Remarks on Selected Elements of the Swiss Post e-Voting System Versions 1.2.3, 1.3, and 1.3.1

Final Version

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

July 31, 2023

Contents

1 Summary

This report contains an update on a small subset of topics regarding the Swiss Post Voting System. These topics have generally seen improvement with the exception of the verifier which remains inadequately specified at the specification and operation levels.

We were asked by the Swiss Federal Chancellery to investigate a specific set of issues which we list below.

Version 1.2.3 of the Voting system:

- Primality testing, which we discuss in Section 2.1;
- Parameter generation p and q, which we discuss in Section 2.2;
- Algorithm RecursiveHashToZq, which we discuss in Section 2.3;
- Auditors' manual checks in verifier specification, which we discuss in Section 2.4.

Version 1.3 of the Voting system:

- Voter authentication, which we discuss in Section 3.1.

Version 1.3.1 of the Voting system:

- We comment on the changes based on our feedbacks on Versions 1.2.3 and 1.3 in a " – *Status* – " paragraph at the end of each subsection of Sections 2 and 3, and comment on other changes in Section 4.

Not version specific

- Implementation, with regard to voting secrecy and universal verifiability (in particular given that the setup component is not trusted), which we discuss in Sections 5.1 and 5.2.

There are also several recent changes to the system which relate to the concerns we have expressed in the past; except where listed above we have not examined these changes.

– *Status* – Many of the changes suggested in the primarily version of this report of May 26, 2023 have been implemented or scheduled for implementation. However, we remain concerned about the lack of clear operational procedures around the use of the verifier, see Sec. 2.4. In our judgment this is incompatible with Art.3.a and Art.3.c of the Federal Chancellery Ordinance of May 25, 2022, on Electronic Voting [9]. Post has scheduled improvements to this issue but not until 2024.

In the rest of this document, we refer to:

- the "Cryptographic Primitives of the Swiss Post Voting System" document as the *Primitives* document;

- the "Swiss Post Voting System – System specification" document as the *Specification* document;
- the "Protocol of the Swiss Post Voting System – Computational Proof of Complete Verifiability and Privacy" document as the *Proof* document;
- the "Swiss Post Voting System – Verifier specification" document as the *Verifier* document.

2 Remarks based on version 1.2.3

2.1 Primality testing

We agree that the solution to check primality proposed in Section 7.1 of the Primitives document offers the expected guarantees. We do not understand why the process starts with the Baillie-PSW test, given that a "standard" Miller-Rabin test is executed next. The Miller-Rabin test offers the guarantees that are needed, and so the Baillie-PSW test just seems redundant. Our view is that the additional Baillie-PSW test makes the prime number testing/generation process more complicated and slower than necessary, even though it does not create any security problem.

– *Status* – Post has agreed that using only the Miller-Rabin test is more consistent and they intend to implement this in release 1.4.

2.2 Parameter generation

The parameter generation algorithm described in Section 7.2 of the Primitives document looks reasonable. We make two improvement suggestions below, which would increase our confidence in the process. We do not feel that they are critical, but they would make the process more consistent with the rest of the specification, or with practices that are more standard and more directly studied in the literature (which is also a source of confidence).

2.2.1 SHAKE version

We wonder why SHAKE128 is used as the hash function rather SHAKE256 as it offers a lower security level than what is used in the RecursiveHashToZq function for instance. Is there any reason to not stick to the same version of SHAKE everywhere?

– *Status* – Post has agreed that using SHAKE256 is more consistent and they intend to implement this in release 1.4.

2.2.2 Non-uniform selection of primes

The new approach proposed to pick candidate values of q deviates from standard practices (e.g., as specified in FIPS 186-5) by iterating on candidate values of q using the jump variable until a suitable value is found, rather than selecting

uniformly random values of q at every step – this is what FIPS 186-5 does, and what was done in Version 1.1 of the Primitives document.

This new approach causes some (p, q) pairs to be selected with a higher probability than others. This is mentioned in the document, and it is a good thing to make it visible. We made a short simulation in order to have an idea of the differences of probabilities, and found that, by just selecting 10 pairs of 2047-2048 bit primes, some values of q will be selected with a probability 80 times higher than others – that is, on only 10 attempts, we found that the difference between two consecutive suitable values of q could vary by a factor of 80. The new algorithm will be 80 times more likely to pick a start random value in an interval that is 80 times larger than another.

On the one hand, we agree that this is most unlikely to lead to attacks for the proposed security parameters. (We conjecture that it may even be possible to prove that this does not create any security issue, though we did not try to write the proof.)

