Report on the Swiss Post e-Voting System

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

March 24, 2022

1 Executive Summary

The Swiss Post e-voting system protocol documentation, code and security proofs show continuing improvement. The clarity of the protocol and documentation is much improved on earlier versions. This improved clarity has exposed many issues that were already present but not visible in the earlier versions of the system; this is progress.

The project and system are highly complex and, for the moment, the review process is adding to the list of open questions rather than reducing it. There are, at present, significant gaps in the protocol specification, verification specification, and proofs. We continue to find issues which we had not noticed in previous iterations. And, as several parts of the system documentation remain missing, our evaluation could not consider the system in full.

Furthermore, and as acknowledged by Swiss Post, several of the issues that we found require structural changes and additions to the current protocol. These protocol evolution steps require subtle cryptographic engineering and have an impact on the alignment between the protocol and the Ordinance on Electronic Voting (OEV), on the security proofs and on the code. These will in turn require new rounds of review.

Consequently, we present this report as an annual milestone rather than a final document, and structure it as such. First we summarise our findings as they relate to Scope 1 of the audit (Sec. 1.1), and Scope 2 (Sec. 1.2), then we discuss recent conversations on ordinance requirements in (Sec. 1.3).

For transparency, the two detailed reports we provided to the Federal Chancellery can be found in Appendices A and B—in a limited number of cases, these are updated based on feedback on earlier drafts. The detailed reports clarify which documents were analysed. A status report on the issues raised in our August report can be found in section A.6.

We encourage the stakeholders in Swiss e-voting to allow adequate time for the system to thoroughly reviewed before restarting the use of e-voting. As we highlight in both our reports, we focused on general issues delaying proper analysis of specific issues until after the general issues are resolved. While progress is being made, it seems likely that the review process will need at least as much time again to properly conclude that the system meets the requirements.

Most of our efforts were dedicated to the evaluation of the conformity of the protocol specification and computational security proofs to the requirements of the OEV (Draft of April 28, 2021) – this corresponds to Scope 1 of the review process. We also dedicated some efforts to the investigation of the conformity of the code to the protocol specification – this belongs to Scope 2.

1.1 Scope 1

The system specification and protocol proof are much improved compared to the status of 2020, but serious issues remain, which are detailed in the reports available in Appendices A and B. We highlight two of these issues here.

The protocol specification is substantially narrower in scope than required by the ordinance. This can lead to various problems.

1. The authentication of data is sketched but not detailed. Following our questions on this subject, Swiss Post realized that the individual verifiability requirement of the system was not satisfied, due to improper authentication.

   Attack: We detailed an attack on individual verifiability (Sec. A.5) due to insufficient signature validation. The attack workedby allowing an adversary to spoof the trusted participants and take control of the protocol.

   The specification was amended on this specific issue (in v.0.9.7), but there is still no general documentation on the data authentication mechanisms.[1] Swiss Post informed us that they are planning to provide this in the future. Swiss Post is implementing changes which prevent the specific attackwe raised; however, we are concerned that the underlying vulnerabilities still exist, at least for now, and other attacks may be possible.

2. The handling of data inconsistencies is not detailed. In a security model in which many system components must not be trusted, it must be expected that components do not claim to have the same view of the system status, and the protocol must specify how inconsistencies are handled.

   Attack: This leads to a class of attacks on individual verifiability (Sec. A.4); We showed, for instance, how the log validation process that is currently specified could lead to the rejection of a ballot that should be counted. We also demonstrated that, in some situations that are compatible with the OEV security model, the logs do not contain enough information to even decide whether a ballot should be counted or not.

   The issue here is that the handling of inconsistencies between the CCRs is, at present, undefined.

[1] See our discussion in Appendix A.5 and discussion on the attack and solution by Swiss Post on `https://gitlab.com/swisspost-evoting/e-voting/e-voting/-/issues/1`.

Swiss Post informed us that they are modifying their protocol in order to address this issue by adding additional verification and consistency checks to the CCRs. The wider issue of detailing how inconsistencies, and verification failures, are managed, still needs to be addressed.

The roles and channels in the protocol specification remain only incompletely aligned with those authorized by the OEV. As a result, we showed that an attack that breaks the secrecy of the votes does not exist within the security model that is described in the current protocol specification, but does exist if the protocol is considered in a security model where the roles and channels are aligned with the OEV requirements.[2]

Attack: We detailed an attack on vote privacy due to lack of ZK proofs of correct key generation (Sec. A.3). The attack exploits the fact that the CCMs are not required to prove knowledge of their share of the secret key. The attack does not work in the trust model listed in the protocol specification but does work in the trust model of the ordinance. This attack highlights discrepancies between the trust model of the protocol and the required trust model, which need to be addressed.

Swiss Post informed us that they are modifying their protocol in order to address this issue.

General consequences The elements that are missing in the protocol specification and the misalignments between the roles and channels of the specification w.r.t. the OEV also reflect on the security definitions and proofs. As an example, we observe that, when inconsistent logs are taken into account and reconciled, the success probability of an attacker trying to cast a ballot without a voter's approval may be around 4 times as high as what the current security theorems assert.[3]

1.2 Scope 2

Scope 2 of the audit covered the software. Our principal focus was on the alignment between the software and the protocol documentation.

- At the time of our review the source code and verification specification were still being developed and it was not possible to do a thorough review. We look forward to undertaking a proper review in the future but this will take time.
- Several of the design choices in the software have security implications which are not considered in the documentation (Sec. A.2).
- It was difficult to find the alignment between the software and the specification. Comments should be added to clearly articulate the correspondence.

[2] See discussions in Appendix A.1.3 and A.3
[3] See discussion in Appendix A.4.3.

- The verifier specification does not adequately detail the structure of the-information logged and how that information is transmitted and received (Sec. A.2.3).

1.3 Verifiability and Privacy when all Control Components are run by private companies

One point which has been much discussed since we submitted our December report is the level of privacy and verifiability the system should achieve when all control components–from a particular group–are controlled by private companies.

We are strongly of the view that the ordinance should require the highest levels of verifiability and privacy–that are technically feasible–even if all control components controlled by private companies are corrupt. The draft ordinance already requires a high level of privacy in this case but does not explicitly require this for verifiability.

Specifically, we suggest adding that "when all the CCRs are corrupted, then verifiability must hold up to the fact that ballots can be dropped." This is a meaningful difference compared to no verifiability at all (as could be the case now): it guarantees for instance that ballots that have been verified by the voters cannot be modified – they can only be dropped. Modification is much worse than dropping: it does not change the number of ballots (so, it is more complicated to spot), and switching a vote for A to a vote for B has an effect that is twice as important as just dropping a vote for A. Then, privacy becomes simple to explain: votes must be as private as what can be expected from the level of verifiability that is required in the same model.

There is significant work to be done by Post, in collaboration with the academic partners, to present formal definitions for these properties.

Whatever the precise formulation, there remains an important practical question: what should the canton (or other authority) do if it sees an abnormally small number of ballots in a bulletin board, and how do we define what "abnormally small" is? But we think that this question should not be treated at the level of the privacy definition.

Contents

References

[1] Myrto Arapinis, Véronique Cortier, Steve Kremer, and Mark Ryan. Practical everlasting privacy. In *International Conference on Principles of Security and Trust*, pages 21–40. Springer, 2013.

[2] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. Sok: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy*, pages 499–516. IEEE Computer Society, 2015.

[3] David Bernhard, Ngoc Khanh Nguyen, and Bogdan Warinschi. Adaptive proofs have straightline extractors (in the random oracle model). In *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017*, volume 10355 of *Lecture Notes in Computer Science*, pages 336–353. Springer, 2017.

[4] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. Version 0.5, 2020.

[5] Johannes Buchmann, Denise Demirel, and Jeroen Van De Graaf. Towards a publicly-verifiable mix-net providing everlasting privacy. In *International Conference on Financial Cryptography and Data Security*, pages 197–204. Springer, 2013.

[6] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[7] Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. Multi-autority secret-ballot elections with linear work. In *Advances in Cryptology - EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1996.

[8] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, page 291–304. ACM, 1985.

[9] Thomas Haines and Clementine Gritti. Improvements in everlasting privacy: efficient and secure zero knowledge proofs. In *International Joint Conference on Electronic Voting*, pages 116–133. Springer, 2019.

[10] H. Krawczyk and P. Eronen. Hmac-based extract-and-expand key derivation function (hkdf). RFC 5869 – `https://datatracker.ietf.org/doc/html/rfc5869`, May 2010.

[11] Karola Marky, Oksana Kulyk, and Melanie Volkamer. Comparative usability evaluation of cast-as-intended verification approaches in internet voting. In *Sicherheit*, volume P-281 of *LNI*, pages 197–208. Gesellschaft für Informatik e.V., 2018.

[12] Microsoft. Electionguard - baseline election parameters. `https://github.com/microsoft/electionguard/blob/main/docs/spec/1.0.0/3_Baseline_parameters.md`.

[13] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *Annual International Cryptology Conference*, pages 373–392. Springer, 2006.

[14] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptol.*, 15(2):75–96, 2002.

A December Report

The state of the code, documentation and our investigation

The Swiss Post e-voting system shows a continuing process of improvement—the protocol and security proofs are much clearer than in earlier versions. In many cases, the improved clarity has exposed issues that were probably always present—this is progress.

The scale and complexity of the project are extremely important and, for the moment, it still appears that every added layer of documentation brings new issues and inconsistencies to light, rather than closing the list of open questions. There are some significant gaps in the protocol specification, verification specification, and proofs—we, too, continue to find issues that we had not noticed in previous iterations.

This was pointed out in the first draft of this report from July 2021 and, despite many changes, this remains true with the current version of the system, which consequently is not yet ready to be thoroughly reviewed. Similarly, various important gaps are found when investigating the consistency between the protocol specification and the code.

In this report, we have attempted to focus on general or major issues with the current version and will leave more specific issues to be properly analysed once the general issues have been resolved. When there are significant discrepancies between the specification and the code, there is not much gain in analysing the specification because there is a good chance it will change.

Our analysis is based on the documents available for the review of the Swiss Post e-voting system and the Swiss Federal Chancellery's updated requirements. Regarding Scope 1, these are:

- The Draft of April 28, 2021, of the Federal Chancellery Ordinance on Electronic Voting (OEV);
- The Cryptographic Primitives of the Swiss Post Voting System, Version 0.9.8 (superseding Version 0.9.5 analyzed in our first draft);
- The Protocol of the Swiss Post Voting System, Version 0.9.11 (superseding Version 0.9.10 analyzed in our first draft);
- The Verifier Specification of the Swiss Post Voting System, Version 0.9.1.

In some cases, we also looked for clarifications in:

- The System Specification of the Swiss Post Voting System, Version 0.9.7 (superseding Version 0.9.6 analyzed in our first draft).

Regarding Scope 2, the following documents and code have been examined:

- The Verifier Specification of the Swiss Post Voting System, Version 0.9.1;

- Source Code of the Verifier of the Swiss Post Voting System, Version 0.8.1.0;
- Source Code of the Swiss Post Voting System, Version 0.12.0.0.

The next section of this report describes our main new findings for Scope 1, the documentation review. Section A.2 gives an overview of new findings for Scope 2, including the code, verification spec, and discrepancies between these and other documents. Section A.3 examines issues related to the absence of some ZK proofs and to discrepancies between the role of some protocol participants in the protocol specification compared to the OEV. Section A.4 examines issues relating to consistency between control component views, and how these impact the proof of vote confirmation security. Finally, Section A.5 describes an issue concerning signature verification, which was disclosed via the Swisspost git. Appendix A.6 is a status report on corrections of the issues raised in our draft report from July 2021. Appendix A.7 is a status report on the various Gitlab issues we have been involved in. As a general pattern, SwissPost has promptly corrected those issues that have an immediate solution, but the more involved and challenging issues will take longer.

A.1 Scope 1

A.1.1 Summary

The materials covered in Scope 1, including the system specification and protocol proof, are much improved over previous versions but serious issues remain.

Incomplete Protocol Specification The protocol specification is substantially narrower in scope than required by the ordinance. We highlight three important aspects below:

- The security of the system depends on how data is authenticated, which is sketched but not detailed. We pointed out the absence of specification of the authentication mechanisms, and highlighted some of the associated potential risks, which led Swiss Post to inspect these mechanisms and uncover an attack against individual verifiability.[4] We believe that this stresses the importance of the completeness of the protocol specification.
- The security of the system also depends on when and by whom various processes can be called, which is not detailed. A proper level of abstraction needs to be chosen and clearly communicated with respect to both these properties.
- The protocol specification focuses on protocol executions in which all the system component actions are synchronous. The verifier specification in some places specifies that verification fails in the case of inconsistency, but

[4] See `https://gitlab.com/swisspost-evoting/e-voting/e-voting/-/issues/1`.

the verification sketch in the System Specification (for example, 12.2.3 - VerifyVotingPhase) only checks the number, not the values, of vote confirmation code attempts. In still other cases, the documents say only that inconsistencies will be investigated.

The OEV No 2.5 requires "As a condition for the successful examination of the proof referred to in Number 2.6, all control components must have recorded the same votes as having been cast in conformity with the system. Cases where the control components show inconsistencies in this respect must be anticipated in accordance with Number 11.11 and the procedure determined in advance." It is the last sentence of the above quote which is not completely addressed by the current documents.

