Examination Report on the Swiss Post e-Voting System

Thomas Haines, Olivier Pereira, Vanessa Teague
thomas@hainest.com, olivier.pereira@uclouvain.be, vanessa.teague@anu.edu.au

September 20, 2022

Contents

1 Executive Summary

The system continues to improve and many previous issues have been resolved or clarified. However it remains the case that critical parts of the system are unspecified and no complete version of system's source code—encompassing the various components and the verifier—has been published. Given these circumstances, we cannot at this time make anything except interim remarks about the (non)-compliance of the system with the Federal Chancellery Ordinance on Electronic Voting.

Most of the system specification has now reached a reasonable state of maturity. Various challenges remain open though, and we point:

- Open questions regarding the alignment between the security model required by the Ordinance and the one used in the security proofs.
- Missing elements in the verifier specification, that break universal verifiablity.
- Missing element in the protocol specification and proof that specify how inconsistent views by Control Components are handled.
- Several technical difficulties in the security definitions and proofs – many seem easy to fix, but some remain unclear to us at this point.

With regards to Scope 2, the verifier code—which is key for all security properties—has been out for less than a month at the timing of writing; this would not have been sufficient time under the best of circumstances. Moreover, there have been numerous difficulties including missing dependencies, failing unit tests, undocumented architecture decisions, and unvalidated inputs.

Our initial impression is that much of the code is in good condition with notable exceptions; however, at present the code as a whole lacks maturity; this is especially true in terms of security and we have been able to find several exploitable vulnerabilities which violate the key requirement of complete verifiability; we expect that more similar vulnerabilities exist in the code at present.

2 Scope 1: Cryptographic protocol

2.1 Scope

Our analysis is based on the documents available for the review of the Swiss Post e-voting system and the Swiss Federal Chancellery's latest version of the Ordinance on Electronic Voting (OEV). These documents are listed here, together with the name that we will use to refer to them in the text:

- Federal Chancellery Ordinance on Electronic Voting (OEV) of May 25, 2022 (status as of July 1, 2022) [1] – (*Ordinance*).
- Swiss Post Voting System – System specification, Version 1.0.0, June 24, 2022 [6] – (*Specification*).
- Protocol of the Swiss Post Voting System – Computational Proof of Complete Verifiability and Privacy, Version 1.00, July 29, 2022 [3] – (*Proof*).
- Cryptographic Primitives of the Swiss Post Voting System – Pseudo-code Specification, Version 1.0.0, June 24, 2022 [2] – (*Primitives*).
- Swiss Post Voting System – Verifier specification, Version 1.0.1, August 19, 2022 [8] – (*Verifier*).

Our analysis also took into account the "Response to examination report" documents issued by Swiss Post regarding Scope 1 & 2 of the 2021 review (all dated of April 20, 2022), as well as the inputs from various interactions with the Federal Chancellery and Swiss Post, whom we all thank for their help and insight.

The system specification and protocol proof continue to improve, including compared to the status of 2021. Still, the material remains incomplete, sometimes inconsistent, and only partially aligned with the ordinance. Though this may be mostly cosmetic in many places, it seems that none of key requirements are satisfied in the system as currently presented. The points below highlight problems with the current documents, they should not be interpreted as exhaustive: the points that are raised may require changes and clarifications that are significant enough to require another careful pass once the current open questions are addressed.

2.2 Alignment of the security model with the Ordinance

2.2.1 Role of the auditors

There are discrepancies between the role of the Verifier (or the Auditor's technical aid, in the wording of the Ordinance and according to Table 1 of the Proof document) and the Ordinance.

First, Table 2 of the Proof document, following the Ordinance, indicates that there is no channel from the Auditor's technical aid to any system component.

This means that these technical aids cannot play any *interactive* role in the protocol execution.

However, as explained in Section 4.4 of the Specification document and discussed in Section 11.4 of the Proof document, the protocol requires the verifier to perform the VerifyConfigPhase algorithm at the end of the SetupVoting algorithm and to report an invalid execution to the Setup Component. Section 11.4 of the Proof document acknowledges that this is a departure from the prescriptions of the Ordinance, and justifies it by saying that, in reality, the Verifier is deployed in the same secure environment as the Setup component, and then offers the same level of security.

We are concerned by this explanation for two technical reasons:

- Art. 2.9.1.1 of the Ordinance appendix confirms that the auditors and their technical aids must be considered untrustworthy for individual verifiability. However, since these technical aids are the only ones to check the ZK proofs produced by the control components, we see that individual verifiability completely depends on the auditors and their technical aids to be trustworthy.

- Art. 2.9.3.1 of the Ordinance appendix confirms that the auditors and their technical aids must be considered untrustworthy for privacy. We suspect that this is satisfied by the protocol, but do not see how it is reflected in the actual security definition and proofs in Section 18. We would expect the adversary to run the VerifyConfigPhase algorithm in the security game of Fig. 37, but it seems to be honestly executed by the challenger at Step 4. Nothing appears in the more detailed Fig. 38 either.

On a more conceptual level, the argument that the departure from the trust model required by the Ordinance is less problematic because the Verifier runs in a trusted environment seems to go against what is envisioned in the explanation of the Ordinance, which states on p. 12: "The use of auditors promotes transparency. Voters should be able to assume that auditors will draw attention to possible irregularities. The use of auditors in the sense of voter-representation [...]" then, on p. 15, "As long as the technical tool [the auditors] use to check the proof (typically a laptop computer) [. . .]".

We certainly support the idea of leaving the auditing work quite transparent, performed by people that are independent of the election organizers, and would be representative of the voters and candidates. Basing the verification process on a Verifier running in a controlled environment seems much less transparent than a process in which multiple auditors come with laptops.

But these difficulties also seem to come from a completely unnecessary nonconformity: would there be a problem with logically merging the current Setup and Verifier components, and to let the auditing work related to universal verifiability be performed by auditing devices that are external to the infrastructure? It would also probably make sense to require more verification work by the control components.

More generally, the instantiation of the auditors and their relation (if any) to the setup component should be better explained.