On the other hand, in the absence of proof, we are wondering whether the extra speed gained by this iterative strategy justifies the deviation from standard practices, and the deviation from the natural idea of "picking primes uniformly within a chosen range". After all, the parameter generation does not require more than a few seconds, or possibly single digit minutes for larger security parameters, on a standard laptop, and it is something that needs to be done at most once per election, so it will hardly be a bottleneck.

Independently of this, we see that the idea of picking $q = 1 \bmod 6$ rather than $q = 1 \bmod 2$ as in Version 1.1 is a clear improvement.

– *Status* – We understand that this issue will be revisited by 2024 in conjunction with measure A.25.

2.3 RecursiveHashToZq

The new RecursiveHashToZq algorithm follows a general approach to obtaining an output in $\mathbb{Z}_q$ that is efficient, and that will provide outputs that will be very hard to distinguish from uniform in $\mathbb{Z}_q$ (this approach was suggested in our August 2021 preliminary report [3, Appendix B.3]).

We are nevertheless slightly concerned by an implementation detail. We do not see how to exploit it in order to break the system, but it creates a situation that may be the source of security issues, and we do not see any mention of this potential problem in the documentation (nor, obviously, any explanation of why it cannot be exploited either). As it is easy to solve, we would advise to solve it rather than spending a non-negligible effort arguing why it may not be a problem.

The concern is that the RecursiveHashToZq algorithm may produce outputs that are related by a simple relation if it is used with the same inputs but different values of q, given that q is not part of the inputs of the underlying hash function - it only explicitly appears in the final modular reduction.

This is not just a theoretical concern, because the RecursiveHashToZq algorithm is indeed used with different values of q in the system. For instance, the q

value is set to $(p-3)/2$ in Algorithm 4.11, and is set to $(p-1)/2$ in Algorithm 8.6. It would then be important to check that, whenever this occurs, RecursiveHashToZq cannot by used with the same inputs for these different values of q. As the system evolves, this may become a tedious requirement.

We observe that the q parameter is already given explicitly as an input of RecursiveHashToZq in Algorithm 8.6. However, it is not an input when RecursiveHashToZq is used in Algorithm 4.11. This discrepancy offers a good reason for pushing the q value into the inputs of the hash function of RecursiveHashToZq: it will make things more consistent than they are now.

So, our suggestion would be:

1. Add the q value as a prefix/suffix/... with proper separation to the hashed values in the definition of RecursiveHashToZq, in order to prevent these related outputs. (As as side effect, this means that q can safely be removed from the inputs of RecursiveHashToZq in Algorithm 8.6.)

2. If this cannot be done for some reason, change: RecursiveHashToZq$(q-1, x)$ into RecursiveHashToZq$(q-1, (q-1, x))$ in Algorithm 4.11.

As a side suggestion, we would advise to add a usage context string in the inputs of RecursiveHashToZq in Algorithm 4.11, as it is done in Algorithm 8.6 with the "commitmentKey" string that is added to the inputs of the hash function. Domain separation is good.

– *Status* – Post agreed with our first suggestion and implemented it in version 1.3.1; we are happy that the domain separation and usage context string look appropriate.

However, there seems to be a notation issue in the updated Specification, though (the implementation looks correct). The Specification indicates that $q||$"RecursiveHash"$||\mathbf{v}$ is hashed, using a notation that is used for the concatenation of (byte) strings in the document. However, q is not a string, and $\mathbf{v}$ may not be one either. Mere concatenation may also create ambiguity, which is turn may create security issues. The implementation, on the other hand, treats this as a tuple, and hashes accordingly. We think that this is correct, and would suggest using a tuple notation in the Specification. We understand this will be revisited by 2024 in conjunction with measure A.25.

2.4 Auditors' manual checks in verifier specification

We do not have much to say about the manual checks performed by the auditors: they are still informal and we cannot be sure of what the auditors are supposed to check from what we read in the Specification and Verifier.

We think that is is crucial to amend the documentation with a detailed description of the use of the verifier by the auditors at the operational level. The verifier's sufficiency for universal verifiability is quite sensitive to how it is used. The voting system, like any engineered system, exhibits a tolerance between the specification, implementation, and operation; in other words, while the implementation and operation are largely the logical consequence of the specification

there is a degree of imprecision, as seen in the code and the documented procedures. The level of variation in the voting system is such that without knowing ahead of time how the verifier will be operated there is an unacceptable risk that it will not ensure the universal verifiability of the system.

– *Status* – Post has indicated they intend to address this next year (2024). Until this is addressed we believe that the systems should considered non-compliant with Art.3.a and Art.3.c of the OEV.

Specifically, it is not the case that "the system and the operational procedures are designed and documented so that the details of the technical and organisational procedures can be checked" " to guarantee verifiable, secure and trustworthy electronic voting."