Given the discrepancy that it creates w.r.t. the OEV, the potential enormous complexity of interpreting the inconsistencies and the likely pressure to avoid rerunning the election, we strongly recommend that this area receive far greater attention than it has to date. We have worked through the implications in some detail for the final return of vote cast confirmation codes, and made some specific recommendations, (see Section A.4), but it may be relevant in other parts of the protocol too.

Divergence between roles in the Protocol Specification and in the OEV The security model and communication channels associated to some of the protocol participants, as described in the protocol specification, seem to be incompatible with the OEV.

In particular, the role of the auditors and of the electoral board, as described in the protocol specification, appears to be problematic. This will be discussed in Section A.3.1.

Incomplete security model and proofs

- The elements that we noted to be missing in the protocol specification are also missing in the security model and definitions. As a result, the security model and definitions are not aligned with how the system is intended to be used and may miss attacks on the system. This is not aligned with the OEV requirements (e.g., Art. 2.14 of the OEV Appendix).

- Apart from these missing elements, we highlight several places where we could not understand why the current security definitions would offer the desired guarantees (despite the fact that we are accustomed to reading and writing such definitions). It is important to add a few paragraphs explaining why the security definitions that are proposed in the text can be used to demonstrate that the security requirements of the OEV are satisfied. Especially with privacy, it seems clear that Definition 17 does not demonstrate that the cryptographic protocol meets the requirements (e.g., Art. 2.14.1 of the OEV Appendix). We highlight specific examples in Section A.1.4.

- The missing elements in the security definitions must of course be expected to have an impact on the guarantees that are claimed in the security theorems. For instance, in Section A.4 we show that some of the (missing) mechanisms that appear to be necessary to deal with possible inconsistencies in the logs of control components would increase the probability of success of some attack strategies, violating the currently claimed security bounds. As a result, the bounds that are claimed in the security theorems offer a view of the actual security of the protocol that is too optimistic.

- Although the security proofs are significantly improved, there are still numerous holes. This highlights the underspecification of various basic security checks in the system as it is defined in the proof document. Furthermore, we were often confused by the very high-level arguments that are provided to demonstrate that successive games must be indistinguishable. Proper reductions to the underlying computational assumptions should be provided, rather than simply invoking these assumptions. We highlight specific examples in Sections A.1.4, A.3, A.4 and A.5.

Consider a major redesign The current protocol does a good job at preventing the attacks that were previously proposed. However, it is quite visible that, in many places, the protocol is the result of a long evolution and has become an addition of patches.

We would strongly encourage Swiss Post to consider the design of a cleaned, minimal, version of the voting protocol, and to add an extra section in the cryptographic protocol document, that would offer a clear view of the role of each cryptographic operation (and what would break in the protocol if each of these operations were missing). We suggest specific simplifications of the protocol in several places in the document, but it is clear that much more can be done, with expected benefits in security, efficiency, and clarity.

A.1.2 Observations regarding the OEV, Draft of 28 April 2021

The new draft ordinance document clarifies many of the requirements that were present in the previous version, which is a very positive thing. As a downside, we feel concerned that it also matches more and more closely what a properly implemented version of the Swiss Post protocol should be doing. Of course, there is a low-level design space that is quite open and that other system providers could explore. But it also does nothing to offer incentives to the design of *stronger* systems. As a result, it does not offer any incentive for a system supplier (Swiss Post, or any other one) to explore the possibilities of offering something more secure than the minimum that the OEV requires.

Would it make sense to refine the requirements by identifying various grades at which security properties are realized by a system? These could be identified from the properties discussed in the dialog of 2020, so that a system would not just be declared to be compliant, but would also receive a grade, so that cantons

would have the possibility to choose a system that obtains a higher grade? This would foster diversity, which in turn is good because it would promote quality.

Here are some possibilities that come to our mind in particular:

- The notions of individual and universal verifiability that are required here are much weaker than those that are standard in the academic literature (the standard notions in the literature are incompatible with the assumption of a trusted print office, or of control components among which one must be honest). In some cases, these differences allow for better usability or other desirable properties; in others, it is unclear why the system should not be required to achieve a higher level of security.

  For example, there is evidence that voters are more likely to successfully perform a code-voting check than a challenge-based ciphertext verification [11] (such as that used in the Estonian system), so in a sense these two options are incomparable: code-return systems inherently have stronger trust assumptions, but problems within the trust assumptions may be more likely to be noticed, while challenge-based systems require an independent device and a lot more work from the voter, and may be successfully used less frequently. The difficult tradeoff between usability and minimal trust for individual verifiability remains perhaps the most important open problem in evoting—we are not suggesting that a better solution than the OEV's current model exists, but rather that the OEV should be sufficiently flexible to incentivise better solutions if they are invented.

  Furthermore, not all the trust assumptions allowed by the OEV are necessary for code-return systems. For example, much of the trusted setup that was shifted into the 'print office' in Scytl's original design is already being moved into a verifiable setup phase by Swiss Post. This is a significant improvement, and more improvements could be made.

  Would it make sense to offer a higher grade to a system that would remove some or all of the current trust assumptions?

  - Articles 2.9 and 2.10 of the annex list participants and channels that *may* be considered trustworthy. It would be interesting to value systems that would not need to consider as trustworthy some or all of these components.
  - The print component is currently a central trusted component. Exploring protocols that would reduce the requirements on this system component should be valued.

- Consider making the universal verifiability proofs open to the public using techniques which do so without impacting long term privacy, possibly building on techniques from [7, 13, 1, 5, 9]. We note that it would probably only make sense to do this if there is a way to add public eligibility verifiability to the system at the same time – the benefit of publishing the

proofs appears to be quite low otherwise, compared to the additional risks (resulting from a bad PRG for instance).

Here are some more specific remarks:

- We are not sure whether the OEV states anything about the handling of a failure event in the cryptographic protocol (say, a system component observes that a ZK proof produced by an other control component does not verify) – the OEV (e.g., Art. 14.1 of the Appendix) seems to rather focus on hardware failures, which are not necessarily a sign of malfeasance. Intuitively, it would seem that a control component cryptographic failure should be taken very seriously, and the offending control component should probably be removed, whereas a cheating client is expected and should not necessarily result in strong action. However, it is not clear to us whether the requirements currently require, and the protocol certainly does not provide, accountability, so it may not be possible to identify the source of the problem. As a result, and in the absence of any accountability or robustness requirement, and of a description of what should happen in case of failure in the protocol specification, it appears that the voting system should just be halted. But this is also opening an important channel for denial of service attacks, and the impact of such a decision would probably be so high that it may not happen even if it would be technically sound. Would it make sense to further clarify requirements of accountability and robustness in the OEV?

A.1.3 Alignment issues between the ordinance and Swiss Post documents

Distribution of control components Art 2.d of the explanation of the ordinance says "control components are separate components of the system that are designed in a variety of ways, operated by different persons and secured by special means;" It appears from the documents that all components are either run by Post or the Canton and share a common code base. This seems inconsistent with "designed in a variety of ways" and "operated by different persons."

A.1.4 Protocol of the Swiss Post Voting System

Outstanding Major issues

- The games in the formal security definitions restrict the adversary's action far more than the system does. This means attacks may be missed. The rest of this section is replete with examples; the central issue is that games do not model the ability of the adversary to control the execution flow of the protocol which in the real system it has significant ability to do through a corrupt voting server.

- In the privacy game it remains unclear what inputs to the oracles are under the adversary's control and which are under the challenger's control. It seems clear that it cannot solely be under the adversary's control or this would invalidate the proofs. This is concerning because it might hide missing validation checks which are required to prevent attacks.

  For example, if the adversary is free to chose bb_{clean} in the online decryption oracle then they can submit a board for mixing and decryption which does not include certain cast ballots; this would cause the proof to fail because the number of elements in $c_{Dec,h}$ would differ depending on β.

- We strongly recommend introducing oracles to many of the games, as has already been done with the privacy game, to better capture the adversary's ability in the real system.

  For example, in the vote rejection experiment the adversary should be given oracle access to the honest CCR which it should be allowed to call many times. The current case only considers the adversary being able place one input to the honest CCR even though in the real system it would clearly be able to do this many times. Strangely, the bound shown in the proof assumes the adversary can call the honest CCR many times; we suspect this is a result of Swiss Post attempting to patch the incorrect bound, after we raised it in our last report, without fixing the underlying issue.

- The main abstraction used for authentication and data consistency in the proof document is injective agreement. Unlike a two party protocol (as in Lowe's definition of injective agreement), it is not immediately clear what injective agreement would mean for this protocol; a definition of injective agreement for the kind of protocol under analysis needs to be given. For instance, we do not know what it means that the setup component and the control components "must achieve injective agreement" (Sec. 13.1.1) or that the voter and the control components "must achieve injective agreement" (Sec. 13.1.2). This notion of agreement is particularly tricky when some of the parties are malicious, as it must be expected here. This also touches the problems discussed in Section A.4 of this report.

Specific issues Several of the issues detailed below are related to holes in the security proof which do correspond to real attacks; the attacks do not work on the protocol as described in the system specification but they do work on the protocol as described in the proof document. This is commonly because the auxiliary information used in the zero-knowledge proofs is not specified in the proof document. While we are not aware of any attacks in this category which bypass the measures in the system specification (except the one described in Section A.4), the holes in the proof should be closed to ensure there aren't any.

- The definition of CombineEncLongCodeShares$ is incomplete since the (zero-knowledge) proofs it receives may not be extractable. For example, they could be copies of the output of the honest party.

- Section 19.1, Proof of Lemma 1. The first hop relies upon the fact that the (zero-knowledge) proof made by the honest CCR doesn't break anything. This isn't guaranteed by weak simulation extractability, which would require duplicates to be filtered. Strictly speaking it is the second hop which is invalid because it is at that point that the proof would fail.

  The attack this flaw in the proof is covering is prevented by details of the system specification which don't appear in the proof version of the protocol. Specifically, the inclusion of the voter identifier into the auxiliary information of the zero-knowledge proof prevents the copying of the honest ballots since they are then invalid.

- 19.1 Proof of Lemma 1. It is unclear why the extraction of the secret keys from the adversarial CCRs can work as it is described. Extraction is assumed to work for one proof, but this may not be enough to guarantee that it can be performed for many proofs: there are famous examples where the extraction complexity grows exponentially with the number of proofs for which extraction is needed [14]. It is known that straightline extractors are needed when extraction must be performed on adaptively generated proofs [3], but the proofs used here do not have a straightline extractor. It is unclear whether this breaks the proof, but the current version of the proof does nothing to explain why it is possible to extract from many proofs in the context of the current protocol.

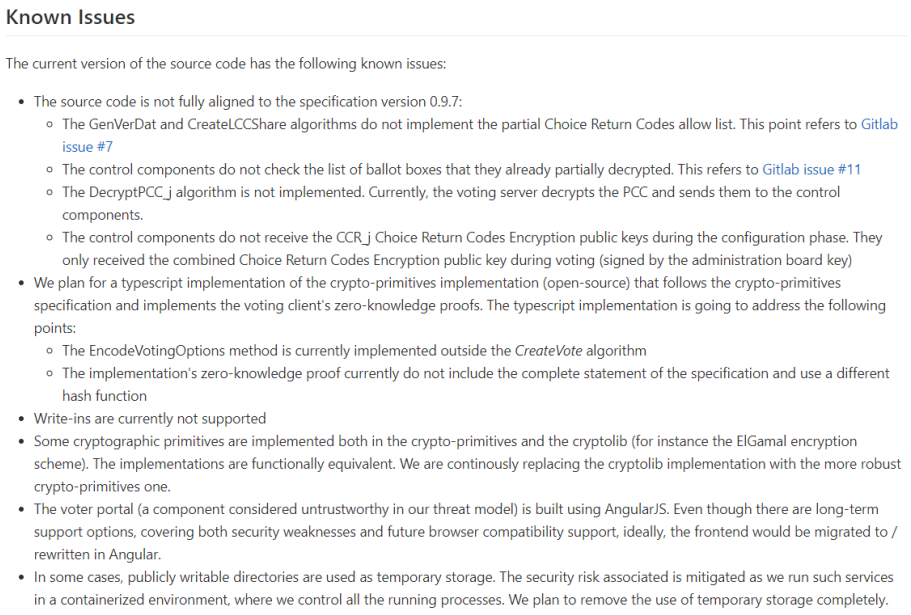
- Figure 28 and Lemma 2.
  - The lack of $\texttt{vc}_{\texttt{id}}$ in Valid ballot will allow ballot copying attacks.
  - The game doesn't seem to include the adversary controlling more than one voter, or other voters voting

- Page 96 there are several incorrect statements.
  - A compliant ballot may not be valid since the proofs could be invalid even if everything else is correct. However, a valid ballot should be compliant.
  - An honest generated ballot will be valid and compliant but it won't be compliant and registered till it is processed.

Minor issues

- The argument for concatenation over truncation on page 19 is not particularly convincing. Since ElGamal is malleable if the adversary can manipulate the message, there are more serious problems than the ability to drop part of the ciphertext.