2.2.2 Roles of the Tally Control Component and Electoral Board

There is also an important discrepancy between the threat model given in Table 1 and Table 5 of the Proof document and the one used in the privacy definition (Def. 15) of the same document. (A very similar point was already raised in our March 2022 report. Many changes were made since then, but this difficulty isn't solved, nor addressed in the responses we received.)

Table 1 of the Proof document indicates that the Tally Control Component and the Electoral board are part of the "Untrustworthy System", and Table 5 of the same document confirms that the "Untrustworthy System" must be considered untrustworthy regarding voting secrecy, which is aligned with the Ordinance.

This is partially confirmed in the Specification document, which indicates that "the Swiss Post Voting system fulfills the desired security objectives *without* relying on the electoral board" – nothing similar is indicated regarding the Tally Control Component in that document.

The "E-Voting Architecture" document indicates that the four online control components (CCRs and CCMs) are controlled by Swiss Post. As such, Art. 2.9.3.3 of the Ordinance seems to apply, implying that "none of these control components is considered trustworthy" for privacy.

But this raises a difficulty: the Tally Control Component and the online control components jointly hold the secret keys that are sufficient to decrypt all the individual votes. But, if they must all be considered untrustworthy, then we do not see how the model described in Table 1 and Table 5 of the Proof document can guarantee the secrecy of the votes: in that corruption models, all the votes can be decrypted as soon as they are cast.

And, indeed, we observe that the formal definition of privacy (Def. 15 of the Proof document) considers a different security model: here, the adversary chooses one control component to behave honestly, within a set that includes the four online control components *and* the tally control component. So, the tally control component may actually be considered as trustworthy, in contradiction with the security model of Table 1 and Table 5.

This certainly makes sense from a security point of view. But now, the relationship with the security model of the ordinance becomes unclear. Section 18.2 indicates that the Tally Control Component, together with the four online control components, should be seen as a single control component group of five components, contrary to what is indicated in Table 1. But this would also appear to be a discrepancy with the Ordinance, which considers groups of 4 control components. This should be reflected much earlier in the document, around Tables 1 and 5, and conformity to the Ordinance should be discussed.

Considering the Tally Control Component as honest (and the online control components as dishonest) now raises the question of the trust that must be placed in the Electoral Board. Leaving it untrustworthy (as in Table 1) makes

little sense since the compromise of the Electoral Board means that the secret key used by the Tally Control Component would be compromised as well, resulting in a complete loss of privacy when the online control components are dishonest.

But what is then the trust model regarding the Electoral Board? And, if there is a need to consider that at least one of the Electoral Board Member should be honest when the Tally Control Component is selected as the honest one, how is it possible that nothing related to the entropy of the passwords selected by the board members appears in the security bounds of Theorem 5? Since SetupTallyEB derives $\mathtt{EB}_{\mathtt{pk}}$ from these passwords, the strength of these passwords seems crucial for privacy.

To conclude: the trust model regarding the Tally Control Component and the Electoral Board should be:

- made consistent throughout the documentation;
- mapped with the roles and trust model defined in the Ordinance.

Furthermore, depending on the trust model that is proposed regarding the electoral board, it may be necessary to adapt the security proofs accordingly.

2.3 Missing Elements in the Specification

2.3.1 Universal verifiability

The evaluation of universal verifiability at the moment is problematic because of missing information in the verifier specification:

- The pTable needs to generated or verified to ensure a consistent view with the information sent to the print office.
- The correct decoding of the primes into voting options needs to be verified.
- Moreover, all steps performed by the auditor with respect to the verifier UI needs to be checked. For example, verifying that the number of voter cards issued matched the number of eligible voters.

Those steps do not seem to be complicated to add, but their absence seems to break universal verifiability.

2.3.2 Handling Inconsistent Views of Confirmed Votes

Section 12 of the Proof document is novel, and addresses a non-conformity that was identified in a previous round of review: how to handle control components that have different views of the set of confirmed votes that need to be included in the tally?

The new protocol that is outlined in that section is plausible. However, the dispute resolution process that is outlined here appears to be completely missing from the Specification document, and does not seem to be reflected in

the security definitions and proofs of the Proof document. As such, the documents do not provide any input that we could use to assess whether this dispute resolution process, once fully specified, would enforce the security guarantees required by the Ordinance.

We elaborate on this question in the next subsection, since it was a central concern in our previous report.

2.4 The Vote Confirmation step

We focus here on the new Vote Confirmation protocol, including the handling of inconsistent views of confirmed votes. This includes Sections 12 and 16.5–8 of the Proof, Section 5.2 of the Specification and Section 4.3 of the Verifier.

We want to start by stressing that, at this stage, we do not see any feasible attack. However, there is still some confusion over the resolution of inconsistent views (among the control components) of which votes were confirmed. This confusion carries over to the proof of Theorem 3, for which the correctness condition does not seem to have been updated for the new resolution process.

We may be confused about the intended process, but we have done our best to explain which parts of the specification, verification specification and proof are confusing or apparently inconsistent.

2.4.1 The verifier specification - Section 4.3

The first bullet point requires that the set of confirmed encrypted votes should be the same for each Control Component, regardless of order. This is perfectly clear, but we do not understand how it relates to the resolution process described in Section 12.2 of the computational proof.

2.4.2 System specification

Typo: In the comment at the bottom of p.60 it means $\texttt{lVCC}_{\texttt{id},j}$ not $\texttt{hlVCC}_{\texttt{id},j}$. The CCRs send $\texttt{hlVCC}_{\texttt{id},j}$ to the VS before they can assess whether the confirmation attempt has been successful.

The system specification is unclear about what happens if different CCRs (initially) compute different lists of confirmed votes. Section 6.1.1 states, "The control components retrieve the mixnet's initial ciphertexts from the list of confirmed votes $\texttt{L}_{\texttt{confirmedVotes},j}$ that they established during the voting phase." However, it is not clear at this point what happens if their lists are different, that is if $\texttt{L}_{\texttt{confirmedVotes},j} \neq \texttt{L}_{\texttt{confirmedVotes},j'}$.

