [Sys22].

The vote-privacy attack in Section 3.3.1 in the symbolic model. While the issue raised here concerns an omission in the System Specification, it also provides an opportunity to test the symbolic models' faithfulness.

We made a simple change in the ProVerif vote-privacy files [Mod22a]; this change allows SVK to leak to the attacker. We tested whether this attacker can distinguish how voters vote, and indeed the attacker can. The ProVerif tool finds a pCC-based test that makes the two voting processes *not diff-equivalent*. It is known that diff-equivalence failure does not imply observational equivalence failure. That said, the test found by ProVerif corresponds to the attack we described in Section 3.3.1.

3.3.2 Other potential vote-privacy issues due to abstractions in the formal model

A further attack is described below, building on the attack in Section 3.3.1. It demonstrates hat there could be an undesirable system behaviour that would be missed due to the lack of a specification for and modeling of the authentication protocol, together with the modeling of write-ins, and the modelling of auditors (in the privacy model).

The attack additionally uses the flaw discovered by Pascal Junod that the RecursiveHash function may not be collision resistant[2]. This is a flaw that a symbolic model cannot detect due to the perfect cryptography assumption. A symbolic model can be amended to analyse the consequences of such a flawed cryptographic primitive. A suitably faithful model would be expected to find an

[2] https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/issues/37

attack, such as the attack described below. A symbolic model that lacks functionality due to simplifications and abstractions could, however, miss attacks.

Another attack on vote-privacy (if `SVK` leaks). We assume that the Voting Server and one Auditor collude to find out how a targeted voter voted.

We again rely on the System Specification's statement that voters are authenticated by sending `SVK` to the Voting Server and thus assume that the Voting Server knows the voter's secret key $\texttt{k}_{\texttt{id}}$. When the Voting Server receives an encrypted vote, the Voting Server will change the encryption of the write-ins. Since the Voting Server does not know the randomness that was used for the encryption, this change will not result in a meaningful entry, but it will serve to mark the voter's ballot. There are two safeguards in the `CreateVote` algorithm that aim to prevent the modification of votes, namely `GenExponentiationProof` and `GenPlaintextEqualityProof`. The Voting Server has the required knowledge ($\texttt{k}_{\texttt{id}}$) to compute a correct `GenExponentiationProof`.

The encryption of write-ins only enters the `GenPlaintextEqualityProof` in the computation of the `RecursiveHash` function. The Voting Server abuses the lack of collision resistance to compute a ciphertext for the write-ins that produces the same hash as the voter's original ciphertext.

The voter cannot notice the change, because write-ins do not have the individual verifiability property.

Finally, when the votes are decrypted to be tallied, the Tally Control Component sends the results of the decryption and mixing process to the auditors for verification. The malicious Auditor colluding with the Voting Server will be looking for votes that have nonsensical write-in entries to identify how the voters targeted by the Voting Server have voted.

4 Recommendations

4.1 System Specification

The System Specification should specify the voter authentication protocol and clarify how the $\texttt{vc}_{\texttt{id}}$ term is initially communicated to the relevant parties.

The clarity of the system specification would also greatly benefit from a more formal description (in Chapter 7) of the measures taken in the distribution of public key certificates and how this intersects with the role of the auditors to ensure channel security.

Finally, the Electoral Board's passwords appear to be hashed and sent to the Tally Control Component (in Figure 7 [Sys22]). This is not described in the text. This feature should be explicitly described in the System Specification and included in the symbolic models.

4.2 Symbolic Models

4.2.1 Increase faithfulness

In the interest of increasing the faithfulness of the symbolic model and thus further increasing the level of assurance their proofs of security provide, we recommend that the models include more of the voting systems' functionality. This entails (1) explicitly modeling write-in votes if the system will be used in elections that require them, (2) modeling alternative protocol flows, such as the voter's ability to stop and resume voting, (3) including missing sub-protocols, protocol components, and roles, i.e., the voter authentication protocol, the Tally Control component, and the Auditor role.

For example, as noted in Section 3.2.2 the Auditor role and offline Tally Control Component which performs the final decryption of the votes are not modeled.

Modeling the Auditor role would explicitly provide the adversary with all information including the full list of decrypted ballots that is available to the auditors. In the present model, the adversary's knowledge of this information is obscured by the mixnet construction.

In addition to achieving a more faithful model, modeling the Tally Control Component would be useful to cover the case where Swiss Post operates all of the CCR/CCM control components. In that case, by [OEV22, Appendix 2.9.3.3], none of these control components can be considered trustworthy for vote privacy. A proof of privacy where the CCR/CCM control components are not trustworthy, but the Tally Control Component is trustworthy would demonstrate that the control components operated by Swiss Post cannot break secrecy nor provide premature partial results.

4.2.2 Document reasons for limitations

There are, of course, practical limits that restrict what can be included in a symbolic model. The modeler of a cryptographic protocol is naturally more keenly aware of these limits than the reviewer. We therefore encourage the inclusion of comments that clarify these issues, such as has been done, for instance, in the file `vote_privacy_CCM2-3-4.pv`, where a looping problem necessitated a simplification.

4.2.3 Give evidence of functionality

Finally, the complexity of the symbolic models themselves is high enough, that we recommend including more evidence of desired functionality. The inclusion of the ability to find Haines' attack is such an example, but we also encourage demonstration of reachability or adversary knowledge. For biprocesses this may require a separate model that projects onto one of the two processes.

References

[AuC22] *Audit concept v1.4 – For examining Swiss internet voting systems.* Swiss Federal Chancellery, 2022.

[Cry22] Cryptographic Primitives of the Swiss Post Voting System – Pseudo-code Specification. 2022. Version 1.0.0. `https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives/-/tree/crypto-primitives-0.15.2.6`. Accessed 17 October 2022.

[ER22] *Partial revision of the Ordinance on Political Rights and total revision of the Federal Chancellery Ordinance on Electronic Voting (Redesign of Trials) – Explanatory report for its entry into force on 1 July 2022.* Swiss Federal Chancellery, 2022.

[Mod22a] Symbolic Analysis of the Swiss Post Voting System – Vote Privacy, July 2022. `https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/documentation-0.8.5.8/Symbolic-models/privacy`. Accessed 17 October 2022.

[Mod22b] Symbolic Analysis of the Swiss Post Voting System – Individual and Universal Verifiability, July 2022. `https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/documentation-0.8.5.8/Symbolic-models/verifiability`. Accessed 17 October 2022.

[OEV22] *Federal Chancellery Ordinance on Electronic Voting.* Swiss Federal Chancellery, July 2022.

[Sys22] Swiss Post Voting System – System Specification. 2022. Version 1.0.0. `https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/documentation-0.8.5.8/System/System_Specification.pdf`. Accessed 17 October 2022.

[Ver22] Swiss Post Voting System – Verifier Specification. 2022. Version 1.0.0. `https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/documentation-0.8.5.8/System/Verifier_Specification.pdf`. Accessed 17 October 2022.