- It might be easier to acknowledge that the definitions are concrete rather than asymptotic and simplify the notation on page 10



Known Issues

The current version of the source code has the following known issues:

- The source code is not fully aligned to the specification version 0.9.7:
  - The GenVerDat and CreateLCCShare algorithms do not implement the partial Choice Return Codes allow list. This point refers to Gitlab issue #7
  - The control components do not check the list of ballot boxes that they already partially decrypted. This refers to Gitlab issue #11
  - The DecryptPCC_j algorithm is not implemented. Currently, the voting server decrypts the PCC and sends them to the control components.
  - The control components do not receive the CCR_j Choice Return Codes Encryption public keys during the configuration phase. They only received the combined Choice Return Codes Encryption public key during voting (signed by the administration board key)
- We plan for a typescript implementation of the crypto-primitives implementation (open-source) that follows the crypto-primitives specification and implements the voting client's zero-knowledge proofs. The typescript implementation is going to address the following points:
  - The EncodeVotingOptions method is currently implemented outside the *CreateVote* algorithm
  - The implementation's zero-knowledge proof currently do not include the complete statement of the specification and use a different hash function
- Write-ins are currently not supported
- Some cryptographic primitives are implemented both in the crypto-primitives and the cryptolib (for instance the ElGamal encryption scheme). The implementations are functionally equivalent. We are continously replacing the cryptolib implementation with the more robust crypto-primitives one.
- The voter portal (a component considered untrustworthy in our threat model) is built using AngularJS. Even though there are long-term support options, covering both security weaknesses and future browser compatibility support, ideally, the frontend would be migrated to / rewritten in Angular.
- In some cases, publicly writable directories are used as temporary storage. The security risk associated is mitigated as we run such services in a containerized environment, where we control all the running processes. We plan to remove the use of temporary storage completely.

Figure 1: Known issues with e-voting system code at time of writing

- Should $g_1, ..., g_L$ be given to the adversary in ESGSP? The reduction does not appear to work without doing so. We raised this issue in our draft report for the DDH game (Fig 16, version 0.9.11) and it has now been corrected, but we neglected to mention the ESGSP game (Fig 17) - this should be corrected similarly.

- The use of correctnessIDs and interaction with the allow list should be better documented. In particular, the connection to the allow list of the partial choice codes should be made clearer.

A.2 Scope 2

A.2.1 Summary

The documents covered in scope 2, including the source code and the verification specification, have improved in the security of the cryptographic primitives, but currently have a large number of issues in terms of the protocol as a whole. Most of these issues are well known to Swiss Post but are still awaiting rectification: see Figures 1 and 2 which show the current known issues listed in the various public Gitlab repositories.

- Our chief concern about the code is the lack of analysis around the use of micro services. Swiss Post's use of micro services is well motivated from a

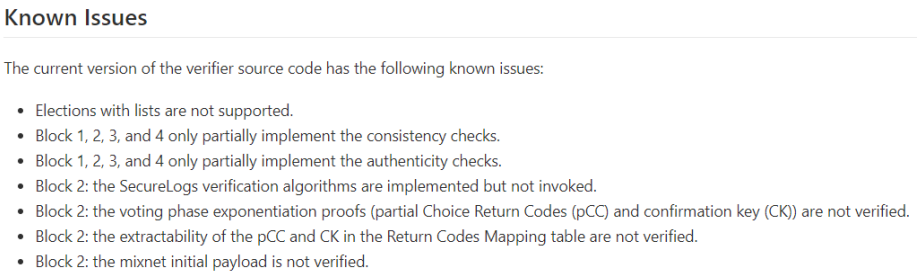


Figure 2: Known issues with e-voting verifier code at time of writing

technical viewpoint but the released documents do not provide evidence that they have thought through the security implications.

For example, the micro services allow the functions to be called many times on a wide range of data, but the security model in the proof assumes most functions can be called only once on a very rigid input. This is a significant discrepancy between the specification and the code, that makes it really hard to decide whether the code does what the specification says, while making it quite possible that the code will lead to many more possible system states than what the specification allows.

- The code is currently missing a good deal of documentation (comments) to properly demonstrate its alignment to the protocol specification.
- The protocol specification considers a single election—as represented by an election id—running at once, whereas the code will run multiple elections in parallel. So when the specification says "we only do this once per voterid" the code often says "we only do this once per voterid per election." This misalignment between the specification and the code could be a source of serious errors and should be rectified.
- The process by which certificates are generated, transmitted and verified is not adequately described in any existing documentation. Given how crucial this process is for security we strongly encourage Swiss Post to expand the documentation to cover this.

A.2.2 Code

Important issues

- It is currently difficult to determine what part of the code is implementing what part of the specification. We encourage Swiss Post to put clear comments in the code which detail how it works within the protocol specification. Anyone should be able to easily find the alignment simply by searching for the names of algorithms.

- The micro services, though they implement various checks, rarely check the data for consistency with their view of the election state, which makes checking the possible execution flows hard. This problem seems to be particularly important because the protocol specification document assumes a very coordinated execution flow.
- The functions VerifyBallotCCR, PartialDecryptPCC and ReturnCodesExponentiationConsumer are used by the control components to decrypt partial choice code and create the long choice return code. They maintain state using computed verification card IDs which store the state of requests. These are indexed by verification card id and election id. This allows the same voter id to be processed twice which violates the "ensure" in Algo 5.6 CreateLCCShare. In theory this might break individual verifiability since the adversary could retrieve multiple sets of return codes for the same voter. This attack does not work on the real system since the election id will change, so the control component will use a different secret key to exponentiate and different long choice return codes will be generated.

Recommendations

- The code for GenEncLongCodeShares should check the request comes from the setup component.
- The code for CombineEncLongCodeShares should check that the input comes from the expected CCRs. That is, it has exactly one input set from each CCR.
- The CCRs should keep logs of what Voter Cards they create return codes for and this should be checked for consistency with each other and the print office.
- All components which contribute to the generation of the secret key used to encrypt votes should prove they know their share of the secret and these proofs should be checked.

Minor issues

- For data to be valid in ReturnCodesExponentiationConsumer a corresponding election id and vote id must have already been processed through decryption but not necessarily with the same data.

A.2.3 "Swiss Post Voting System: Verifier specification"

The principal issue with the audit specification is that it does not adequately detail the structure of the information logged and how that information is transmitted and received; put in other words it is difficult to tell where the inputs to the various algorithms are coming form. Since most of the parties the verifier

receives information from may be dishonest, the details of what data is received and from whom are vitally important.

Important issues

- It is important that the logging is used with the special termination message required. This is mentioned in the second paragraph of 4.3 but not in the actual algorithms.

Minor issues

- On page 23 when discussing checking that no confirmation attempts have been dropped it is written that the verifier must ensure $ca_{non-final} \cap ca_{final} = ca$. Presumably what is meant is $(ca_{non-final} \cup ca_{final}) = ca$. and $ca_{non-final} \cap ca_{final} = \emptyset$. That is, every confirmation attempt is either final or non-final.

- It doesn't seem anyone ever checks that the ids are actually unique. We are not sure if this leads to attacks. It might make sense to add such an algorithm to the VerifyConfigPhase.

A.2.4 Issues with the verifier

At present the verifier implementation is too incomplete to be analysed and we are going to wait until later for this analysis. Our main questions around its function relate to data consistency and authentication, both of which are listed as known issues and future work in the verifier's readme document. We elaborate on this question in the next few sections.

A.3 Absence of ZK proofs of correct key generation

The CCMs do not prove knowledge of the secret keys corresponding to the public key that they publish. This is important since the absence of these proofs means that a minority of parties may know the secret key, which should have been generated in a distributed manner.

The following attack illustrates discrepancies between the OEV, the protocol specification and the security proofs. Although we do not think it would work in the security model of the protocol specification, the proof does not characterise the possible attacks sufficiently. Even more importantly, this scenario shows a point in which the trust model of the protocol specification is inconsistent with the OEV.

An attack scenario on privacy Let us consider the following variation on the classical attack described in Sec. 13.6 of the protocol specification. We consider a case where the voting server, the election board and one of the online CCMs are controlled by the adversary. The adversary sees the inputs of the honest CCMs $(EL_{pk,1}, EL_{pk,2})$ through the voting server (Fig. 20 of protocol

specification) and creates a share which cancels them out. This is done by inverting their shares and adding one of its own $EL_{pk,3} = \frac{EL'_{pk,3}}{\prod_{i=1}^{2} EL_{pk,i}}$. The setup component acts honestly and computes $EL_{pk} = \prod_{i=1}^{2} EL_{pk,i} \cdot EB_{pk}$ which simplifies to $EL'_{pk,3} \cdot EB_{pk}$. At this point the adversary knows the secret key used to encrypt votes and can break privacy as the votes are submitted.

We observe that this attack scenario does not exist in the more abstract model that is used in the security proof, since that model considers one single online CCM (merging $\texttt{CCM}_1$, $\texttt{CCM}_2$ and $\texttt{CCM}_3$).

This attack would also not work in the security model of the protocol specification, because:

1. It is considered that some electoral board members cannot be corrupted (Table 1).

2. It is considered that the auditors, among which one of them is supposed to be honest, authorize the electoral board member to reveal their secret key to the offline CCM, and this would only happen after a successful mixing, which $\texttt{CCM}_3$ would not be able to complete. So, $\texttt{CCM}_4$ would never receive the decryption key shares.

A.3.1 Divergence between roles in the Protocol Specification and in the OEV

However, we believe that these two aspects of the protocol specification are incompatible with the OEV security model.

The electoral board The role of the electoral board is currently undefined. In Table 1 of the specification, the electoral board is not matched to any system participant of the OEV. As such, and following Art. 2.1 of the OEV Appendix, it should be placed within the "untrustworthy system" category. However, the protocol specification indicates, on p. 7, that "Even if some electoral board members are untrustworthy, we consider the electoral board trustworthy as a whole." We could not find any formal definition of "trustworthy as a whole". Our best guess comes from p. 89 of the protocol specification that indicates: "Besides, we omit Shamir's secret sharing algorithm that splits the offline mixing control component's electoral board secret key EB sk among the electoral board members". We then suspect that "trustworthy as a whole" means that the number of compromised members must be lower that the threshold that is chosen for the secret sharing scheme. In any case, this seems to be incompatible with the OEV that would mark the electoral board as untrustworthy. However, if the electoral board is untrustworthy, then the attack scenario described above works, and privacy is broken.

One possible way to solve this issue would be to declare that the electoral board is an extra control component group, and therefore cannot be completely compromised. This would require extra care because the electoral board key is specified (Sec. 13.2 of the protocol specification) to be shared with Shamir's

secret sharing scheme, which can accommodate any threshold, and identifying the electoral board as a control component would require to stick to the trivial case where all key shares are necessary in order to recover the secret (because otherwise 3 out of 4 dishonest participants could collude to decrypt). And, in this case, a simpler additive secret sharing scheme can be used instead of Shamir's.

The auditors Art. 2.2 of the OEV Appendix forbids any outgoing communication from the auditor and from its technical aid. This is consistent with Table 2 of the protocol specification, which indicates the communication channels with the auditors and their technical aid just as in the OEV Appendix. We also believe that it is a good practice to only require the participation of the auditors once the election is complete (which may not prevent auditors starting their task as soon as data are available – the whole protocol should just not depend on this).

However, as pointed above, the protocol specification also requires the auditors to complete VerifyOnlineTally and send information to the Electoral Board and last CCM before they complete the tally phase. Similarly, Figure 23 of the specification shows that the auditors must run VerifyVotingPhase before the tally phase starts, and that the beginning of the tally phase in conditioned to a successful verification of the voting phase by the auditors.

There are many ways to address these issues. One of them would be to create an additional auditing control component group that would take the role currently assigned to the auditors in the protocol specification (the auditors in the sense of the OEV would run the verification protocol once the election is complete). Another option would be to ask all the CCMs to run the VerifyVotingPhase themselves before they start tallying, and the electoral board to run VerifyOnlineTally before they release their keys to the offline $\texttt{CCM}_4$. We did not analyze these options in detail, and there certainly are other ones that could be considered.

A.3.2 Recommendations

The discussion of this section highlights important discrepancies between the OEV and the specification, and shows that non-trivial steps are required in order to solve these discrepancies: the technical choices that are made may have a vital impact on the security of the protocol.

Interestingly, it appears that one approach to solve these issues would be to make a major redesign of the tallying phase, in order to make it more standard. The current tallying protocol has several unusual aspects that seem to make it less efficient and weaker at the same time:

- The online CCMs all perform an expensive decryption step.
- The electoral board keeps a (shared) copy of a secret key generated in a centralized way by the setup component.

A standard way to proceed would be as follows:

- The CCMs perform all the mixing work without decrypting anything.
- The electoral board (or the CCMs) generate the election public key in a completely distributed way, each one proving the knowledge of their secret key share, and only start decrypting after verification that the mixing process was correct.

This would have several advantages:

- This would reduce latency in the mixing process: each CCM can multiply each of its input ciphertexts with a precomputed encryption of 1, shuffle the ciphertexts, and pass the result to the next CCM. This does not require any expensive exponentiation during the sequential shuffling process (contrary to a decryption task). The CCMs can then compute their proofs of correct shuffle all in parallel.
- This would remove an over-reliance on the setup component.
- This would overcome the discrepancies with the OEV.

A.4 The challenges of consistency checking for defending against attacks on vote confirmation and verification

This section concerns the very final step of the voting phase, in which a voter enters her ballot casting key $\texttt{BCK}_{\texttt{id}}$ at her client, which transforms it into $\texttt{CK}_{\texttt{id}}$ and sends it to the voting server. She should receive the correct Vote Cast Return Code $\texttt{VCC}_{\texttt{id}}$ only if her ballot will be included.

The adversary's objective is either to return the correct $\texttt{VCC}_{\texttt{id}}$ to the voter, while producing a vote transcript that leads to the rejection of her vote, or to produce a vote transcript that leads to the inclusion of a vote for which the voter never entered her ballot casting key $\texttt{BCK}_{\texttt{id}}$.