2.4.3 The computational proof - Section 12

This section suffers from a lack of clarity around exactly what list of votes gets included. There are 5 active participants: the VS and the four CCRs. They might (through malice or accident) produce different sets of confirmed votes $\texttt{L}_{\texttt{confirmedVotes},j}$

Some of us misunderstood the sentence "we do not foresee a recovery procedure for handling inconsistencies in the ConfirmVote protocol"—it seems that there is a recovery procedure (sketched in Section 12.2), but this is performed by some external party rather than the CCRs themselves. These questions are not resolved by the discussion in Section 12.2, which seems to assume that the honest control component's transcript is accepted, without explaining how to tell which control component is honest.

One interpretation is that a vote will be included into the final $\mathtt{L}_{\mathtt{confirmedVotes}}$ if *any* CCR can produce a complete set of four hashed long vote cast return code shares $\{\mathtt{hlVCC}_{id,j}\}_{j=1}^{4}$ that can be used to extract a share $\mathtt{lVCC}_{\mathtt{id},j}$ from the long Vote Cast Return Codes allow list $\mathtt{L}_{\mathtt{lVCC}}$. This mathematical property is well-defined, but Section 12 does not clarify who checks it or how they communicate the results to the other participants. When a (perhaps cheating) CCR is obliged to "provide evidence," to whom is that evidence provided?

A caution about assuming the honest CCR can be identified when resolving inconsistencies. Looking at the resolution process of 12.2.1 and 12.2, one might be tempted to think that the CCR whose $\mathtt{L}_{\mathtt{confirmedVotes}}$ is a superset of the others' must be honest. However, this is not necessarily the case. Consider the following behaviour, which does not violate any of the security goals of the system, but does allow a dishonest CCR to make it seem as if they have properly counted more votes than others.

Suppose a voter wishes to confirm their vote and sends the correct $\mathtt{BCK}_{\mathtt{id}}$. The VS and a CCR (wlog $\mathtt{CCR}_4$) are malicious. The honest client computes and uploads the correct $\mathtt{CK}_{\mathtt{id}}$. Dishonest $\mathtt{CCR}_4$ computes the wrong value of $\mathtt{hlVCC}_{\mathtt{id},4}$, which is forwarded to an honest CCR (wlog $\mathtt{CCR}_1$). Although honest, $\mathtt{CCR}_1$'s run of VerifyLVCCHash (Algorithm 5.10) fails, and it does not add $\mathtt{vc}_{\mathtt{id}}$ to $\mathtt{L}_{\mathtt{confirmedVotes},1}$. The VS is unable to compute the correct $\mathtt{VCC}_{\mathtt{id}}$, and this voter does not receive their Vote Cast Return Code.

Meanwhile, the dishonest CCR shares the correct $\mathtt{hlVCC}_{\mathtt{id},4}$ value with the dishonest VS. Note that VS now has the correct $\mathtt{hlVCC}_{\mathtt{id},j}$ for $j = 1, 2, 3, 4$, so they can gather enough evidence to prove that the vote should have been included, during the resolution process sketched in Section 12.2.

This works whether $\mathtt{CCR}_2$ and $\mathtt{CCR}_3$ are honest or malicious.

Nothing about this violates any of the security goals: the voter confirmed their vote, and their vote will be counted. However, it serves as a caution against laying blame in the case that one CCR seems to have omitted a vote that others can prove should have been counted. In this example, the honest CCR has been made to look as if it maliciously dropped a vote when it did not. The protocol should not be assumed to have *accountability*.

Our expected intuition for the security proof The following is our intuitive understanding of how we think the argument for protocol security would work. There are two crucial parts of the (intuitive) argument that a vote cannot be accepted without the voter having entered the correct ballot confirmation

code.

1. An honest CCR will not add $\texttt{vc}_{\texttt{id}}$ to $\texttt{L}_{\texttt{confirmedVotes}}$ unless it has received all four valid $\texttt{hlVCC}_{\texttt{id}}$ values.

2. An honest CCR will not compute a valid $\texttt{hlVCC}_{\texttt{id}}$ value without the correct $\texttt{CK}_{\texttt{id}}$.

It follows that at least one honest CCR must have received the correct $\texttt{CK}_{\texttt{id}}$.

There also needs to be an argument that it is hard for the (three) dishonest CCRs to guess (and verify) either $\texttt{CK}_{\texttt{id}}$ or $\texttt{VCC}_{\texttt{id}}$ without the collusion of the fourth CCR. (Note that it is straightforward if all four CCRs collude to enumerate all possible $\texttt{BCK}_{\texttt{id}}$ values until they find the correct one. This would, with high probability, require all of them to exceed the permitted number of guessing attempts.)

Now consider the argument that a vote cannot be rejected unanimously if a voter has received the correct $\texttt{VCC}_{\texttt{id}}$. This relies on the following steps.

1. The VS cannot compute $\texttt{VCC}_{\texttt{id}}$ without all four $\texttt{lVCC}_{\texttt{id},j}$ values.

2. An honest CCR will send $\texttt{lVCC}_{\texttt{id},j}$ only if it adds $\texttt{vc}_{\texttt{id}}$ to $\texttt{L}_{\texttt{confirmedVotes}}$ (Alg. 5.10).

3. An honest CCR will add $\texttt{vc}_{\texttt{id}}$ to $\texttt{L}_{\texttt{confirmedVotes}}$ if and only if it has a complete, correct list of $\texttt{hlVCC}_{\texttt{id}}$ values.

Since at least one CCR must be honest, it follows that it must have added $\texttt{vc}_{\texttt{id}}$ to its version of $\texttt{L}_{\texttt{confirmedVotes}}$ and it must have the evidence (the list of $\texttt{hlVCC}_{\texttt{id}}$ values) to prove that this was done correctly.

We now examine the security proofs with this intuition in mind.

2.4.4 Section 16.5 Vote rejection and the proof of Theorem 2

We cannot see an attack on vote rejection. However, the reasons that this is difficult are not well clarified by the proof. Although the games, theorem statements and proofs are much improved from previous versions, there are still some ambiguities and gaps.