3 Remarks based on version 1.3

3.1 Voter Authentication

Based on previous comments, Swiss Post has redesigned and documented the authentication mechanisms between the voter and the voting server. The new process is defined in Section 5.1 of the Specification document, with supporting algorithms in the Primitives document, and a security discussion in Section 11.2 of the Proof document.

We found analysing the security of this process challenging because it was unclear what security goals it met in the wider system – we assume that it is there in order to support conformity to the requirement of effective authentication of the OEV Ordinance [9, Sec. 2.8], as well as the requirement of protection against systematic compromise of the voter authentication credentials [9, Sec. 23.5].

We did not identify any critical weakness within the security model that we assume that the designers have in mind. However, that security model surprised us: it appears to assume that the voting server is trustworthy (the voting server is the one who verifies the authentication, so the authentication protocol does not seem to make sense if the voting server cannot be trusted), even though the Proof document makes it clear that the voting server is categorized as an untrustworthy component. This is however consistent with the discussion in Section 11.2 of the Proof document, which essentially explains that the authentication protocol does not bring any extra security to the system and does not degrade it either.

– *Status* –

- Post has made clearer what the extended authentication factor will likely look like in Section 4.1.4 of the Specification (Version 1.3.1). See Section 3.1.1 for our original discussion.
- Post considers implementing, in Version 1.4, our suggestion of tieing the authentication to the message sent. See Section 3.1.2 for our original discussion.

- Post has corrected the mistake in Argon2id parameters in the Primitives (Version 1.3.1). See Section 3.1.2 for our original discussion.
- Post did not directly comment on how they intend to conform with [9, Sec. 23.5]. See Section 3.1.1 for our original discussion.
- The possible issues arising from the use of a time-stamp, including the requirement of a synchronised global clock, have not been addressed. See Section 3.1.1 for our original discussion. Post has announced they intend to add an error message that highlights cases of unsynchronized clocks for the version at the end of this month.We suggest an alternative below.
- Voter authentication will be revisited by 2024 in conjunction with a follow-up on the existing measure A.16.

3.1.1 On the security model

It is our understanding that this voter authentication protocol, by adding an extended authentication factor $\texttt{EA}_{\texttt{id}}$, is aimed to be an important element towards achieving conformity with [9, Sec. 23.5], namely:

> It must be ensured that none of the elements of the client-sided authentication credentials can be systematically intercepted, changed or redirected during transmission. For authentication, measures and technologies must be used that sufficiently minimise the risk of systematic abuse by third parties.

Of course, the security model in the ordinance asserts that the communication channel between the print component and the voter may be considered trustworthy, which would imply that voting cards are always delivered to the expected voter without ever being intercepted, changed or redirected during transmission [9, Sec. 2.10.2] We may question this in practice, within the spirit of [9, Sec. 23.5], considering different security models. First, let us consider an actor who tries to collect a set of voting cards of a size that would be comparable to the typical result margin in an election.

- In the context of a local election, he may try to corrupt a postman, or another employee of the postal services with access to the voting card distribution circuit, in order to access a set of voting cards. He may also walk in the streets, following postmen delivering voting cards in mailboxes, and steal some of these cards from the mailboxes. In these two cases, some voters may complain that they did not receive their voting card. A more challenging, but more stealthy strategy, would be to search through paper trash during the days following the voting card delivery, looking for unopened voting card envelopes: taking advantage of these voting cards in order to impersonate voters seems unlikely to trigger a complaint, given that the voters did not even care to open the envelope.[1]

[1] We thank Aleks Essex for raising this scenario.

- If a significant part of the voters are voting from abroad, the postal services of some unfriendly countries may be coerced into intercepting a significant set of voting cards, and then vote on behalf of the corresponding voters. An alternative would be to simply take a copy of all the voting cards they see. Then, the local ISPs may be ordered to monitor the Internet activity of the potential voters, in order to see if they are trying to cast a vote. If they do not, then the copied voting cards may be used to cast a ballot on behalf of the corresponding voters.

All these scenarios violate the assumption that the channel from the print component to the voter is trustworthy. However, they offer the potential of systematic abuse in the real world.

A natural way of preventing this systematic abuse is to have a strong voter authentication mechanism, and our guess is that the purpose of the extended authentication factor used in the voter authentication protocol is to prevent exactly these kinds of scenarios: in the absence of the extended authentication factor, access to the voting card (and the the Start Voting Key in particular) is sufficient to cast a ballot. However, as long as the extended authentication factor remains safe, intercepting voting cards may, at worst, prevent a voter from using e-voting.

We may then consider two variations on this scenario, depending on whether the malicious actor collecting voting cards colludes with the (untrustworthy) voting server or not.

If the malicious voting card collector does not collude with the voting server, then the proposed voter authentication protocol would offer a layer of protection, under the assumption that the extended authentication factor remains safe.