The attacks described in this report rely on some inconsistencies between the logs of different CCRs for the vote confirmation phase.[5] We find it fairly difficult to understand how the system would behave, should those inconsistencies happen. We believe that the treatment of these inconsistencies should be an explicit part of the protocol specification, and that the security proof should demonstrate why this treatment is compatible with the FCh OEV.

Our analysis suggests that a treatment of these inconsistencies that would be compatible with the OEV would be in contradiction with the current protocol security analysis, violating some of the claimed security bounds.

Our analysis focuses on specific examples. We do not currently have a proof that the proposed modifications in the protocol are sufficient, because there may be other attacks along similar lines. An updated protocol specification,

[5] This was reported to SwissPost as a gitlab issue: `https://gitlab.com/swisspost-evoting/verifier/verifier/-/issues/1`, which is currently confidential but should soon be visible.

and updated security proofs, could show how our attacks, and other similar ones, can be excluded.

A.4.1 What inconsistent logs should be permitted?

Let us consider CCR logs that are almost, but not perfectly, consistent. This may be the result of communication mishaps, of a corrupted voting server, or of one, or a few malicious CCRs for instance.

We focus on the $\mathsf{L}_{\mathtt{confirmed},j}$ logs and, in the rest of this discussion, we omit $\mathtt{lVCC}_{\mathtt{id}}$ and the ZKPs, because we assume these are honestly generated, consistent with the other data, and pass verification.

Omission Suppose three CCRs show a certain confirmation attempt but one missed it, so their logs look like:

$$\begin{array}{ll} \mathtt{CCR}_j: & (\mathtt{vc}_{\mathtt{id}}, 1, \mathtt{CK}_{\mathtt{id}}, *, *) \text{ for } j = 1, 2, 3. \\ \mathtt{CCR}_4: & \text{No record for } \mathtt{vc}_{\mathtt{id}} \end{array}$$

Such logs could appear in a scenario like the following one, in which a dishonest $\mathtt{CCR}_4$ colludes with a dishonest voting client and VS.

1. The client and server-side components all perform the vote-sending and Choice Return Code generation and return honestly. The client displays the (correct) Choice Return Code to the voter.

2. The voter enters her true Ballot Casting key $\mathtt{BCK}_{\mathtt{id}}$. The client honestly computes $\mathtt{CK}_{\mathtt{id}}$ and sends it to the Voting Server.

3. The Voting Server honestly forwards $\mathtt{CK}_{\mathtt{id}}$ to all the CCRs.

4. The honest CCRs ($j = 1, 2, 3$) perform all the steps of Section 12.2.2.2 of the protocol specification correctly, including logging, and return $\mathtt{lVCC}_{\mathtt{id},j}$ ($j = 1, 2, 3$) to the Voting Server.

5. Cheating $\mathtt{CCR}_4$ computes $\mathtt{lVCC}_{id,4}$ correctly, *but logs nothing and returns the value secretly to the Voting Server.*

6. The Voting server makes whatever logs are specified when it receives only three responses ($j = 1, 2, 3$). (This is currently not explicitly specified in Section 12.2.2.3.)

7. *The Voting server also computes correctly (but does not log) the value of* $\mathtt{lVCC}_{id}$ *derived from a correct execution of 12.2.2.3 using the* $\mathtt{lVCC}_{id,j}$*'s received from honest CCRs (*$j = 1, 2, 3$*), plus the* $\mathtt{lVCC}_{id,4}$ *it received out-of-band from the cheating* $\mathtt{CCR}_4$. This result should correspond exactly to an honest execution with a valid Vote Cast Return Code, and should therefore find a match in the `CMTable` at Step 3 of Section 12.2.2.3.

8. *The Voting Server then sends the (correct)* $\mathtt{VCC}_{\mathtt{id}}$ *value back to the colluding voting client out-of-band.*

Thus the voter submitted his $\texttt{BCK}_{\texttt{id}}$ and received a final confirmation with the correct code.

However, such logs could also appear in a scenario like the following one, in which a dishonest $\texttt{CCR}_4$ colludes with a dishonest voting client, while the VS is honest.

1. *The client modifies the vote choices made by the voter and submits an incorrect ballot to the Voting Server. The CCRs compute the corresponding choice return codes, which the voter rejects since they do not match her choices.*

2. The voter does not enter her Ballot Casting key $\texttt{BCK}_{\texttt{id}}$. *The client guesses a* $\texttt{BCK}_{\texttt{id}}$ *value, computes the corresponding* $\texttt{CK}_{\texttt{id}}$ *and sends it to the Voting Server.*

3. The Voting Server honestly forwards $\texttt{CK}_{\texttt{id}}$ to all the CCRs.

4. The honest CCRs ($j = 1, 2, 3$) perform all the steps of Section 12.2.2.2 of the protocol specification correctly, including logging, and return $\texttt{lVCC}_{\texttt{id},j}$ ($j = 1, 2, 3$) to the Voting Server.

5. *Cheating* $\texttt{CCR}_4$ *does nothing, and returns no value to the Voting server.*

6. The Voting server makes whatever logs are specified when it receives only three responses ($j = 1, 2, 3$). (This is currently not explicitly specified in Section 12.2.2.3.) It also returns no Vote Cast Return code to the voter.

Thus the voter never entered her $\texttt{BCK}_{\texttt{id}}$ and received no Vote Cast Return code. (These logs could of course also be the result of other scenarios – we are just describing two examples that result from opposite voter actions and views.)

Message reordering Now suppose the CCR logs show the same (two) confirmation attempts, but in a different order, so their logs look like:

$$\begin{array}{ll} \texttt{CCR}_j: & (\texttt{vc}_{\texttt{id}}, 1, \texttt{CK}_{\texttt{id}}, *, *), (\texttt{vc}_{\texttt{id}}, 2, \texttt{CK2}_{\texttt{id}}, *, *) \text{ for } j = 1, 2, 3. \\ \texttt{CCR}_4: & (\texttt{vc}_{\texttt{id}}, 1, \texttt{CK2}_{\texttt{id}}, *, *), (\texttt{vc}_{\texttt{id}}, 2, \texttt{CK}_{\texttt{id}}, *, *) \end{array}$$

These logs could be the result of various scenarios very similar to the previous ones. For instance, it may be the case that we have an honest voting client, that the voter entered a correct $\texttt{BCK}_{\texttt{id}}$ value, resulting in a correct $\texttt{CK}_{\texttt{id}}$ being sent to the voting server, but that the malicious voting server created $\texttt{CK2}_{\texttt{id}}$ as well and sent the values $\texttt{CK}_{\texttt{id}}$ and $\texttt{CK2}_{\texttt{id}}$ to the first three CCRs, and the values $\texttt{CK2}_{\texttt{id}}$ and $\texttt{CK}_{\texttt{id}}$ to $\texttt{CCR}_4$. The corrupted voting server may then decide to send the correct Vote Cast Return code to the voting client, after reordering the responses from $\texttt{CCR}_4$. The voter would then have a complete voting session. In another scenario, the voting server would not send the correct Vote Cast Return code to the voter. In yet another scenario, the voting client is corrupted, and both $\texttt{CK}_{\texttt{id}}$ and $\texttt{CK2}_{\texttt{id}}$ are incorrect values.

Divergence Now suppose all the CCRs show two confirmation attempts, but all with different values, so their logs look like:

$$\begin{aligned}
\mathtt{CCR}_1: &\ (\mathtt{vc}_{\mathtt{id}}, 1, \mathtt{CK1}_{\mathtt{id}}, *, *), (\mathtt{vc}_{\mathtt{id}}, 2, \mathtt{CK2}_{\mathtt{id}}, *, *) \\
\mathtt{CCR}_2: &\ (\mathtt{vc}_{\mathtt{id}}, 1, \mathtt{CK3}_{\mathtt{id}}, *, *), (\mathtt{vc}_{\mathtt{id}}, 2, \mathtt{CK4}_{\mathtt{id}}, *, *) \\
\mathtt{CCR}_3: &\ (\mathtt{vc}_{\mathtt{id}}, 1, \mathtt{CK5}_{\mathtt{id}}, *, *), (\mathtt{vc}_{\mathtt{id}}, 2, \mathtt{CK6}_{\mathtt{id}}, *, *) \\
\mathtt{CCR}_4: &\ (\mathtt{vc}_{\mathtt{id}}, 1, \mathtt{CK7}_{\mathtt{id}}, *, *), (\mathtt{vc}_{\mathtt{id}}, 2, \mathtt{CK8}_{\mathtt{id}}, *, *)
\end{aligned}$$

These logs could be the result of a malicious voting server who sent random $\mathtt{CK}_{\mathtt{id}}$ values to the CCRs – and this could happen whether or not the voter entered his correct $\mathtt{BCK}_{\mathtt{id}}$. Alternatively, they could be the result of an honest voter entering his correct $\mathtt{BCK}_{\mathtt{id}}$ on a second attempt, resulting in the submission of $\mathtt{CK1}_{\mathtt{id}}$ and $\mathtt{CK2}_{\mathtt{id}}$ to all the CCRs, and then of incorrect behavior by $\mathtt{CCR}_2$, $\mathtt{CCR}_3$ and $\mathtt{CCR}_4$, which would log random $\mathtt{CK}_{\mathtt{id}}$ values and may or may not compute and return the correct lVCC codes to the voting server.

Discussion In all three cases, there is no appropriate consistent information from any single attempt to extract a valid Vote Cast Code. Also, it is not possible to decide, just from these logs, what went wrong: these transcripts could be the result of an innocent communication problem, of a corrupted VS, or of the corruption of one or more CCRs.

In the message reordering case, the logs offer sufficient information to verify whether the correct $\mathtt{CK}_{\mathtt{id}}$ value is in the list, based on the $\mathtt{lVCC}_{\mathtt{id}}$ values from the logs and on the CMtable. In the other two cases, the logs offer no way to decide whether the correct $\mathtt{CK}_{\mathtt{id}}$ is in the list, and whether the voter ever submitted his $\mathtt{BCK}_{\mathtt{id}}$ value.

We are not certain exactly how an honest VS would deal with this situation, but we did not find anything that suggests it should be alert to this possibility and insist on obtaining a response (in case of omission) or rearrange the responses (in case of reordering). It is also unclear whether a $\mathtt{VCC}_{\mathtt{id}}$ would be returned to the voter in any of these cases.

A.4.2 What do the specification documents say about these cases?

Protocol Specification The scenarios described above describe some inconsistencies between the logs of different CCRs for the vote confirmation phase. At present, in version 0.9.11 of the protocol specification documents, the consistency checks described in the VerifyVotingPhase algorithm (Sec. 12.2.3), which decide whether votes are tallied, are only incompletely specified—it is not clear whether the proposed scenarios would pass or not.

Step 5 of the verification of the CCR logs indicates: "Check the equality of $\mathtt{vc}_{\mathtt{id}}$ and confirmation attempts number in $\{\mathsf{L}_{\mathtt{confirmed}_j}\}_{j=1}^{m}$. Our understanding is that the "Omission" case would fail on this criterion, but that the "Message reordering" and the "Divergence" cases would pass, since all the CCRs have 2 attempt for $\mathtt{vc}_{\mathtt{id}}$.

The presence of extractable short Vote Cast Return Codes is also verified. Here, we expect that the "Divergence" case would fail because of the absence of

$\texttt{lVCC}_{\texttt{id},j}$ tuples in the CCR logs that make it possible to extract a return code from CMtable. The case of the "Message reordering" is less clear: VS could have marked the ballot as extractable, and the right $\texttt{lVCC}_{\texttt{id},j}$ values will be found in the CCR logs, even though they won't correspond to the same attempt: even though we do not find any suggestion that a honest VS would try to reorder values coming from the CCR in order to see if they lead to an extractable code (and hence would mark the ballot as non-extractable), the VS is not trusted to follow the protocol specification and could mark the ballot as extractable. Besides, the verification process does not seem to require that the right $\texttt{lVCC}_{\texttt{id},j}$ values must come from identical attempt numbers in the CCR logs: this could make this ballot pass verification.

Protocol Specification, again Much later, in Section 16.2 of the protocol specification document, there is an indication that auditors who find an inconsistency could start interacting with other system components, perform an analysis, which could result in a modification of the voting server and control component's state and in the list of ballots to be included in the tally.

There is no specification of this state modification process specified there, however. So, based on the protocol specification, we do not know if this ballot would be marked as valid and ready to be counted or not, and many approaches would be possible: we outline some of them below.

One simple approach would be to discard the ballot, because of the failure of the verification process, and because a VS that behaves according to the protocol specification would not have sent any $\texttt{VCC}_{\texttt{id}}$ to the voter – so the voter would not have received any proof that her ballot was recorded. However, this approach is problematic, because, as discussed above, the logs are also compatible with protocol sessions during which the voter would have seen the correct $\texttt{VCC}_{\texttt{id}}$, which would be proof that the ballot was recorded for tally.

Another approach would accept the ballot in the "message reordering" case, because the logs contain all the information that is needed to verify that the voter actually submitted her $\texttt{BCK}_{\texttt{id}}$ (if the reordering is detected), and to reject the other two cases because, from the logs only, it is not possible to determine that the voter submitted her $\texttt{BCK}_{\texttt{id}}$.