Game $\texttt{rac}-\texttt{rej}.1$: The unimportance of exponentiation proof soundness In Line 13 of $\texttt{CreateLVCCShare}_j$, $\texttt{CCR}_j$ generates a proof of valid exponentiation for $\texttt{lVCC}_{\texttt{id},j}$, the value that is subsequently used by VS to create the Short Vote Cast Return code $\texttt{VCC}_{\texttt{id}}$. The exponentiation proof is never verified, by either VS or the auditors. Similarly, although $\texttt{hlVCC}_{\texttt{id},j}$ is supposed to be a hash of $\texttt{lVCC}_{\texttt{id},j}$, this is never checked either.

We do not believe that either of these checks are necessary, so this does not seem to lead to an attack. We think the protocol would be just as secure if the exponentiation proof were omitted.

Again, the proof does not seem to reflect accurately why the protocol is secure, because the proof of Theorems 2 and 3 (vote rejection and injection)

both include terms related to the soundness of the ZKP, though this proof is never verified. This affects the first game step in each theorem.

To put it more concretely, it would not matter if the adversary cheated on the exponentiation ZKP, because the honest $\texttt{CCR}$ would have added $\texttt{vc}_{\texttt{id}}$ to $\texttt{L}_{\texttt{confirmedVotes}}$ by then anyway, thus preventing vote rejection.

This does not make the theorem wrong—the extra term in the inequality seems to be added unnecessarily, but this does not make the theorem false—however, it does further support our observation that the proof is not closely connected with the reasons that the protocols are secure.

Game $\texttt{rac} - \texttt{rej}.3$: and the definition of $\texttt{lVCC}_{id,j}$ The proof of indistinguishability here includes the crucial statement: "Whenever the oracle $\mathcal{O}\texttt{HonestVerifyLVCCShares}$ returns $\texttt{lVCC}_{\texttt{id},j}$, it marks the ballot with $\texttt{vc}_{\texttt{id}}$ as confirmed." This is certainly a vital part of the argument, but it suffers from some ambiguity here about whether we are referring to things by their name or their value. Suppose the honest $\texttt{CCR}$ can be induced to return the correct value of $\texttt{lVCC}_{\texttt{id},j}$, but in the belief that it is retrieving the value for some other voter $\texttt{id}'$—how would that possibility be included in this part of the proof?

Again we do not think there is a real attack here, but there is a gap in the argument that the correct value could not be derived by tricking the honest $\texttt{CCR}$ in the context of confirming another voter's vote.

2.4.5 Section 16.7 Vote injection

We cannot see an attack on vote injection. However, the reasons that this is difficult are not well clarified by the proof.

The definition of $\texttt{bad}_{\texttt{inj}}$ does not seem to be the right one, given the process for resolving inconsistencies described in Section 12.2 of the Computational Proof. Although that is not (yet) very precisely defined, it seems that a vote will be included if any $\texttt{CCR}$ can produce a complete set of four hashed long vote cast return code shares $\{\texttt{hlVCC}_{id,j}\}_{j=1}^{4}$ that can be used to extract a share $\texttt{lVCC}_{\texttt{id},j}$ from the long Vote Cast Return Codes allow list $\texttt{L}_{\texttt{lVCC}}$. This $\texttt{CCR}$ need not necessarily be the honest one—indeed, no-one knows which one is honest when they are trying to resolve inconsistencies (see above).

So this proof needs to be modified to define the bad event more carefully based on the resolution process. This requires defining the resolution process more precisely and then using that definition in the proof that vote injection is not possible.

2.5 Notes on the Cryptographic Primitives, Definitions and Proofs

The cryptographic primitives used in the protocol are outlined in the Computational proof document [3], where security definitions are also given, and primitives are instantiated in the Cryptographic Primitives document [2].

We list some inconsistencies, suspected typos/errors, questions, and suggestions here.

2.5.1 Symmetric encryption

Section 4 of the Proof documents defines IND-CPA, IV-based encryption, then mentions that it will use an authenticated encryption (AE) scheme in reality, which is fine since an IV-based AE is also expected to be IND-CPA secure.

Section 5 of the Primitives document however defines something different: it defines an AEAD (so, an AE that also supports associated data that are authenticated but not encrypted), and calls it nonce-based.

- The change of terminology (nonce-based vs. IV-based) looks entirely cosmetic. Would it make sense to use an "IV-based" terminology everywhere? Since you are using the GCM-mode, which requires the use of an unpredictable value, and since the literature (e.g., the original McGrew and Viega paper and the NIST Special Publication 800-38D), our recommendation would be to call it an IV everywhere.

- The GenCredDat algorithm (Alg. 4.8 in the Specification) actually makes use of associated data. So, this creates a discrepancy between the encryption primitive defined in the Proof document and the one actually used in the protocol. Our suggestion would be to define an AEAD in the Proof document, and possibly to explain why the authenticity of the ciphertext and associated data actually does not matter in the security proofs.

2.5.2 Weak Pseudorandom Functions

Section 5 of the Proof document has a definition of weak pseudorandom function that we cannot parse.

- In the rPRF oracle definition, the adversary has access to an oracle that appears to always return $F_{pp}(k, x)$, which makes little sense for an oracle. We would rather expect the adversary to be able to query an oracle that would return $(x, F_{pp}(k, x))$ pairs with a freshly selected random value x on each oracle call.

- Similarly, in the sPRF oracle, we would expect the adversary to see $(x, R(x))$ pairs for freshly selected random values of x.

2.5.3 Proof Systems

The definitions provided in Section 6 of the proof document appear to be quite approximate.

- We cannot see how Definition 4 conveys the commonly accepted idea of special honest verifier zero-knowledge, which requires that a simulator, on receiving a statement and the verifier random coins as input, should

be able to produce a proof transcript that matches the distribution of a honest proof/argument conditioned on these coins. This appears clearly in Definition 3 of the Bayer-Groth paper that you are referring to, where the simulator receives as input the adversarially chosen verifier randomness ρ. The notion of SHVZK is also defined for public coin protocols, but we do not find any mention of this in Definition 4 of the Proof document.