The story is different if the malicious voting card collector colludes with the voting server and, here, we are touching a specific aspect of the current authentication protocol design: in the Specification document, the extended authentication factor is only used by the voting server, and not by any control component. Furthermore, the verification card keystores are also computed independently of the extended authentication factor. This means that, if the voting server colludes with someone who stole voting cards, then the voter authentication can be completely bypassed (without compromising the extended authentication factor), the Verification Card Secret Key can be decrypted, and votes can be cast on behalf of the voters who got their card stolen.

We would find it attractive to modify the protocol in such a way that, even if the voting server is compromised, it remains infeasible to cast a vote without knowledge of the extended authentication factor: this would definitely support compliance with [9, Sec. 23.5]. One step in that direction might be to require the setup component to encrypt Verification Card Secret Key $\mathtt{k_{id}}$ with a key that is also derived from the extended authentication factor in the GenCredDat algorithm (Algo. 4.9 in the Specification).

It is also worth clarifying the secrecy assumptions for the extended authentication factor. For example, if this information will be held by the Cantons (which we assume will be the case in practice) then it is important that (at

least the same part of) the Cantonal administration does not also see the voting cards, or the benefit of the extended authentication factor is lost against that attacker.

The Specification document is also quite vague regarding the implementation of the extended authentication factor. It is suggested, in Section 4.1.4, that it may be the voter's birth date, and that seems to be the case in practice. Of course, this is a fairly weak authentication factor in practice: birth dates are not particularly secret. They may be relatively easy to find, possibly through social medias, in the context of local elections. And, if the adversary is a foreign government agency of the country in which the voter lives, then it is most likely that the birth date has been given through visa requirements, or registration with the local municipality. At best, the birth date will slow down the adversary a bit. Besides, if the voting server is compromised and has access to the Start Voting Key $\texttt{SVK}_{\texttt{id}}$, then brute force may be easy to perform even if Verification Card Secret Key $\texttt{k}_{\texttt{id}}$ is only protected by a combination of $\texttt{SVK}_{\texttt{id}}$ and the birth date. A more sophisticated solution would prevent such a brute force attack by having the control components involved in the voter authentication protocol as well, and relying on the honest control component to limit the number or the rate of authentication attempts.

Overall, the voter authentication process appears to remain weak, and may be one of the main practical weaknesses of the voting system in general. However, to the best of our understanding, this is not related to any specific issue that could be overcome within the proposed voting system, but rather to the lack of availability of a remote authentication infrastructure for Swiss voters.

3.1.2 On the Voter Authentication protocol design

The authentication protocol is suprisingly complex compared to what it seems to be trying to achieve. It modifies and uses the TOTP protocol in ways that we cannot really explain. We list our main concerns below.

1. The authentication protocol is executed at each step of the voting process, and not just once per session, and we cannot see why – at least in its current form. Typically, the TOTP protocol is played once in a TLS session, in order to authenticate a user (the voter in our case), and it is not tied to any message content. Here the authentication protocol is repeated at every step of the voting process. In what scenario would it be possible that the protocol succeeds at the beginning of a voting session and fails at a later point? A simpler alternative would be to take advantage of TLS, authenticate the user once in a standard way, and then rely on the TLS session and its message authentication. Alternatively, if TLS is not trusted (for example, because of the use of proxies in front of the server), then the messages themselves need a Message Authentication Code, which would be easy to derive from the shared secret. But this is not done currently, since the various voter authentication sessions are not tied to any message content. Our suggestion would be to either play the authentication

protocol only once per voting session, or to tie each session of the voter authentication protocol to the associated voting protocol messages.

2. Section 3.5 of the Specification: why should we have both a unique voter identifier and a base authentication challenge? Is it because `CredentialID` is used to limit the number of guesses of `hAuthID`? This is the only reason we can think of, but it is not stated anywhere. It would be helpful to either explain why these two values are needed, or to remove one of them otherwise.

3. Building on the previous remark, we are confused by the role of $\texttt{salt}_{\texttt{id}}$: we do not see the benefit of having a salt that is not independent of the keying material, because this salt does not add any extra entropy. What are the benefits of the current approach compared to computing:

$$\texttt{bhhAuth}_{\texttt{id}} \leftarrow \mathsf{GetArgon2id}(\texttt{authStep} \| \texttt{hAuthID}_{\texttt{id}} \| \texttt{T}, \text{""})$$

(we removed the type conversion steps for readability). This saves the computation of $\texttt{salt}_{\texttt{id}}$, and $\texttt{bhhAuth}_{\texttt{id}}$ keeps being dependent of `ee` and $\texttt{SVK}_{\texttt{id}}$ through $\texttt{hAuthID}_{\texttt{id}}$.