In a third approach, a more in-depth investigation would be organized regarding the "omission" and "divergence" cases, in order to determine if the logs offer any compelling evidence that the voter submitted her $\texttt{BCK}_{\texttt{id}}$. This would require further interaction with some system components, in order to obtain missing elements.

In any case, if the inclusion of the ballot is decided in any of the 3 cases, it appears that the logs and states of various system components would need to be modified, and we do not know how such a sensitive process would happen, as it is not part of the protocol specification.

Besides, such a decision seems to be incompatible with the OEV: any such action would require a communication originating from the auditors (or auditors' technical aids) towards some system components, which is not allowed according

to Art 2.2 of the OEV Appendix.

How are these questions handled in the security proof? The relevant section is in 16.2, where Theorem 3 formalises the idea that a voter should not receive a valid Vote Cast Return code for a vote that is not included.

The security proof does not properly cover cases like this because

- The description on page 102 does not line up the explanation of Art. 2.5 of the ordinance—the cases and responses need to be handled,
- The game in Figure 30 does not model multiple code submission attempts,
- The game is hugely reliant on data consistency, for example it is not clear that the adversary is allowed to choose to omit some confirmations entirely and produce something like the Omission transcript.
- It does not seem possible in the game for the adversary to block the honest voter's transmission of $\texttt{CK}_{\texttt{id}}$ to the honest $\texttt{CCR}$, but a real adversary who controlled a corrupt VS could easily do so.Step 11 is hard to understand, because the definition of ConfirmVote does not output a tuple, so it is not clear exactly what the adversary can control.
- The proof tries to cover attacks that the game doesn't (like allowing multiple attempts), but this says very little as long as the security properties are not fully defined.
- Figure 31 has similar issues.

Verifier Specification The verifier specification (version 0.9.1) is more demanding, and it appears from Section 4.1 that none of the inconsistencies that we propose would pass verification: verification step 2.43 requires strict equality accross control components of the $\texttt{hCK}_{\texttt{id}}, \texttt{attempts}_{\texttt{id}}, \texttt{vc}_{\texttt{id}}$ values. This would in particular imply that the "Message reordering" case, which may have passed the previous verification steps, would still result in a verification failure.

Contrary to what appears in Section 16.2 of the protocol specification document, the verifier specification just concludes with a failure, and there is no suggestion that any log reconciliation attempt should be made.

A.4.3 What should be done?

The OEV has several articles related to this phase of the protocol. Art 2.5 requires, "As a condition for the successful examination of the proof referred to in Number 2.6, all control components must have recorded the same votes as having been cast in conformity with the system. Cases where the control components show inconsistencies in this respect must be anticipated in accordance with Number 11.11 and the procedure determined in advance."

Articles 5, 6 and 8 of the OEV indicate that any voter who receives a proof that his vote has been registered should have his vote counted as long as at least one control component from each group is honest.

Option 1: inconsistent logs cause complete verification failure One way of dealing with these problems is to enforce the strong verification failure response described in the Verification spec: complete election verification failure in the case of any inconsistencies.

It was unclear to us whether this approach would be consistent with the OEV. Arguably complete verification failure is a "procedure determined in advance" and arguably there is an implicit expectation that the guarantees of Art 5,6 and 8 are allowed to fail when verification fails. (Though this could probably be specified more clearly in the OEV, if it is intended.) Either way, this seems to be a very strong reaction to inconsistencies that may easily appear due to normal network issues, particularly given that the CCRs are supposed to be under the control of different organisations.

Option 2: accept inconsistent logs as long as at least one `CCR` has the correct $\texttt{CK}_{\texttt{id}}$. We observe that, as soon as one of the CCRs has the correct $\texttt{CK}_{\texttt{id}}$ value logged, then it may be the case that the voter saw his final $\texttt{VCC}_{\texttt{id}}$ code: it is compatible with the security model that the one CCR that shows the correct $\texttt{CK}_{\texttt{id}}$ provided its decryption factor, and that the other CCRs did the same without logging anything, while the VS would have returned the correct $\texttt{VCC}_{\texttt{id}}$ to the voter, without logging anything either, and without marking the ballot as cast.

As a result, Art 5,6 and 8 imply that, if one of the CCRs has the correct $\texttt{CK}_{\texttt{id}}$ value logged, the corresponding ballot must marked for inclusion in the tally. (This discussion is only about the ballot confirmation step, and abstracts from what may happen in previous protocol steps. A different conclusion may be reached depending on other parts of the logs – see discussion below.)

The question is how to change the protocol to achieve this. This prompts for non negligible protocol changes.

Should the auditors get involved? The current process by which auditors could possibly lead to this result remains unspecified. We believe that these cases, being compatible with the security model of the OEV, should be part of the protocol specification. Otherwise, it is impossible to assert that the decisions that would be made by the auditors would be compatible with the OEV.

Furthermore, the task that is currently proposed to the auditors raises non-negligible practical challenges. In particular, for the "omission" and "divergence" cases, the logs do not contain enough information to decide if the correct $\texttt{CK}_{\texttt{id}}$ value is there. This issue could be solved by requesting more $\texttt{lVCC}_{\texttt{id}}$ values from CCRs that could be compromised and may refuse to respond – or would just refuse to respond because this is what the protocol specification currently requires them to do. Support from the set-up component might also be requested, but this raises its own security challenges.

Furthermore, this process should not be the responsibility of the auditors: the task of the auditors should not be to accomplish fundamental protocol steps, and it seems that this is incompatible with the OEV anyway. Other system

components must be in charge here.

Adding an explicit agreement protocol or bulletin board It appears that the fundamental source of this problem is *agreement* on the input data between the CCRs, vote server and auditors. In the earliest versions of this protocol, the voting server's role was taken by a Bulletin Board, which represents a completely different set of trust assumptions, specifically the impossibility of showing different views of history to different participants. Substituting an untrusted Voting Server left the protocol without the right grounding assumptions to make it secure.

The right way to address these issues would be to (re-)introduce an explicit notion of a mechanism for all the CCRs to agree on the values of $\texttt{CK}_{\texttt{id}}$. There are several ways to think about this, which may not actually be very different in practice:

- a direct agreement protocol executed by the CCRs;
- an optimistic behavior by the CCRs during the voting phase, in which they expect agreement, followed by a verification step by some other components (perhaps the CCMs) in order to check that agreement had been achieved, with a recovery protocol being executed if it is not;
- a bulletin board, which would implicitly involve some form of agreement under some assumptions.

It is important to consider the security assumptions under which this protocol or component would be trustworthy. There is a vast literature on distributed agreement—Byzantine protocols (such as [6]) are secure only against at most $\lfloor \frac{n-1}{3} \rfloor$ faulty participants out of n in total. However, the assumption of digital signatures should allow you to use protocols secure only against stopping failures, for which arbitrary numbers of misbehaving parties can be tolerated.

After an explicit agreement step, it would make more sense to impose a strict policy of verification failure in the event of any inconsistency, i.e. Option 1.

Changes in the security definitions and security theorems The current security definitions assume that there is no inconsistency between the various logs. As such they do not capture various system behaviors that can occur within the security model defined in the OEV, and the security theorems in the protocol specification documents say nothing about such cases.

It seems however fairly clear that, if it is decided that a ballot must be marked for inclusion in the tally when only one of the CCRs has the correct $\texttt{CK}_{\texttt{id}}$ value logged, then the current security claims are incorrect.

As an example, let us consider a vote injection attack, in the setting of Section 16.2.2 of the Protocol specification, but with the amended ballot inclusion rule. The attack follows a scenario very similar to the "Divergence" case discussed above and is as follows:

1. A malicious voting client modifies the choices made by a voter, resulting in incorrect choice return codes being returned.

2. The voter is cautious, notices the incorrect choice return codes, and does not type his $\texttt{BCK}_{\texttt{id}}$.

3. The corrupted voting client then sends, using an off-band communication channel, the $k_{\texttt{id}}$ of the voter to a malicious voting server.

4. The corrupted voting server makes $4 \cdot \mathsf{maxConfAttempts}$ guesses on the value of $\texttt{BCK}_{\texttt{id}}$, computes the corresponding $\texttt{CK}_{\texttt{id}}$, and sends $\mathsf{maxConfAttempts}$ distinct guessed $\texttt{BCK}_{\texttt{id}}$ values to each of the CCRs. Each CCR will then include $\mathsf{maxConfAttempts}$ values of the form $(\texttt{vc}_{\texttt{id}}, *, \texttt{CK}_{\texttt{id}}, *, *)$ in their logs.

5. The inconsistency between the logs is detected, and the ballot is accepted if the correct $\texttt{CK}_{\texttt{id}}$ appears in any of the CCR's logs.

As a result of this process, the ballot is accepted with a probability at least $4 \cdot \mathsf{maxConfAttempts}/|\mathcal{C}_{bck}|$.

This attack success probability is expected to be around 4 times as big as the maximum success probability of a vote injection attack according to Theorem 3, which only offers a factor $\mathsf{maxConfAttempts}/|\mathcal{C}_{bck}|$ only.

Of course, increasing this probability by a factor 4 may remain low enough. But it remains substantially higher than what the security analysis of the protocol claims.

A.4.4 Discussion

We believe that the issues presented above have always been present at this stage of the protocol, and that SwissPost's improved description of Scytl's protocol has brought them to light rather than introducing them.

We also suspect that similar issues may occur at other steps of the protocol. For instance, inconsistencies may also happen in the process of computing the partial Choice Return Codes, earlier in the protocol. While the logs may be inconsistent, a voter may or may not have seen the correct Choice Return Codes. In both cases, the voter may have entered his $\texttt{BCK}_{\texttt{id}}$ (this could also happen with one incorrect Choice Return Code that the voter would not notice), and it may then be tricky to decide what to do if the correct $\texttt{CK}_{\texttt{id}}$ appears in one of the CCR's logs.

As a result, we do not make any claim that it is sufficient to follow the policy discussed above, of including a ballot in the tally as soon as only one of the CCRs has the correct $\texttt{CK}_{\texttt{id}}$ value logged. This choice may actually be incorrect in some cases as well, depending on what happened at other steps in the protocol.

Indeed, we also do not have a proof that either the strict option (Option 1) or the explicit-agreement (Option 2) are secure—these are simply suggested directions to investigate.

We believe that important updates are needed in the protocol specification, security definitions, and security theorems, in order to address these issues, and offer evidence that they are solved.

A.5 Issues with signature verification

This section of our report refers to a vulnerability disclosed to Swiss Post in March 2021 prior to the current review process starting. We include it here for completeness since some of our other findings depend on this vulnerability. We also include it because the underlying vulnerability is still not patched.

When verifying signatures the Swiss Post Voting system[6] failed to check that the signatures came from the party it expected to be corresponding with. This potentially allowed attacks on integrity by spoofing the input of honest parties. These attacks could be caught by the verifier, but since the relevant parts of the verifier were not published at the point the bug was submitted (March 2021), it was not possible to verify this. Swiss Post has now confirmed how they intend to resolve this issue and pending some slight updates to the documentation and code, the known attacks from this vulnerability should be fixed. We expect that Swiss Post will release a Gitlab issue about this vulnerability but this has not yet occurred.

A.5.1 Key recommendations

Check identity The signature verification should check that the corresponding party is correct. This could be done by checking that the X.509 certificate's subject field contains the expected name.

Check key usage All certificates in the chain should be checked to verify that they are being used for a valid purpose (using the attributes provided in RFC 5280).

Secure initialisation It is crucially important that the root certificates are correctly loaded. The documentation should clearly describe how this is accomplished.

A.5.2 Details

This section of the report describes the problem as it existed in March of 2021. The current public version includes several improvements which partially address this issue; Swiss Post has confirmed they intended to update the documentation to completely address the attacks raised.

Many of the authentication checks in the system verify that the input is signed but not who it is signed by. Since the adversary has valid signing keys

[6] This vulnerability was detected in version 0.7

it can then impersonate honest parties. Examples appear to include validateChoiceCodesEncryptionKey in VotingCardSetDataGeneratorServiceImpl and validateSignature in ChoiceCodesGenerationServiceImpl.

For example, this could allow the adversary to impersonate the one honest return code control component starting in the config phase and running undetected until the logs of the control components are examined in 12.2.3 VerifyVotingPhase.

The key issue here is that the system, when verifying signatures, does not check that the attached X.509 certificate's subject field matches the expected party or that the keys are being used for a purpose which the signer of the key's certificate intended. No check has been found which prevents the control components from impersonating the one honest control component. This would allow the one honest control component to be bypassed, which breaks cast-as-intended verification; the setup component would honestly combine the shares of the return codes but all the shares would be coming from the adversary.

No audit of the config phase described in the computational proof or system specification, at the time this issue was reported, would catch this attack on cast-as-intended. Nor was the verifier for the config phase in the repository. However, it was an open question if the attack (or a similar attack) would go undetected by the verifier specification and implementation that were (and to a significant extent are) unreleased and under development.

In conclusion, the identified vulnerability did appear to lead to manipulation that goes undetected by the voter but not by the system based on the then released material. However, the attack was caught by then unreleased checks.

A.5.3 Resolution

Swiss Post has prevented the attack detailed in this report by a manual process which checks that the certificates used in the verification are the correct certificates. This certainly prevents the specific attack detailed in this report. More details on the resolutions should appear soon when Swiss Post posts an issue on their Gitlab repo related to this finding.

Summary

The underlying vulnerabilities described here are still present in the SignatureChecker class in the verifier and the various signature verification implementations in the voting system. While there are no currently known attacks which exploit the vulnerabilities, we nevertheless strongly encourage Swiss Post to patch the underlying vulnerabilities by implementing the key recommendations of this report.