- It is mentioned after Definition 5 that special soundness implies soundness in the sense of Definition 3. This only appears to be the case when the challenge space is super-polynomial, which may or may not be the case depending on the protocol that is considered.
- In Definition 6 of non-interactive zero-knowledge in EPROM, the $\mathcal{P}^{\mathsf{RO}}$ oracle appears to be undefined.
- In Definition 7 of simulation-soundness, at Step 4 of the sSOUND experiment, it is only checked that $((st^*, c^*, e^*) \notin \mathcal{T}$ while traditional definitions, including the one of Faust et al., will check that the full proof has not been already given to the adversary – hence including the response z. Given that π^* must pass verification, the two notions would be equivalent when the protocol has unique responses. This appears to be the case in the protocols that you are considering, but it is not mentioned or discussed.
- In Definition 8 of Weak simulation extractability, we suspect that d should be constant rather than polynomially bounded in the security parameter. We also observe that it is verified that $(st^*, \pi^*) \notin \mathcal{T}$, which appears to be inconsistent with the definition of $\mathcal{T}$ that does not contain any full proof π^*. Including full proofs in $\mathcal{T}$ would however address the issue pointed in Definition 7, and be consistent with common definitions.

2.5.4 Extractors

Section 13.2 of the Proof document makes the following claim:

> In some game hops, we require extractors: functions that extract witnesses from NIZK proofs or plaintexts from ciphertexts. Their existence allows us to state clear mathematical definitions and precise game hops. Extractors are used only by the Challenger in our games and never run in reality; therefore, they can be computationally unbounded.

This last argument seems to only offer one part of the picture. Indeed, challengers may never be run in reality, but we may still need them to run efficiently in security proofs.

For instance, looking at the proof of Lemma 2 (Vote Compliance), the reductions that are outlined there do not seem to work as explained. If we observe $\mathsf{vc}.2$, the ideas appears to build $\mathcal{B}_2$ as emulating an interaction between the $\mathsf{vc}.1$ challenger and the adversary in order to obtain the π_{EqEnc} proof that can be

used to break the simulation soundness property. However, that strategy does not seem to work because simulation soundness is only defined against a computationally bounded adversary, and the challenger that $\mathcal{B}_2$ needs to emulate does not run in polynomial time because it needs to run the Extract algorithm. The same question can be raised in other places where unbounded extractors are used.

2.5.5 Proof of Vote Privacy

There are several aspects that we cannot really follow in the vote privacy definition (Def. 15) and theorem (Thm. 5).

- As discussed above, we do not understand why, at Step 4 of the vote privacy experiment, the VerifyConfigPhase is not left for the adversary to run.
- As discussed above, we do not understand how the Electoral Board does not play a role in Theorem 5 since its member jointly hold the secret decryption key of the Tally Control Component, which may be the only decryption key that is hoped to be out of reach of the adversary (when $h = m + 1$).
- The bound in the statement of Theorem 5 seems to be incomplete: those associated to $\mathcal{B}_9$ and $\mathcal{B}_{10}$, which appear in the game priv.8, seem to be missing.
- We do not see how the soundness or extractability of the Schnorr proof does not play a role in this theorem (but only the fact that these proofs are ZK). If we assume that a Schnorr proof is just an empty string and always declared as valid, we certainly have a ZK protocol. But we then have the usual attack on privacy in which we may have the first control component as the only honest one, and the other control components picking their public keys in such a way that they cancel the key of this first component. There, the malicious control components become able to decrypt all the ballots as they come, before the tallying phase even starts.

2.5.6 Context Verification

Input vs. Context. The Specification makes a distinction between algorithm inputs and context. Such a distinction certainly makes sense, but also raises the question of whether some context element really are context that may not need any specific verification steps, or actual protocol inputs that need to be verified.

This question becomes quite visible when looking at the code of the system, which appears to support multiple elections as represented by an election event ID, which may each contain several verification set ids, simultaneously. This is poorly aligned with the algorithms as shown in the specification which list this information as context which is "invariant." In particular if the context depends on the input, then crucial checks on the context may be forgotten since

they are viewed as invariant. The attack on universal verifiability described in Section 3.3.4 can be seen as an instance of that kind of issue.

Election context in proofs. Related to the previous issue, we note that certain security properties are not captured (well) by the current documentation. For example, it is clear that the security requirements include that a voter should only be able to vote for the ballot box which they are registered; it's far from clear from the description and proof that this is enforced – the context appears to be given for granted.

In general, we think that context variables need more discussion throughout the documents.

2.5.7 Primality testing

Section 4.6 of the Primitives document specifies a fairly sophisticated primality test. The motivation that is provided for the use of this test is the potential challenges that could come from adversarially chosen prime candidates. However, this concern does not seem to apply here since this primality test will only be applied to large integers that must be selected in a way that is verifiably (pseudo-)random.

We are then confused as of why 6 pages of the document are needed to describe a test that could be replaced with something considerably simpler and less error-prone (e.g., a standard Miller-Rabin test, as proposed).

It is also suprising to see such a test detailed in full here: primality tests are available in numerous standard libraries. So why are you not relying on any such library, just as it is done for implementation of the standard cryptographic mechanisms like the AES? Is there anything in your test that differs from those available in standard libraries? This would help understanding the purpose of this long specification.

3 Scope 2: Software

Summary

With respect to the code, we have only had time to conduct an initial pass and a few small security checks; we have not begun to assess compliance with most of requirements in this scope. Due to these factors this report should be viewed as very interim. The verifier code—which is crucial for all security properties—has been out for less than a month at the timing of writing; this would not have been sufficient time under the best of circumstances. Moreover, there have been numerous difficulties which have slowed our progress including missing dependencies, failing unit tests, undocumented architecture decisions, and unvalidated inputs.

Our initial impression is that much of the code is in good condition with notable exceptions; we expect most of it is compatible with ordinance requirements but the missing pieces are crucial. Moreover, at present the code as a whole lacks maturity; this is especially true in terms of security and we have been able to find exploitable vulnerabilities which violate the key requirement of complete verifiability (Sec. 3.3.4); we expect that more similar vulnerabilities exist in the code at present. This risk is amplified by missing details in the specifications, as we detailed in our analysis of scope 1.