– *Status* – Post commented that: "We do not think that setting a blank salt would be good practice." and modified Algorithm 5.1 by adding a fresh 256 bit nonce into each computed salt. The goal of this nonce is claimed to be in support of multiple authentication attempts during a single time step. As this nonce is selected by the voting client, we do not think that it can offer the extra independent randomness that would be needed to offer an effective salt – and Post makes no such claim either.

We would favor two options here:

(a) **Empty salt.** Use an empty salt and make it clear in the Specification that domain separation is guaranteed, and must remain guaranteed by the input keying material of Argon. The salt that is currently used does not offer the features that are needed from a salt (i.e., entropy that is independent of the keying material), so using no salt clarifies the protocol design. This choice is not discussed in the Argon design, but the Argon design focuses on the use of salt for password hashing, which is not the use case here. The HKDF paper [5] offers more discussions regarding the role of the salt and explicitly specifies the choice of using a null or constant salt when no independent source of randomness is available. We believe that it is a sound choice here too, as long as there is no independent random source that can be used as a salt.

(b) **Server-chosen nonce as salt.** The random nonce proposed in the latest version of Algorithm 5.1 could be used as a salt, but it should be chosen by the voting server and not by the voting client. This would offer more robustness to the protocol, but may marginally

increase the communication costs in some cases. More importantly, it would simplify the protocol by offering the opportunity to remove the requirement of synchronized clocks, which is a potential source of failures in practice. Indeed, the Unix time that is used in the protocol in order to prevent message replays could be replaced by the use of the nonce in a challenge-response mode, that would have the same effect. Of course, the server would have to manage the life time of the nonces, but this is already needed in order to prevent nonce replays in the current version.

4. The substitution of HMAC (in the TOTP protocol) with Argon2id (here) raises efficiency concerns that do not seem necessary: Argon2id is designed to be slow and memory demanding, while HMAC is fast, memory efficient, and broadly available.

   While investigating this question, we noticed a discrepancy between the Specification and the implementation: the Specification indicates that Argon2id is used with the LOW_MEMORY profile, which corresponds to the use of 16GiB according to Sec. 4.5 of the Primitives document. Such a requirement would be completely unrealistic in a browser, and a very risky constraint on the server side. We however realized that the implementation is much more prudent, and uses a variant of Argon2id that only requires 64MiB. We assume that the headers of the table Sec. 4.5 of the Primitives document are incorrect.

   Nevertheless, we do not understand why the voting process needs to be slowed down, both on the client side and on the server side, by multiple evaluations of Argon2id, while the TOTP specification uses HMAC: this will be much faster and less memory demanding. The voting system already uses HMAC as part of HKDF, and HMAC is also widely supported in browsers.

   Our suggestion here is then to follow the TOTP standard more closely by replacing the computation of $\texttt{bhhAuth}_{\texttt{id}}$ with something like:

   $$\texttt{bhhAuth}_{\texttt{id}} \leftarrow \mathsf{HMAC}(\texttt{hAuthID}_{\texttt{id}}, \texttt{authStep} \| T)$$

   This would in particular make it possible to perform the VerifyAuthenticationChallenge function without evaluating the Argon2id function.

   – *Status* – Post indicated that they prefer to keep using Argon2id that is already available in their TypeScript implementation, and that using the HMAC implementation that is available in browsers would require adapting their code, which they did not wish to do.

   We do not think that this choice raises security concerns, but we think that it may be an obvious place for speed improvements, should the voting client or the voting server be slow on some platforms. And, again, switching to HMAC would also make the protocol closer to well-studied standards.

5. The description on page 57 of the system specification is confusing.

   > "This ensures the freshness of the voting client's request and proves knowledge of the voter's Start Voting Key $\texttt{SVK}_{\texttt{id}}$ and extended authentication factor $\texttt{EA}_{\texttt{id}}$." and "For the shared secret, we use the Start Voting Key $\texttt{SVK}_{\texttt{id}}$ and extended authentication factor $\texttt{EA}_{\texttt{id}}$"

   This seems to imply that the Start Voting Key is a shared secret between the voter and the voting server as required in the TOTP protocol.[2] However, it is crucial to the security of the system that the (untrusted) voting server not know SVK. In fact, the real shared secret is derived from the SVK and provided to the voting server by the setup component. This is not a real problem in the specification, just a confusing use of language in this description.

   – *Status* – This has been rephrased.

3.2 Individual Verifiability

In the second addendum to our previous report [4], Section 2.4, we discussed some remaining issues with the specification of Vote Confirmation Agreement. These affect the proofs of individual verifiability, particularly Theorems 2 and 3, which state that (under certain assumptions) a vote that has been confirmed cannot be excluded, and a vote that has not been confirmed cannot be included. SwissPosts's April 2023 update does not contain new material for these sections, so we will wait on further updates before doing more analysis.