Future versions of the Swiss Post Voting system aiming for higher levels of assurance may wish to dispense with certificate chains entirely and load all certificates through a manual process; this would eliminate the need to trust any root certificate authority.

A.6 Status of issues raised in our draft report

A.6.1 Crypto primitives of the Swiss Post Voting system

Algorithm 3.1 Swiss Post has implemented our suggestion to truncate the random string before testing if the resulting integer is in the desired range. This makes the system more efficient.

Algorithm 3.6 Swiss Post has implemented additional domain separation to prevent the issue we described. We have not carefully reviewed their revised version.

Section 4.1 We commented that the default security level was relatively low and recommended they adopt the extended security level. We do not appear to have received a response.

Algorithm 4.1 We noted that algorithm required to p to be a multiple of 8. Swiss Post has updated the document to reflect this.

Algorithm 5.6 We noted the generation of group generators did not fit the standard discrete log assumption. Swiss Post has now rectified the issue.

Typos The typos we raised have been fixed.

A.6.2 Protocol of the Swiss Post Voting System

4.11 Election public key In our previous report we expressed our concern about how the public key of the election is transmitted to the CCRs and detailed a possible attack. We were especially concerned because Thomas Haines had notified Swiss Post in March that they were not authenticating communication properly, see section A.5 for details.

Swiss Post has confirmed that we had indeed identified an attack on individual verifiability which would not have been caught until invalid votes were detected in the output of the final decryption. Their gitlab issue on the attack can be found at `https://gitlab.com/swisspost-evoting/e-voting/e-voting/-/issues/1`.

Production of verification data In our previous report we noted that while the production of data for verification seemed adequate it was not clear that the data was being appropriately verified; this remains the case.

Ambiguous and unaligned specification In our previous report we noted differences between the "Protocol of the Swiss Posting Voting System" document and the "Swiss Post Voting System: System specification." The specific examples we raised have now be rectified but several key differences still remain as detailed elsewhere in this report. There are also differences between the System Specification (which includes some verification) and the Verification Specification.

Problems with verifiability specification The specific issue we raised have been addressed but problems still exist as detailed elsewhere in this report.

Problems with security definitions We highlighted in our previous report that the security definitions did not capture the properties required. These issues have not been resolved.

Section 2.1.1 We noted that the electoral board was not adequately described in the document. This has improved somewhat but issues remain as detailed else where in this document.

Function H Our recommendation to define the function H has not been implemented.

KDF Our recommendations on both the definition of KDF and the choice of KDF have been implemented.

Section 7 Most of our recommendations on section 7 have not been implemented; several of the definitions remain incorrect.

DDH We highlighted that DDH was defined incorrectly which Swiss Post has now fixed. An analogous issue exists in the definition of ESGSP which has not been fixed.

Groups and generators Our recommendation on verifying the group parameters as part of the VerifyConfigPhase protocol has been implemented though this aspects of VerifyConfigPhase protocol are found only the verification specification and not in the proof document.

Product of primes Our recommendation on verifying the product of primes has been implemented.

Logs We recommended the way in which logs are created, stored and verified be detailed. Some additional information has appeared in the verification specification but more details are still required as described elsewhere in this report.

Section 12.2.1.3 We noted that the way in which the election public key is transmitted was different between different documents. This has now been rectified.

Section 12.2.1.7 In our previous report we asked how the system enforces that the ballots contained a valid selection of answers. Swiss Post has now explained that is accomplished using the allow list. We encourage them to update the documentation to make this clearer.

Section 15.1 In our previous report we noted that the definition of IdealSetupVoting was unclear. The definition is mostly hinted at in the caption of Figure 25 but this leaves several variables undefined. This has not been rectified.

Section 15.1 We had complained that the claims about what Lemma 1 proved were not correct. Swiss Post has not removed the claims in question.

Section 15.1 We noted that since the public election parameters were not checked Lemma 1 does not hold since the there is no guarantee to have a SGSP instance. This issue has not been addressed in the proof document though the relevant checks are listed in the verifier document.

Section 16.2.1 We noted that the game for ballot rejection did not appear to align with either the system or the required security properties. Swiss Post appears to have made an effort to address but the problem is still present.

Figure 33 We noted that Bpriv was never designed to be used as a standalone definition but in conjunction with strong-consistency. We recommend Swiss Post either prove the system has strong-consistency or explain why proving strong-consistency is not necessary, since this is a significant departure from the literature. Neither has been done.

Ballot privacy game We made many comments on figure 34. Several of our comments have been implemented but several also remain unresolved. At present the game still does not capture the property of privacy as described the ordinance.

A.7 Status of Gitlab issues we opened

We briefly discuss here the status of the various Gitlab issues we have been been involved in including those opened before the review process. Note, we have taken the issue numbers from Gitlab and hence the list below is not consecutive.

Documentation Issue 1 Discussion about responsible disclosure definition
`https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/issues/1`

To our knowledge no changes have been made as a result of the discussion of the issue. We remain concerned that the restrictions on disclosure, particularly when the finding is discovered close to an election, might not be the public's interest.

Documentation Issue 2 The algorithm GenCMTable allows an adversary to recover the election event's set of possible short return codes
`https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/issues/2`

The issue raised has now been patched. The fact that the issue was not immediately obvious but hidden by the notation is frightening. We think this issue is a great example of the care that must be taken in analysing a system of this complexity and the paramount importance of clarity in the documentation.

Documentation Issue 3 Various minor issues and potential issues
`https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/issues/3`

Many of the issues raised in this report have been resolved. However, there are notable exceptions:

- there are still mistakes in the cryptographic definitions;
- the way in which parties authenticate to each other is still problematic;
- the alignment between the protocol and the ordinance is still problematic.

The current status of these issues is discussed at various places through out this report and I will not repeat them herre.

Code Issue 1 Insufficient Signature Validation of the Election Public Key resulting in possible attacks against individual verifiability
`https://gitlab.com/swisspost-evoting/e-voting/e-voting/-/issues/1`

We discuss the issue in the first dot point of 1.1. The attack is still possible on the protocol as described in the current documents but this should be rectified in the new versions.

Verification Issue 1 `https://gitlab.com/swisspost-evoting/verifier/verifier/-/issues/1` This issue is confidential at the time of writing, but visible to some other users of the SwissPost gitlab.

B August Report

B.1 Introduction

We have, in a limited time, attempted to read and consider all the documents available for Scope 1 of the review of the Swiss Post e-voting system and the Swiss Federal Chancellery's updated requirements, namely:

- The Draft of April 28, 2021, of the Federal Chancellery Ordinance on Electronic Voting (OEV);
- The Cryptographic Primitives of the Swiss Post Voting System, Version 0.9.5;
- The Protocol of the Swiss Post Voting System, Version 0.9.10.

In some cases, we also looked for clarifications in:

- The System Specification of the Swiss Post Voting System, Version 0.9.6.

This draft should not be considered a complete review - we have read all the documents, but there could be subtle interactions that we have missed. It should, instead, be considered as a list of suggestions for improvement, which we hope to refine as we see improved documents. In some cases, we think we see an error; in others, we simply did not understand, and seek further clarification for a followup draft.

This report is arranged according to the documents it analyses. We first consider the federal chancellery's new ordinance draft, then the cryptographic primitives, and finally the protocol and its security proofs.

Summary The description of the protocol is enormously improved from prior versions. We would like to thank Swiss Post for the much-improved readability. This makes it much easier to understand and examine the underlying logic, something that we have struggled to do for earlier drafts.

This improved clarity does make some gaps evident which have probably always been there, but are now much more obvious because most of the rest is clear. This is truly an improvement, and we hope the attached observations can help improve it further.

Our (preliminary) conclusions on the cryptographic protocol documentation are as follows:

- The current protocol does a good job at preventing the attacks that were previously proposed. However, it is quite visible that, in many places, the protocol is the result of a long evolution and has become an addition of patches. For instance, while reading the protocol description, we wondered why the $p_i^{k_{id}}$ are ElGamal encrypted with the CCR public key, while the $\mathtt{BCK_{id}}^{2 \cdot \mathtt{k_{id}}}$ term is sent in clear to the CCR's? We could not find any explanation in the cryptographic protocol description document,

but then found a discussion in Section 5.1.4 of the protocol specification, which explains that this layer of encryption is indeed not needed. We can however imagine that it adds a relatively significant amount of latency on the server side.
We would strongly encourage to consider the design of a cleaned, minimal, version of the current protocol, and to add an extra section in the cryptographic protocol document, that would offer a clear view of the role of each cryptographic operation (and what would break in the protocol if each of these operations were missing). We suggest simplifications of the protocol in several places in the document, but it is clear that much more can be done, with expected benefits in security and in efficiency.

- The security model adopted in the system does not always seem to be aligned with the OEV requirements. In particular, the structure and the role of the electoral board is presented in different ways in the document threat model, in the protocol specification, and in the security proofs. It does not seem to comply with the OEV in the last two cases.

- In a similar spirit, it is currently extremely difficult to match the security definitions with the OEV security requirements. It would be very helpful to add a few paragraphs explaining what each step of the security games is doing, and why the security definitions offer the necessary guarantees. We highlight several places where we could not understand why the current security definitions would offer the desired guarantees. Especially with privacy it seems clear that definition does not demonstrate that the cryptographic protocol meets the requirements, as required by 2.14.1 of the technical requirements.

- The security proofs remain very high level, and it is extremely hard to be convinced that they are correct. In particular, there is essentially no reduction offered between any two consecutive game hops, but only claims that they must be indistinguishable based on some underlying assumption. Quite often, it is not obvious that the claims are justified and, without writing all the missing proof details, we observed several places where they simply seem to be incorrect. In some cases, it appears that attacks against the system could work – this is only based on the cryptographic protocol specification, and we do not know if they would work on the real system. Writing the actual security reductions that justify why two games are indistinguishable would make the document much more convincing.

- A full detailed description of the verification protocol is a crucial part of verification soundness - without it, we cannot make any concrete analysis of either individual or universal verifiability. We have identified some gaps in the proof in each case, but we hope to make a more complete assessment when the full details will be available.

B.2 Observations regarding the OEV, Draft of 28 April 2021

The new draft ordinance document clarifies many of the requirements that were present in the previous version, which is a very positive thing. As a downside, we feel concerned that it also matches more and more closely what a properly implemented version of the Swiss Post protocol should be doing. Of course, there is a low-level design space that is quite open and that other system providers could explore. But it also does nothing to offer incentives to the design of *stronger* systems. As a result, it does not offer any incentive for a system supplier (Swiss Post, or any other one) to explore the possibilities of offering something more secure than the minimum that the OEV requires.

Would it make sense to refine the requirements by identifying various grades at which security properties are realized by a system? These could be identified from the properties discussed in the dialog of 2020, so that a system would not just be declared to be compliant, but would also receive a grade, so that cantons would have the possibility to choose a system that obtains a higher grade?

Here are some possibilities that come to our mind in particular:

- The notions of individual and universal verifiability that are required here are much weaker than those that are standard in the academic literature (the standard notions in the literature are incompatible with the assumption of a trusted print office, or of control components among which one must be honest). Would it make sense to offer a higher grade to a system that would remove some or all of the current trust assumptions?

- Articles 2.9 and 2.10 of the annex lists participants and channels that *may* be considered trustworthy. It would be interesting to value systems that would not need to consider as trustworthy some or all of these components. (We would also query the terminology, though this is possibly a matter of translation—we would generally have said "trusted," which is a protocol property, while "trustworthy" is an inherent property of a particular instance. Probably "trusted" could be a shorthand for "assumed by the protocol to be trustworthy.")

- The print component is currently a central trusted component. Exploring protocols that would reduce the requirements on this system component should be valued.

Here are some more specific remarks:

- We are not sure whether the OEV states anything about the handling of a failure event in the cryptographic protocol (say, a system component observes that a ZK proof produced by an other control component does not verify) – the OEV seems to rather focus on hardware failures, which are not necessarily a sign of malfeasance. Intuitively, it would seem that a control component cryptographic failure should be taken very seriously, and the offending control component should probably be removed,

whereas a cheating client is expected and should not necessarily result in strong action. However, the requirements do not currently require (and the protocol does not provide) accountability, so it may not be possible to identify the source of the problem. As a result, and in the absence of any accountability or robustness requirement, and of a description of what should happen in case of failure in the protocol specification, it appears that the voting system should just be halted. But this is also opening an important channel for denial of service attacks, and the impact of such a decision would probably be so high that it may not happen even if it would be technically sound.
Would it make sense to add requirements of accountability and robustness in the OEV?

- We are concerned that requirement 2.7.3 of the OEV might just be unrealistic (or at least conflicting with the usability requirements that are stated elsewhere). And, indeed, we did not find anything in the protocol documentation that would help satisfying requirement 2.7.3. If a voting client simply gets the voting application from the untrusted voting server, then the voting server can send a voting application that will encrypt the votes with incorrect keys (or encrypt with the correct keys and send the votes in clear in parallel to the honest protocol), ... One could imagine to ask the voters to download a local copy of the voting application, hash it, and compare that hash with something printed on the voting card. But this is obviously not usable for a regular voter. And any check performed in the voting application itself could be bypassed: the server can offer an application that will claim to be authentic. An alternative would be to have a signed stand-alone voting application rather than a web application, but this also raises practical difficulties. So, while our opinion is that the Swiss Post system does not satisfy Requirement 2.7.3, we also do not have any reasonable suggestion to make on how that requirement could be satisfied.
  Would it make sense to revise this requirement in the OEV?