Positively, the vulnerabilities we have found so far—while severe—are easy to fix. From our perspective the existence of exploitable vulnerabilities at this point is neither surprising, worrisome, nor indicative of bad practices. Indeed, we expect to see such vulnerabilities being detected until the system achieves a greater degree of maturity. We expect that with about six months of work, at the current rate, a fair degree of certainty could be be achieved that the code is free of the application specific vulnerabilities we are currently finding; this estimate depends on further examination not finding vulnerabilities which are deeper than those currently discovered.

3.1 Scope

Material in scope In addition to those materials mentioned in scope 1 we looked at the following code packages.

- Crypto-Primitives Source Code, Version 0.15.2.3 [4] – (*Primitives Code*)
- Crypto-Primitives-Domain Source Code, Version 0.15.2.5 [4] – (*Primitives Domain Code*)
- Verifier Source Code, Version 1.0.0 [7] – (*Verifier Code*)
- E-voting Source Code, Version 0.15.3.0 [5] – (*System Code*)

Questions in scope Our analysis in scope 2 covers the parts of the audit concept detailed in Table 1.

	Requirements of the draft OEV
d) Assess the alignment between software development products	24.1.9, 24.1.11, 25.1.3, 25.2.8
e) Assess the implementation of the protocol	2.5, 2.6, 2.7,2.8,2.12, 3.17, 25.1.2
f) Assess the functionalities	3.13, 2.11, 4, 5.1, 8.10, 9, 10, 11.5, 11.6, 25.7

Table 1: Requirement of the draft OEV covered in scope

3.2 Methodology

Our analysis consisted principally of manual code review using an Integrated Development Environment (IDE). When we suspected we had found a problem, we followed up by writing custom unit tests to verify. Due to time constraints we have achieved very little beyond understanding how the code works. We hope to complete a fuller review in the future as we detail in future work (Sec. 3.5).

3.3 High level comments

In this section we will discuss the high level issues that have arisen in our examination of the code so far.

3.3.1 Lack of maturity

The code is in fairly good condition but has not yet reached the level of maturity we would expect for security critical systems. Various indicators of this are detailed below, the first three hindered the examination of the code whereas the final ones reflects possible or actual security problems.

Missing dependencies When the *Verifier Code* was released we were not able to build it initially because it depended on version 0.15.2.5 of the *Primitives Domain Code* whereas the most recent version of the *Primitives Domain Code* then available was 0.15.2.3. This was fixed by Post within 48 hours of us notifying them of the issue.

At the time of writing it remains the case that the *System Code* available is incompatible with the released *Verifier Code* as noted in `https://gitlab.com/swisspost-evoting/verifier/verifier/-/issues/2`; this has hindered us in testing possible vulnerabilities.

Undocumented architecture decisions Both the *System Code* and *Verifier Code* make use of slightly different variants of inversion of control using the framework Spring. These programming techniques increase modularity and extensibility at the cost of loss of comprehensibility of code flow for those unfamiliar with the framework. These design decisions may well be warranted but require better documentation so that those unfamiliar with the details of the framework can understand the alignment between the code and specifications. We are now fairly comfortable with the decisions ourselves based on discussions with Post but the points they raised in those meetings should be documented.

Failing unit tests Having the correct dependencies we then tried to build the code but where unsuccessful due to failing unit tests. We were able to trace these down to a bug in the code responsible for retrieving data from disk which was behaving differently for Post then it was for us. This bug will be fixed in a future release see `https://gitlab.com/swisspost-evoting/verifier/verifier/-/issues/3`.

Unvalidated inputs The concern in scope 1 (Sec. 2.5.6) about the separation of input and context in the pseudocode algorithms turns out to be far from theoretical. Looking at the code it seems that often the context is taken as input without validation. For example, looking at the control components confirm vote code it seems that the verification card set of the voter is taken as input without validation; these invalid inputs are eventually caught by the allow list but that errors of this kind are being caught late by unrelated security checks is concerning.

We will discuss a similar example in Sec. 3.3.4 which does not get caught.

3.3.2 Technical debt

Post has made extensive improvements compared with the sVote system they started with. There are points, however, were the system still has significant unnecessary complexity which makes auditing harder and creates possible vulnerabilities.

For example, conceptually there are three tiers of IDs in the system: election IDs, voting rights IDs, and voter IDs. However, the system implements at least five different kinds of IDs with several required to have a one-to-one correspondence with each other. Refactoring the system to use a simpler ID system would reduce the attack surface significantly.

3.3.3 Auditability

The design decisions in the code are fairly common in commercial software development; the decisions focus on modularity and allowing reuse. Better

justification is needed to explain why these decisions are appropriate for the verifier given the substantial cost to auditability, we give a few examples below.

In many cases the security logic is spread across the processor, service, algorithms, and underlying data structures. This leads both to a high level of duplication and to difficult seeing all the checks happen; in several cases we resorted to writing unit tests to check the attacks are detected rather than try and read the code.

As alluded to in 2.5.6, often elements of the "context" are received as input with no apparent validation. It increases the reviewer's difficulty in assessing the code to have check each instance to see if it breaks anything.

3.3.4 An example attack on universal verifiability

The universal verifiability of the system relies (largely) upon a chain of zero-knowledge proofs. These proofs demonstrate that the announced result, for a given ballot box, is the correct decryption (and permutation) of that ballot box (which is here used to refer to an agreed upon collection of ciphertexts).

The vulnerability underlying the attack below is as follows: the verifier is inconsistent in how it extracts data from the disk; this inconsistency breaks the chain of zero-knowledge proofs. Specifically, it sometimes extracts data based on the filename/location of the data and sometimes based on the content of the data. The inconsistent data could plausibly have been detected by the following parts of the verification specification to varying degrees but the implementation does not prevent the exploitation of the vulnerability:

- The authentication checks could plausibly have helped. However, they take the context data as input from the signer, in this case the adversary, rather than check based on their own view of the context.
- No consistency check is performed that each online control component has contributed exactly one shuffle payload per ballot box.

Attack: We will assume below that the auditor and control component 1 are honest but all other control components are dishonest.