– *Status* – This question remains open. However, it is covered in the existing measure A.24 and so should be addressed by 2024.

4 Remarks based on version 1.3.1

We examined various other changes in version 1.3.1. We did not find any issues and include the examined points below for reference.

Changes to the storage of CMTable and pCC allow list:
The important safeguards appear unaffected by the changes; we did not understand how the EntityManager worked.

Optimized the exponentiation proof generation and verification by adding parallelization:
This looked fine.

[2] `https://datatracker.ietf.org/doc/html/rfc6238`

Optimized the GenExponentiationProof algorithm by omitting the costly input check on the input elements:
The GenExponentiationProof algorithm no longer checks that the statement it is constructing a proof for is true; we could see no issue with removing this check.

Various merges and chunk-wise execution:
We could see no issue with the changes.

Optimized the performance of consistency verifications by using a specialized library that avoid deserializing the entire payload:
We assume this is done in the new data extraction service but we were unclear on the details.

5 Scope 2: Code analysis

This section summaries our understanding[3] of the significant outstanding issues within Scope 2, as well as information about the depth of our knowledge; we discuss these in the context of privacy (Sec. 5.1), consensus and universal verifiability (Sec. 5.2), which we have summarised into the corresponding sections.

5.1 Vote Secrecy

The following seven questions which relate to the vote secrecy of the voting system were considered. Specifically:

1. Is election secret key EL_{sk} secret and uniformly distributed?
2. Is the user secret key k_{id} secret and uniformly distributed?
3. Where does the choice code allow list `LpCC` come from? Does checking for `pCC` inside it leak any details?
4. Where does the code table `CMtable` come from? Does checking inside it leak any details?
5. Did the voter know what they were voting?
6. Do the decryptions occur only when they are supposed to?
7. Can you decide the vote from the return codes or the voter's response to the return codes?

These questions have largely satisfactory answers. However, in examining the last question an interesting discovery was made. A way was found for the adversary to learn all the return codes but crucially not the relation between the codes and the voting options. We quote the relevant text below.

[3]The information presented is largely based on our own investigations but also draws upon the work completed by the undergraduate students at the Australian National University Yangda Bei, Lekh Bhatia, and Julian Crosby.

With some degree of simplification to recover the return code they *(the adversary)* would need pC_{id} which was decrypted from $c_{pc,id}$ in the `GenCMTable` algrothim. $c_{pc,id}$ is the product of $c_{expPPC,j,id}$ which are the result of exponentiation of the encryption $\tilde{p}_k^{k_{id}}$ by four $k_{j,id}$ known to the four control components. In short the adversary would need

$$\tilde{p}_k^{k_{id} \sum k_{j,id}}$$

where $\tilde{p}_k$ is the prime encoding of the voting option, k_{id} is the voter's secret key, and $k_{j,id}$ is the jth's control component secret specific to voter id.

When all four of the control components are dishonest, as allowed by the privacy threat model, it is simple for them to choose $\sum k_{j,id} = 0$ so that the value they need to find is the group identity. However, once this element is the group identity there is no value in the GenCMTable which ties the recovered code back to a particular voting option.

The above observation relates to earlier discussion among the commissioned experts about whether zero was a potentially dangerous special case for secret values. This case demonstrates that it can be because it effectively removes k_{id}, which is unknown to the adversary in this threat model, from contributing to the shared secret. We note that checking the $k_{j,id}$s individually, through their public keys, would not suffice since the sum could still be zero. In our view the best solution is to combine secrets additively not multiplicatively wherever possible but it is not clear how to do that in this case.

– *Status* – We received no feedback from Post on this point; we do not believe the odd behavior detailed above needs to be addressed. We expect this particular point to become moot with measure A.13 if not before.

5.2 Consensus Issues and Universal Verifiability

The Federal Chancellery's Ordinance on Electronic Voting (OEV)'s threat model for universal verifiability clearly envisions that the impairment causable by a malicious `setup component` should be mitigated; exactly delineating what unusual states the `setup component` can place the other components into, without detection, and the security implications of these states is an enormous amount of work which would need to be kept updated with each release; we strongly encourage strengthening consensus, doing so would effectively reduce the attack surface which needs to be examined thus making the security of the system more understandable.

The current specification and implementation of the Swiss Post e-Voting system misses mitigating the impact of a dishonest `setup component` on numerous occasions.

We detail several issues below (Sec. 5.2.3) but one stands out. At present the `verifier` does not check that the selected decoded voting options it receives are valid. Adding a check to ensure this should be matter of priority.

5.2.1 Security Requirements

The exact requirements envisioned by the OEV for universal verifiability remain vague to us; future versions of the legislation and/or explanatory reports may wish to define this property more precisely.