B.3 Review of "Cryptographic Primitives of the Swiss Post Voting System" (Version 0.9.5)

We follow the order of the document. Next to each item, we provide a label:
– *Improvement* – indicates that we do not believe that the comment is related to something critical in the system. Addressing such items would typically make the system more efficient or more robust.
– *Recommendation* – indicates that the comment (or the question) is related to something that may be critical in the system and should be addressed.

Algorithm 3.1 – *Improvement* – It appears that this algorithm may be very inefficient when m is of the form $2^{8i} + s$ where s is small. For instance, when $s = 0$ it may be expected to loop around 256 times before it gets a

suitable r (this may happen quite often in `GenPermutation`, for instance). It would be more efficient to truncate r to the bit length of $m-1$ so that it would take on average 1.5 iterations of the algorithm before finding a suitable r.

An alternative, that would avoid the need of a loop, would be to pick r as a random integer of byte length $\mathsf{byteLength}(m-1)+10$ (where 10 is a security parameter), and then set $r \leftarrow r \bmod m$. The resulting distribution would be almost perfectly uniform (in the sense of statistical distance, and provided that the random bytes are uniform).

Algorithm 3.6 – *Recommendation* – It appears that this algorithm could lead to strings and integers that do not represent the same "thing" having the same hash: the IntegerToByteArray algorithm can produce any ByteArray, including some that would be the encoding of strings.

An alternative, used in other systems (e.g., in ElectionGuard), is to encode integers as strings, and then to only process strings. There certainly are other options, e.g., prepending all hashed elements with a typing indication, just as `0x` is used to indicate a hex representation.

This domain separation is discussed in Section 3.2 but does not seem to be correct.The property is used in the "Computational Proof," Definition 11, which states that H is "a hash function with domain separation, thus the list of variables is encoded uniquely into a binary string."

We do not see how this can be exploited in the present protocol, but it is also uncomfortable to need to care about this, and the document does not offer any suggestion that this has been checked, since the hash function is just assumed to be ideal and to offer domain separation.

Section 4.1 – *Improvement* – The *default* security level with $|p| = 2048$ is indeed accepted in some reports, but for relatively short term security, e.g., until 2030 for the two referenced reports. Given the importance of long-term security for votes, we would recommend to adopt the *extended* security level. By comparison, ElectionGuard adopted $|p| = 4096$ bits [12].

Algorithm 4.1 – *Recommendation* – It appears that this algorithm expects $|p|$ to be a multiple of 8. As other choices might be possible, at least for testing, it would make sense to make that requirement explicit.

Section 5 – *Recommendation* – There, and in other places: "Telerius" should be spelled as "Terelius".

Section 5.1.2 – *Improvement* – The algorithm that selects the matrix dimension is surprising. The main advantage of the Bayer-Groth mixnet against the Terelius-Wikström mixnet is that it leads to much more compact proofs (the algorithm is considerably more complicated otherwise, and we would advise to switch to the Terelius-Wikström mixnet if the size of the proofs is not a concern: the implementation would be much more

straightforward to review and maintain). But the Bayer-Groth mixnet offers most of its benefits when we can arrange the ciphertexts in a square, or nearly square, matrix (slightly different matrix formats may be beneficial depending on the size of the parameters p and q and optimization goals). However, the algorithm that is adopted here may lead to a matrix of any possible shape, including a single line matrix, as the document explains.

If, for your purpose, the Bayer-Groth mix-net works best with square matrices (as it is indicated in the document), why not extend the input list of ciphertexts with enough $(1, 1)$ ciphertexts (which are encryptions of "1" with randomness "0") to reach an optimal matrix format? In the worst case, for N ciphertexts, this would require adding $2\sqrt{N}$ extra ciphertexts in the mixing process, which seems to be a negligible amount of extra work when N is large enough for caring about the protocol execution time. It will then be immediate to remove the "1" plaintexts from the output. And this will make it always possible to obtain the benefits of the Bayer-Groth mix-net.

Algorithm 5.6 – *Recommendation* – Given the other parameters of the system, this algorithm seems to only generate 512-bit generators (squared 256 bit values). This does not fit the standard discrete logarithm assumption, which would pick uniformly random generators in $\mathbb{G}_q$. It would then be either necessary to introduce a new computational assumption stating that the DL problem is hard for the special generators that are selected here, in order to obtain a sound security proof (we do not recommend this approach), or to pick the generators uniformly (we recommend that approach).

One option would be to use a key derivation function, or the SHAKE128 algorithm that is used in Algorithm 4.1, in order to produce a uniformly random element in $\mathbb{Z}_p^*$ and to square it.

B.4 Review of "Protocol of the Swiss Post Voting System" (Version 0.9.10)

This document is vastly improved compared to the previous versions that we saw, which makes it much more useful, and also helps spotting possible shortcomings or suggestions for improvements.

We start with some general remarks, then review the document section by section.

B.5 General overview

Production of Verification Data: in general the system seems to be producing enough data to allow for the verification of election within the allowed trust model. (Caveats around authentication and data consistency apply). However, as we have already noted, if this data is not carefully verified attacks are possible.

Ambiguous and Unaligned Specification: It's really unclear what's happening with the spec. The system spec and the proof are too different protocols. Moreover, there is very poor alignment between the material released (spec, proof, etc) and the draft ordinance. Issues included differences in system participants, communication channels, security requirements, etc.
Example:

- Does the voting client send the data directly to the CCRs or via the voting server. Figure 21 in Section 11.2 in the proof document shows this occurring through the voter server whereas Figure 8 of the system specification has this happening directly.
- The ordinance would seem to suggest it must send it via the voting server since no channel exists between the control components and the user device
- The ordinance now refers to set-up component and print component while the documents still refer to print office. We assume that the spec should be explicit about the set-up component and print component, and split the (current) role of the print office in two parts.

It will not be possible to properly assess the compliance of the system with the ordinance until it is clear what the system does and how this is meant to align with the ordinance.

Problem with Verifiability Specification The current description of the verification is inadequate, as various verification steps are missing. For example:

- Public system parameters are not checked; this breaks privacy and integrity, as detailed below.
- The voting server can fudge much of the information that passes through it (Election key, user submissions); this breaks privacy and integrity. For example, by providing inconsistent views of the voters' confirmation key to the the CCRs.Improved verification and strong mechanisms to deliver configuration information would be needed.

Problems with Security Definitions The definitions in the proof document do not seem to capture the properties required. For instance, we explain below why we believe that the definition of privacy is too weak, and that the "ideal setup" does not offer the properties that it is claimed to offer, while this would seem to be crucial for individual verifiability.

B.6 Section 2

Section 2.1.1 – *Recommendation* – The parties include an Electoral Board, which is matched to one control component from Section 2.1 of the OEV Annex. However, in Section 12.1.2.2, it appears that the Electoral Board

is really one single party, that receives a secret key from the print office. Furthermore, it appears that this party is then also fully trusted (see, for instance, the definition of privacy, where at Step 4 of the bpriv game, it is indicated that $\mathtt{EB}_{\mathtt{sk}}$ is never known by the adversary. This appears to be in contradiction with Article 2.9.3 of the OEV Annex. In Section 13.2, it also appears that the electoral board is actually composed of multiple parties that keep shares of this secret key. Who computes these shares, and how are they distributed?
Our suggestion (see also comments on Section 12.3) would be to have the Electoral Board as a control component generating the Electoral Board key in a distributed way (just like the other control component do), and to leave all the decryption operations to that control component, while removing the decryption steps by the CCMs.

B.7 Section 3

Key compression – *Improvement* – The justification that is offered in Section 3.1 for the compression of excess keys is not very convincing. As far as we can see, the length of any ciphertext is imposed by the format of the election, and should be be part of the verification steps before decryption. So, the compression of excess keys does not seem particularly helpful. However, it makes the specification more complex.

B.8 Section 4

Symmetric encryption algorithm – *Recommendation* – There should be a specification of which symmetric encryption algorithm is used. The document that specifies the cryptographic primitives does not specify any symmetric encryption algorithm either.

B.9 Section 5

Function H – *Recommendation* – The function H with range $\mathbb{G}_q$ is undefined in this document, and in the "Cryptographic Primitives" document. It should be defined (and it would be useful in other places, including for selecting the generators of the Pedersen commitments).

If RecursiveHash is intended to be used here, its definition should be expanded to input elements of $\mathbb{G}_q$.

B.10 Section 6

Pseudorandom KDF – *Recommendation* – Definition 3 (Pseudorandom KDF) is much weaker than what is usually required from a KDF (e.g., [10]):

- It is normally expected that a KDF can be called many times (like a PRF), while the current definition only provides one single output (like a PRG).

- It is normally expected that the distribution of the inputs of the KDF can be correlated to adversarially chosen/known elements, but this is not the case here.

This creates a gap in the proofs. For instance, all the CCRs use the KDF via the DeriveKey function on a single $\mathtt{k'}_j$ value and multiple $\mathtt{vcd}_{\mathtt{id}}$ values (Section 12.1.1.5). It is immediate to design a KDF that would satisfy Definition 3 and would lead to a completely insecure system when it is used in this way (e.g., the definition is compatible with a KDF that would leak $\mathtt{k'}_j$).

KDF choice – *Recommendation* – The KDF function specified here is the MGF1 function. However, the cryptographic primitives document does not discuss this, and uses SHAKE128 for similar purposes in Algorithm 4.1. We would recommend to specify a single function that would be used as a KDF. As a third option, we would recommend to consider the HKDF function [10], which is designed to be particularly robust, has been adopted in TLS 1.3 for instance, and for which numerous well-reviewed implementations exist.

DeriveKey – *Improvement* – In the DeriveKey algorithm, why is the KDF function used to produce an output of byteLength p bytes, rather than of byteLength $q - 1$ bytes? It is also slightly surprising to see that the loops iterates the KDF on k, while other algorithms (e.g., the algorithm for deriving the commitment key) iterate on a fixed k concatenated with a counter. We would recommend sticking to that approach, which for instance decreases the (low) risk of a loop that would never end, and is the one used in most standards (e.g., FIPS 186-4).

B.11 Section 7

Completeness – *Improvement* – Definition 4 seems overly complicated.

- Why would $\mathcal{A}$ pick a (st, w) pair, rather than quantifying on all pairs in the relation as it is usually done?
- Why do you care about defining the tr variable, which appears to be unused?

Soundness – *Recommendation* – Definition 5 is also non-standard. In the traditional soundness definition (e.g., in [8]), there is a universal quantification on all the false statements (this is stronger than Definition 5). What is defined here rather looks like existential soundness (Attack Game 20.1 in [4]). The notion that you are really looking for should be clarified.

Special HVZK – *Recommendation* – Definition 6 is not a definition of special honest verifier zero-knowledge, which is a definition that only makes sense in the context of Σ-protocols. The usual definition of Special HVZK is available in [4, Definition 19.5] for instance. In particular definition 6

misses the requirement that the simulator always produce accepting transcripts.

Simulation soundness – *Recommendation* – In Definition 9, at Step 4 of the simulation soundness game, the notation $(st^*, \pi^*) \notin \mathcal{T}$ looks odd. $\mathcal{T}$ is supposed to be the list of programmed RO queries. As such, there is no reason for $\mathcal{T}$ to contain (st, π) pairs. We imagine that what is meant here is that (st^*, π^*) is not in the list of input outputs of the $\mathcal{S}'_{\mathsf{ps}}$ oracle, and that this list should be maintained by $\mathcal{S}'_{\mathsf{ps}}$?

Simulation extractability – *Recommendation* – Small typo at the end of the definition of *ext*: the final w should be w^*.

B.12 Section 8

Fiat-Shamir transform – *Improvement* – The definition of the Fiat-Shamir transform in Section 8.2 looks relatively ad-hoc, in particular when multiple challenges are needed and an extra "1" is concatenated to the hashed message. Besides, it seems slightly inconvenient that the size of the challenges is, by definition, the length of the output of the hash function.
One more flexible and uniform option would be to compute the hash, possibly as described in the document when there is one single challenge, then to derive the challenges by using a KDF on this hash and constant strings identifying each challenge that is produced. In the case of the HKDF function, the hash could be computed using the "extract" function of HKDF, and the various challenges could be produced using the "expand" function.

B.13 Section 9

DDH The definition of the DDH game (Fig. 16) seems wrong—the adversary should get $g_1, g_2, \ldots, g_l$ as well.

SGSP and DDH – *Improvement* – In Section 9.2, why are you requiring both the hardness of DDH and SGSP in order to imply the hardness of the ESGSP problem? Assuming that SGSP is hard seems to imply that DDH is hard as well in the kind of group that is considered here.

SGSP? – *Improvement* – The reliance on the hardness of the SGSP problem seems to be quite specific to this voting system (as you are pointing it, it was introduced for the Norwegian voting system, from which this system is derived). We are not aware of any specific study of the hardness of this problem (contrary to DDH, for instance) and, as such, this does only inspire limited confidence. It would be satisfactory to be able to remove that assumption, and rely on DDH instead. This would of course be an important change in the protocol.