The attack works by altering the order and names of the shuffle payloads going to the verifier. The result of this reordering is that the online control components' shuffles will verify as expected BUT when the verifier attempts to verify the tally control component's payload it will NOT do this with respect to the output of the CC4 as the verifier intends, but some other output. In the example we provided to Post, the verifier checks the tally control component shuffle of the ballot box 750a359fc3bd48aca4a1156666846267 with respect to the (decrypted and permuted) output of CC1 for the ballot box 99208915ab634a4293e36fcf4efadf54. This means that the validity of the tally proofs no longer link the results for 750a359fc3bd48aca4a1156666846267 to its own ballot box but to that of 99208915ab634a4293e36fcf4efadf54.

We initially thought the attack would not work because, for technical reasons, the mismatched input must be from control component 1, 2, or 3. In other

words, the dishonest tally control component's proof would be verified with the wrong input ciphertexts but would ultimately result in garbled group elements which would be detected by the VerifyProcessPlaintextsAlgorithm. We realised later that the adversary could choose the secret keys of the dishonest control components such that they cancel each other out and the decryption of the ballots using the secret keys of the first control component and election board would have the same result as decrypting using all keys. This means that this verification should then pass.

Impact The attack allows the adversary to change the election result without detection by the voter or system. The limitation on the attack is that the result the adversary claims, with respect to a given ballot box, must have a relationship to the ballots cast in a different ballot box. The impact of the attack depends significantly on the election parameters but it seems likely it could be exploited in practice.

Resolution Our understanding from talking with Post is that they will address this by using consistency checks to ensure the file names and contents have the expected correspondence. This approach would seem to work but we encourage any such requirement to be documented in the verifier specification or architecture document.

3.3.5 Other observations

- The setup component (re)learns the verification ids from the CCRs and checks consistency, this seems fine as long as at least one CCR is honest.
- VerifyPrimesMappingTableConsistency does not check the same Prime does not represent multiple options. This concrete issue subsumed by wider issue around the handling of the Primes Table discussed in scope 1.
- The completeness check in the tally phase of verification would not catch missing ballot boxes but this would be caught by the consistency check.

3.4 Detailed comment

In this section we will make comments on Ordinance requirements as applicable.

3.4.1 d) Assess the alignment between software development products

24.1.9, Traceability between functional specifications and security requirements is guaranteed to interface level.
This requirement is currently not satisfied for the reasons discussed in scope 1.

24.1.11, Traceability between the entire source code and the specifications of the security functions is ensured and their correspondence is evident.
This requirement is currently not satisfied for the reasons discussed in (Sec. 3.3.3).

25.1.3, A description must be provided of the link between the legal requirements and the cryptographic protocol, the specifications and the documentation of the architecture.
We have conducted no analysis with respect to this requirement to date.

25.2.8, The cryptographic protocol, specification, design and source code are aligned.
This requirement is currently not satisfied for the reasons discussed in (Sec. 3.3.3).

3.4.2 e) Assess the implementation of the protocol

2.5, Individual verifiability
We have made some small preliminary investigations into the system's compliance with this requirement; we have nothing to report so far.

2.6, Universal verifiability
We have made some small preliminary investigations into the system's compliance with this requirement; the system does not comply with this requirement for the reasons discussed in (Sec. 3.3.4).

2.7, Preserving voting secrecy and excluding premature partial results
We have made some small preliminary investigations into the system's compliance with this requirement; we have nothing to report so far.

2.8, Effective authentication
We have conducted no analysis with respect to this requirement to date.

2.12.1, Only one vote can be cast with the authentication credentials assigned to a voter.
We have conducted no analysis with respect to this requirement to date.

2.12.2, The person voting enters their vote on the user device
We have conducted no analysis with respect to this requirement to date.

2.12.3, The person voting can change the vote up to the point of confirming the decision to cast it and can check the vote against a summary.
We have conducted no analysis with respect to this requirement to date.

2.12.4, After the person voting has had the opportunity to check the vote against the summary, he or she confirms on the user device that he or she wants to cast the vote as entered
We have conducted no analysis with respect to this requirement to date.

2.12.5, The proofs of correct voting under Number 2.5 must be divided into at least two sequential items of proof. Any indication presented as an item of proof must make a genuine contribution to the soundness of the proof referred to in Number 2.5.
We have conducted no analysis with respect to this requirement to date.

2.12.6, The user device displays the first item of proof to the person voting after he or she has confirmed on the user device that he or she wants to cast the vote.
We have conducted no analysis with respect to this requirement to date.

2.12.7, The user device will not display the next item of proof to the voting person until the voting person has entered into the user device that the previous item of proof is correct.
We have conducted no analysis with respect to this requirement to date.

2.12.8, By confirming that the penultimate item of proof is correct, the voting person confirms his or her decision to cast the vote definitively.
We have conducted no analysis with respect to this requirement to date.

2.12.9, The group of control components registers the vote as having been cast in conformity with the system when it has received confirmation of the definitive decision to cast the vote.
We have conducted no analysis with respect to this requirement to date.

2.12.10, When the person voting has checked the last item of proof as being correct, the voting process is complete. The last item of proof should be made particularly easy to check, by limiting the check as far as possible to the correct display of a single code or other simple indication.
We have conducted no analysis with respect to this requirement to date.

2.12.11, If voting data are imported, a setup component or a print component must no longer be considered trustworthy from that point on.
We have conducted no analysis with respect to this requirement to date.

3.17 Trustworthy components may perform only the intended operations.
We have conducted no analysis with respect to this requirement to date.

25.1.2 All cryptographic protocol requirements across all work products associated with the software development process must be traceable.
This requirement is currently not satisfied for the reasons discussed in (Sec. 3.3.3).

3.4.3 f) Assess the functionalities

3.13, Data exchange or storage media, such as USB flash drives, must be removed after the data has been uploaded to the trustworthy components and may only be reused before the data is destroyed if there was no critical data on the trustworthy component before the data was uploaded. Data exchange or storage media must be reformatted and any data on them must be destroyed before they are used with the aid of a component operated in accordance with the requirements for trustworthy components.
We have conducted no analysis with respect to this requirement to date.