When considering how to define verifiability the Systemisation of Knowledge paper by Cortier *et al.* [2] is a useful resource. For example, they note:

> Kusters *et al.* [7] have proven that, in general, individual and universal verifiability (even assuming that only eligible voters vote) do not imply end-to-end verifiability.

Putting aside the issue with separating individual and universal verifiability. They summarise three definitions of universal verifiability which exist in the literature [1, 6, 8]. All of these definitions assume a clear correspondence, independent of any trust assumption, between the meaning of a ballot (an encrypted vote) and a choice (a plaintext vote); realising this in a deployed system is not directly feasible since doing so depends, at a minimum, on shared global parameters. However, the fact that all the formal definitions in the literature rely on this clear correspondence emphasises its importance.

We, therefore, believe the following subproperty is crucial to demonstrating compliance with any reasonable interpretation of the requirement.

- The system must include sufficient verification checks to ensure a consistent view of the votes between the `control components` and `verifier`.

A consistent view of the votes means both the group elements making up the ciphertexts and all information required to give those a definitive meaning in the language of the result. Without a consistent view between the `control components` and the `verifier` the language of "the votes" used in the OEV is simply undefined.

We further believe this is necessary because of how individual verifiability and universal verifiability are defined:

> 2 The requirements for individual verifiability are as follows:
>
> a. The person voting is given the opportunity to ascertain whether the vote as entered on the user device has been manipulated or intercepted on the user device or during transmission; to this end, the person voting receives proof that the *trustworthy* part of the system (Art. 8) has registered the vote *as it was entered by the person voting* on the user device as being in conformity with the system; proof of correct registration is provided for each partial vote.

We note that the only part of system which is trustworthy and stores the vote is the honest `control component`; this places the honest `control component` as the ultimate source for the meaning of this vote at the end of the voting phase.

3 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.

We emphasise that universal verifiability requires that the `verifier` receives proof that the result corresponds to the votes in the honest `control component`. If the `verifier` and `control component` have an undetected inconsistent view on any pertinent information relating to the meaning of the votes then the intended flow of verifiability is not guaranteed.

Looking at the specification and the source code, it seems that ensuring a consistent view between the `voter`s and the `control component`s have been a priority but consistency checks between the `control component`s and the `verifier` have been largely ad-hoc; the consistency checks in the `verifier`, which exist in the code but are largely nonexistent in the verifier specification, are of use here but leave room for improvement.

5.2.2 Recommendations

We recommend a careful documentation of all information on which the `control component`s and `verifier` need to agree, along with what checks ensure consistency of this data when the `setup component` is dishonest. We briefly include some examples below but these are not exhaustive:

- The encrypted votes (group elements)
- The election public key
- The mapping between decrypted votes (group elements) and plain language voting options
- Who the eligible voters (verification cards) are
- What verification card set and ballot box the eligible voters belong to

One approach would be to document all information the `control component`s and `verifier` receive and ensure consistency across all the data.

5.2.3 Details of problems

(Lack of) Filtering of invalid selections The filtering of invalid selections after decryption is straightforward but not done; we *strongly* recommend doing this. At present the system relies on the `control components` to filter invalid selections at voting time but this is challenging because they do not see the votes in plaintext. To perform this action they rely on the allowlist provided by the `setup component`; this mechanism, however, only works when the `setup component` follows the protocol.

– *Status* – Post has retracted their initial response to this issue; we do not appear to have received any updated response from them at the time of writing.

We understand from the Chancellery that a new measure will be added to the catalogue which will require this issue to be addressed in the future.

(Lack of) Verification of the semantic information in the prime mapping table Post has added semantic information to the prime mapping table which we believe is supposed to address similar concerns to those we raise here. However, only the verifier appears to check the semantic information with any other source of information; consequently, the semantic information is of extremely limited use at present. The semantic information should be checked by the voting client and control components as well.

Voting Client should check that the plain language description of the voting options it receives, and displays to the voter, agrees with the semantic information in the pTable.

Control Components should check the correctness of the semantic information when it receives the pTable in the ElectionContextProcesser. At present it is unclear against what the `control components` could check this data; this highlights the complexity of the use of the `control components` as a source of truth regarding the votes and their meaning. One clear improvement would be to send the canton election configuration file to the control components and have them check the consistency of this configuration with the semantic information.

– *Status* – Post appears to have misunderstood our comment here, we are well aware that the semantic information is included in the zero-knowledge proofs. However, this semantic information is not checked against any other information in the voting client or control component. The current solution makes the components agree on a bitstring which they do not use for any purpose; this is what we meant when we said it as of limited value.