2.11.1 The probability of the attacker being able to falsify a proof under Number 2.5 if he changes a partial vote, suppresses a partial vote or casts a vote in someone else's name must not exceed 0.1%.
We have conducted no analysis with respect to this requirement to date.

2.11.2 The probability of the attacker being able to falsify a proof under Number 2.6 if he causes the calculated result to deviate by 0.1% from the correct result by altering and suppressing votes cast in conformity with the system or by entering votes not cast in conformity with the system may not exceed % per proposal, list or candidate selection.
We have conducted no analysis with respect to this requirement to date.

2.11.3 If the probability of the attacker being able to falsify a proof under Number 2.6 is not negligible in the cryptographic sense, it must be possible to reduce the probability of success as desired by repeated tallying, by providing the auditors with an additional, independent

proof under Number 2.6 for each count.
We have conducted no analysis with respect to this requirement to date.

4.1, The person voting must declare that he or she is aware of the rules on electronic voting and of his or her own responsibilities.
We have conducted no analysis with respect to this requirement to date.

4.2, Before casting a vote, the person voting is notified that he or she is taking part in a ballot in the same way as voting by post or voting in person at the ballot box. The person voting may only cast his or her vote after confirming that he or she has taken note of this.
We have conducted no analysis with respect to this requirement to date.

4.3, When voting, the person voting is requested to check the proofs in accordance with Number 2.5 against the verification reference and to report any doubts as to its correctness to the canton
We have conducted no analysis with respect to this requirement to date.

4.4, At any time before casting an electronic vote definitively, the voter may still choose to cast his or her vote via a conventional voting channel.
We have conducted no analysis with respect to this requirement to date.

4.5, The client-side system as it appears to the person voting does not influence the person voting in his or her decision on how to vote.
We have conducted no analysis with respect to this requirement to date.

4.6, The user guidance must not lead persons voting to cast hasty or ill-considered votes.
We have conducted no analysis with respect to this requirement to date.

4.7, The system does not offer the person voting any functionality allowing them to print out or store their vote
We have conducted no analysis with respect to this requirement to date.

4.8, The person voting is not shown any information after the voting process is completed about the content of the vote that has been encrypted and cast.
We have conducted no analysis with respect to this requirement to date.

4.9, A voter who is unable to cast a vote because third parties have cast a vote using his or her voting papers unlawfully may still be allowed by the canton to vote provided the canton declares the unlawfully cast vote null and void. Voting secrecy in accordance with

Number 2.7 must be preserved.
We have conducted no analysis with respect to this requirement to date.

4.10, Voters with disabilities may be provided with a simplified procedure for checking the proofs. Only in such a case are derogations from the requirements set out in Number 2.9.1 permitted.
We have conducted no analysis with respect to this requirement to date.

4.11, As long as the system has not registered confirmation of a definitive electronic vote, the voter may still choose to cast his or her vote via a conventional voting channel.
We have conducted no analysis with respect to this requirement to date.

4.12, The use of a means of authentication independent of electronic voting is permitted. Effects on the integrity of the verification of the right to vote and the preservation of voting secrecy must be examined in detail as part of the risk assessment.
We have conducted no analysis with respect to this requirement to date.

5.1, If the electoral register data is imported from a third-party system that is outside the canton's control, the data must be encrypted and signed. The signature must be verified on receipt of the data. For delivery to the printing office, the provisions of Number 7 take precedence.
We have conducted no analysis with respect to this requirement to date.

8.11, The information essential for secure voting is sent with the voting papers. Voters are told that if in doubt, they should comply with the information in the voting papers rather than the information displayed on the user device.
We have conducted no analysis with respect to this requirement to date.

9, The electronic voting channel is only available during the permitted period.
We have conducted no analysis with respect to this requirement to date.

10, Votes not cast in conformity with the system are not stored in the electronic ballot box.
We have conducted no analysis with respect to this requirement to date.

11.1, The decryption of the votes and the tallying may not begin before Polling Sunday
We have conducted no analysis with respect to this requirement to date.

11.5, If the result data is transmitted to a third-party system that is outside the canton's control, the data must be encrypted and signed.
We have conducted no analysis with respect to this requirement to date.

11.6, The system allows the polling card to be used to determine whether someone has cast an electronic vote.
We have conducted no analysis with respect to this requirement to date.

25.7.2, The software must be user-friendly. The user guidance is based on generally familiar schemes.
We have conducted no analysis with respect to this requirement to date.

25.7.3 The client-side system as it appears to the person voting complies with Accessibility Standard eCH-005913, with the exception of the requirements for alternative communication forms in Chapter 2.4 of the standard. The cantons shall ensure that compliance is confirmed by a specialist entity.
We have conducted no analysis with respect to this requirement to date.

3.5 Future work

Currently left as future work is a proper evaluation of the software with respect to the requirements detailed in Table 1. We intend to start with requirements 2.5 (Individual verifiability), 2.6 (Universal verifiability), 2.7 (Privacy); we envision that our addendum will provide good analysis of compliance with the above three requirements but we expect that we will not be able to examine most of the other requirements in this time frame.

References

[1] Swiss Federal Chancellery. Federal Chancellery Ordinance on Electronic Voting (OEV). https://www.fedlex.admin.ch/eli/cc/2022/336/en, July 2022.

[2] Swiss Post. Cryptographic Primitives of the Swiss Post Voting System – Pseudo-code Specification. https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives, June 2022.

[3] Swiss Post. Protocol of the Swiss Post Voting System – Computational Proof of Complete Verifiability and Privacy. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation, July 2022.

[4] Swiss Post. Swiss Post Voting System – Crypto-primitives Source Code. https://gitlab.com/swisspost-evoting/crypto-primitives, 2022.

[5] Swiss Post. Swiss Post Voting System – Source Code. https://gitlab.com/swisspost-evoting/e-voting/e-voting, 2022.

[6] Swiss Post. Swiss Post Voting System – System Specification. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation, June 2022.

[7] Swiss Post. Swiss Post Voting System – Verifier Source Code. https://gitlab.com/swisspost-evoting-int/verifier/verifier, 2022.

[8] Swiss Post. Swiss Post Voting System – Verifier specification. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation, August 2022.