(Lack of) Restrictions on calling micro-services It appears many micro-services run when they are not needed; for example, we cannot see any reason why the micro-services implementing the config phase functionality of the `control components` cannot be called in the voting phase.

In most cases there are restrictions which appear to stop this odd behaviour from affecting security. We are particularly concerned if the `GenEncLongCodeSharesProcessor` runs during the voting phase since this would allow the `control component`'s view of the eligible voters and the allow list to be tampered with after the `control component`'s view was verified the config verification.

– *Status* – Post has agreed to add further restrictions on when microservices can run in version 1.4.

(Lack of) Verification of voter numbers in the `control components` The `control components` receive a view of the number of eligible voters in each verification card set but this view does not appear to ever be used. We suggest adding a check to verify config which ensures the control components agree on the number of eligible voters; we further suggest that the `GenEncLongCodeSharesProcessor` checks that the number of verification cards it processes does not exceeded the number of eligible voters.

– *Status* – Post appears to have misunderstood our comment here. We are aware of the consistency checks the control components do internally to themselves which respect to the election configuration. We would still like to see:

1. A check in the verify config phase of the verifier which checks the consistency of the election configuration received by the control components.
2. A check in the `GenEncLongCodeSharesProcessor` that the number of voting cards processed is less than or equal to the number of eligible voters.

(Lack of) Consensus on the public key (in the specification) The verifier specification does not include any direct check that the `control component`s and the `verifier` agree on the public key. The check `VerifyCcmElectionPublicKeyConsistency` checks that the `setup component` and `control component`s agree on the `control component`'s share of the key but not the key as a whole.

Many of the zero-knowledge proofs which appear in the system include the election public key as auxiliary information. Each `control component` checks these proofs using its local view of the public key and then sends these proofs to the `verifier` which checks the proofs are the same in `VerifyConfirmedEncryptedVotesConsistency` and that they are valid with its local view of the public key in `VerifyOnlineControlComponents`. The details of `VerifyConfirmedEncryptedVotesConsistency` (which ensure that the proofs are the same) are not present in the specification.

– *Status* – Post's response did not address this issue. To summarise the problem, the current specification achieves consensus of the public key and prime mapping table indirectly by use the voters' zero-knowledge proofs.

This should work in practice, especially because the implementation checks the consistent view of these proofs; however, it creates additional and unnecessary trust assumptions on consensus.

(Lack of) Consensus on the prime mapping table (in the specification) The same discussion above directly applies to the prime mapping table.

6 Editorial notes

In the Specification document:

- In Algorithm 3.1, on the last line, $\forall i \in [0, \omega - 1)$ should read $\forall i \in [0, \omega)$.
- In Algorithm 3.9, the "Extended authentication factor" appears for the first time, without any explanation. It is only in Section 4.1.4, 14 pages later in the document, that it is suggested that it could be the voter birth date. It would help the reader to explain this around Algorithm 3.9.

– *Status* – Both of these have been addressed in version 1.3.1

References

[1] Véronique Cortier, Fabienne Eigner, Steve Kremer, Matteo Maffei, and Cyrille Wiedling. Type-based verification of electronic voting protocols. In *POST*, volume 9036 of *Lecture Notes in Computer Science*, pages 303–323. Springer, 2015.

[2] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. Sok: Verifiability notions for e-voting protocols. In *IEEE Symposium on Security and Privacy*, pages 779–798. IEEE Computer Society, 2016.

[3] Thomas Haines, Olivier Pereira, and Vanessa Teague. Report on the swiss post e-voting system. `https://www.newsd.admin.ch/newsd/message/attachments/71147.pdf`, March 2022.

[4] Thomas Haines, Olivier Pereira, and Vanessa Teague. Second addendum to report on the swiss post e-voting system. `https://www.bk.admin.ch/dam/bk/en/dokumente/pore/E_Voting/Examination_Reports_March2023/Scopes%201%20and%202%20Final%20Report%20Addendum%202%20Thomas%20Haines,%20Olivier%20Pereira,%20Vanessa%20Teague%2013.02.2023.pdf.download.pdf`, February 2023.

[5] Hugo Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. Cryptology ePrint Archive, Paper 2010/264, 2010. `https://eprint.iacr.org/2010/264`.

[6] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 2010.

[7] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash attacks on the verifiability of e-voting systems. In *IEEE Symposium on Security and Privacy*, pages 395–409. IEEE Computer Society, 2012.

[8] Ben Smyth, Steven Frink, and Michael R. Clarkson. Computational election verifiability: Definitions and an analysis of helios and JCJ. *IACR Cryptol. ePrint Arch.*, page 233, 2015.

[9] Swiss Federal Chancellery. Federal chancellery ordinance on electronic voting of 25 may 2022. Available from `https://www.fedlex.admin.ch/eli/cc/2022/336/en`, May 2022.