B.14 Section 10

CRS – *Recommendation* – It is confusing that the group description and parameters are made available in a `crs`. The string in the `crs` model is *assumed* to be sampled according to some specific distribution, as a trusted setup, and it is typically used in security proofs in such a way that the challenger produces that string together with some trapdoor/auxiliary information. Here, we do not want to assume anything about this `crs`: we want it to produced in a verifiable way. So, the generation of the parameters should be part of a setup protocol rather than claimed to be a `crs`, and the outputs of this protocol should be verified by the concerned parties. These verification steps appear to be largely missing for the moment.

Groups and generators – *Recommendation* – The group parameters are produced in Section 10.1, but we do not see where they are verified. It seems very important, for privacy and verifiability reasons, that they are verified early enough. Ideally, the CCR's and CCM's would check that the ElGamal parameters are correct, and the CCM's would also check that the commitment keys are correctly produced. A lighter version would delegate the trust to the auditors, who would verify all these parameters in the VerifyConfigPhase protocol. But this is weaker, and possibly too weak because it assumes that there is a trustworthy auditor running the VerifyConfigPhase protocol *at the right time*, while it would be preferable that auditors could perform their task at any time.

Product of primes – *Recommendation* – The fact that the product of any subset of ψ primes remains smaller than p, as discussed in Section 10.3, should be an explicit part of the VerifyConfigPhase protocol.

B.15 Section 11

Logs – *Recommendation* – The `Logs` appear for the first time in Figure 19. They however play an extremely important role in the protocol, and are used basically everywhere. There should be a dedicated (sub)section about these logs, possibly before Sec. 11.1, explaining how they are created, who can write them, how their authenticity has to be checked, what security guarantees they offer, ... In effect, there are several cryptographic protocols missing here, despite their paramount importance.

Election public key – *Recommendation* – It is surprising that the election public key $\texttt{EL}_{\texttt{pk}}$ is not transmitted to the CCRs: such a communication would seem to be the natural way to inform them about the value of that key. In the current description, how is $\texttt{EL}_{\texttt{pk}}$ given as input to $\mathsf{VerifyBallotCCR}_j$? This should be clarified. If the CCRs do not have access to a properly certified version of $\texttt{EL}_{\texttt{pk}}$, then it appears that individual verifiability (among other properties) could be broken: a malicious voting server could provide a fake $\texttt{EL}_{\texttt{pk}}*$ to the voters and to the CCRs, who

would then validate ballots that would later be declared invalid by the auditors and the CCMs.

B.16 Section 12

Section 12.1.1.1 – *Improvement* – Why is k'_j sampled from $\mathbb{Z}_q^*$? It seems that any random string of suitable length (128 bits, or 256 bits) would be what is needed here.

Section 12.1.2.2 – *Recommendation* – The $\texttt{EB}_{\texttt{sk}}$ key is a single key generated by the print office. However, in Section 2.1.1, it is indicated that there is an electoral board that has a shared version of this key. If the key is expected to be shared, then we would assume that it is generated in a distributed way, as it is the case for the control components.

Section 12.1.1.4 – *Recommendation* – Some of the notations are hard to follow, and it would help to make them more consistent/explicit. For instance:

- At Step 5, it would help to state explicitly that $\mathbf{hpcc}_{\texttt{id}}$ is the vector of squared partial Choice Return Codes.
- At Step 6, the definition of $\mathbf{hpccHash}_{\texttt{id}}$ is confusing: we apparently have a vector that is made of a sequence of hashes, rather than a single hash (the same issue comes back in Item 7 of Section 12.2.1.6), but there is no index for that sequence. The notations are different and more clear in the Specification document. They could be adopted here.
- At Step 7, why do we need to add pTable in the logs? Section 10.3 indicates that they already are in the $\texttt{crs}$.

Section 12.2.1.2 – *Recommendation* – At the end of item 6, a ν notation appears (it comes back in some other places). Is it just ρ?

Section 12.2.1.3 – *Recommendation* – The $\mathsf{VerifyBallotCCR}_j$ function takes $\texttt{EL}_{\texttt{pk}}$ as an input, and it appears that it is through this input that the CCRs learn the election public key – we could not find any other channel by which they could learn it, and the System Specification document is more problematic since it claims (Figure 8) that the CCRs learn that key directly from the voting device, even though there is normally no communication channel from the voting device to the CCRs.

This seems to enable an attack against individual verifiability, and against the vote rejection property in particular: the voting server can give a voter and the CCRs an incorrect election public key, this will lead to an accepted ballot (since the CCRs will perform their verification against that incorrect public key), but the ballot will be rejected after mixnet decryption, assuming that the CCMs decrypt with the correct election public key.

Section 12.2.1.7 – *Recommendation* – At Step 5, why is it enough to check that there are entries for each of the ψ choices? Don't we have more specific ballot validity rules, e.g., in order to express that there is an answer to each single question on the ballot, and that we do not have no answer to one question and two answers to another question?

Section 12.1.1.8 – *Improvement* – The benefits of the Keystore do not seem very clear. Why not adopting a simpler process in which $\mathtt{k}_{\mathtt{id}}$ is derived from the Start Voting Key, that would go through PBKDF, then be expanded to $\mathbb{Z}_q$ using the H function? This completely non-interactive process seems to be just as secure, would save storage on the server side, and possibly avoid one round of communication between the voting client and the voting server.

Section 12.2.2.3 – *Improvement* – At Step 2, the notations are surprising. One would expect that $\mathtt{lVCC}_{\mathtt{id}}$ would be the product of the $\mathtt{lVCC}_{\mathtt{id},j}$ and not a hash of something else. (Note we are not saying the hash isn't needed for security, merely that the notation is confusing.)

Section 12.2.3 – *Recommendation* – It appears that $\mathsf{Logs}_{\mathtt{PO}}$ is missing from the inputs of the VerifyVotingPhase function (they are used at the third step of the parsing phase).

Section 12.3 – *Improvement* – It is unexpected that, at each step of the online phase, a decryption with a CCM key happens. We do not see any good reason for that, and it seems to have several downsides:

- If the CCM keys matter (which is not clear given that they are not produced in a verifiable way, implying that the global CCM key could be set to "1" and offer no confidentiality at all if the extra key from the trusted print office wasn't added – see also our remark on Section 12.1.2.1), then it would be important for each CCM to verify that it uses its keys to decrypt ciphertexts that it should decrypt. This would mean, among other things, running most of the steps of the VerifyConfigPhase, VerifyVotingPhase and VerifyOnlineTally functions before performing decryption. This is missing for the moment.
- It increases the latency of the mixing process. In a mix-net, most of the work is typically executed in parallel. Each mixer computes re-encryption factors in advance, then the input ciphertexts are shuffled and re-randomized very fast (it just takes 2 multiplications for a standard ElGamal ciphertext) and passed among the mixers, then all the mixers start computing their proofs in parallel, all the proofs are verified, and the decryption starts, again as a parallel process between all the key holders. When decryption is needed after each mixing step, the process becomes much more sequential: full exponentiations are needed before passing any ciphertext to the next mixer, and verification steps become important before running any decryption operation.

Unless there is some very good reason, which we do not see, for having one round of decryption after each mixing operation, we would recommend to remove those sequential decryption steps, and to run decryption as a parallel process after the final mixing and verification.

Section 12.3.2.1 – *Recommendation* – It appears that $\text{CCM}_{m'}$ uses the electoral board secret key $\texttt{EB}_{sk}$. How does it obtain it? It is also stated in other places (Section 2, Section 18) that this key can never be known to the adversary. Does that mean that $\text{CCM}_{m'}$ can never be corrupted? This seems incompatible with the requirement that any CCM is potentially corrupted, provided that one of them remains honest. See also discussion about the bpriv game in Section 18.

B.17 Section 13

Authentication – *Improvement* – The explanations contained in this section should come much earlier, as part to a description of the setup. Given the threat model, most protocols of Section 12 looked completely insecure until it is added here that channels are authenticated, and that public keys are verified through appropriate ceremonies.

B.18 Section 15

Section 15.1 – *Recommendation* – The definition of the IdealSetupVoting protocol is both very important and very unclear, as it is mostly hinted in the caption of Figure 25. Several variables seem to be just undefined: $\mathsf{GenVerDat}^{\$}$, the $\pi^{\$}$ proofs, ... There should be a detailed definition of this protocol, in order to make a review feasible.

Section 15.1 – *Recommendation* – Along the lines of the previous item: It is claimed that "The real setup is correct and private if..." That just does not seem true because the adversary may very well know all the return codes which clearly would seem to not be a private setup. For example if the encryption in GenCMTable was weak the adversary could break in and learn the return codes. We would suggest changing the ideal version of setup to return a random GenCMTable (with no relation to the return codes).

Section 15.1 The fact that public system parameters are not checked breaks Lemma 1, which relies on the SGSP problem, while there is no guarantee to have a SGSP problem instance. This break the sent-as-intended and recorded-as-confirmed proofs as well, since they rely on Lemma 1.

Section 15.2 – *Recommendation* – As it is written, it is not clear that a compliant ballot would be a valid ballot, contrary to what is claimed. A valid ballot has $\tilde{\texttt{E1}} = \texttt{E1}^{\texttt{k}_{\texttt{id}}}$ (unless the exponentiation proof is not sound). But a compliant ballot can have $\tilde{\texttt{E1}}$ being *any* encryption of the message

encrypted in $\texttt{E1}$ raised to the power $\texttt{k}_{\texttt{id}}$, that is the randomness of the two ciphertexts can be completely unrelated, which is not the case in a valid ballot.

Section 15.2 The notion of CompliantRegisteredBallot is problematic as it is not self-contained. As it is, $\mathsf{L}_{\texttt{sentVotes},h}$ and $\mathsf{L}_{\mathsf{pCC}}$ could just be anything, which makes it hard to make sense of this definition.

B.19 Section 16

Section 16.2 The Recorded-as-confirmed definition mentions that the auditors check that all control components use the same Confirmation Key $\mathsf{CK}_{\mathsf{id}}$ as a basis, but this is not mentioned in Algorithm 12.2.3 (VerifyVotingPhase).

Section 16.2.1 Section 16.2 contains an analysis of whether it is possible for the adversary either to get a vote confirmed when it should not be, or block its confirmation when it will be counted. Unfortunately, the games do not seem to adequately capture this. In particular, for the Vote rejection game:

- In the real protocol, all the messages are relayed through the untrusted Voting Server, so the attacker has the power to reorder, drop, or modify the messages. It might not send consistent messages to the CCRs, it might withhold a confirmation hoping that the voter will resend it, etc. The game (in Figure 30) doesn't make it entirely clear that this is permitted, but it should be—Step 11 is a little hard to interpret on this point, but it should include full adversary control of the communications in the ConfirmVote runs.

  In summary, the attacker model should include the opportunity to block, reorder or interfere with some or all channels via the untrusted Voting Server.
- The game in Figure 30 does not seem to include multiple runs, but the protocol allows for multiple confirmation attempts. So for example a real attacker might be able to mix and match values from different runs of the protocol, including perhaps some that it has interrupted—this should be reflected in the game.

B.20 Section 18

Figure 33 – *Recommendation* – Bpriv [2] was never designed to be used as a standalone definition but in conjunction with strong-consistency. Bpriv on it's own would allow an election system which publishes the voter's name next to the plaintext ballot to be considered private; it seems likely similar problems occur in the definition used in the Swiss Post system. If it is not the case, then explanations should be offered, since this is a significant departure from the literature.

Figure 33 – *Recommendation* – Several aspects of this game definition remain unclear, which makes it hard to evaluate.

- At Step 3, writing that $\texttt{EB}_{sk}$ is unknown to the adversary seems to imply that the last CCM can never be corrupted. Is that right?
- At Step 3 of the bpriv game definition, what is c?
- At Step 5, why does the challenger initialize the bb ballot box? It seems that it is only used by the adversary, who has full control of it.
- In the voting phase, we guess that access to the $\mathcal{O}\mathsf{HonestCCRexp}$ oracle is missing?
- In the tally phase, can the adversary decide to call the $\mathcal{O}\mathsf{HonestCCM}_{\mathsf{offline}}$ oracle if he decided that CCM_h is the online CCM (and conversely)?

Figure 34 – *Recommendation* – The use of sets fails to capture possible ballot copying attacks on the cleaned bulletin board. In the game, the recovery algorithm recovers all ballots, and not only honest ballots despite what the informal description says which again may miss ballot copying attacks, etc. (Strictly speaking the definition may not miss the issue depending on how the set operations are interpreted but at human verifying proof may well.)

Figure 34 – *Recommendation* – The definition requires that the two sets of votes (V_h^0, V_h^1) are balanced which we have never seen for a definition which is going to always count as though bb_0 anyway. Why this difference? How can we be convinced that it does not introduce weaknesses?

Figure 34 – *Recommendation* – In the $\mathcal{O}\mathsf{HonestCCM}_{\mathsf{online}}$ oracle, when $\beta = 1$, it appears that the ciphertexts that are mixed are not rerandomized – they are just partially decrypted. As a result, the first component of each ciphertext is not modified at all and this reveals the permutation that is applied. This is probably not the intent of the authors.

Figure 34 – *Improvement* – In the $\mathcal{O}\mathsf{HonestCCM}_{\mathsf{offline}}$ oracle, the $\mathsf{bb}_{\mathsf{clean}}$ input is never used. Is that expected?

Figure 35 – *Recommendation* – Some elements are not clear:

- In the $\mathcal{O}\mathsf{CastBallot}$ oracle, at Step 5, how are the ν variables defined? Are they just the $\mathbf{v}$ variables?
- In the $\mathcal{O}\mathsf{ConfirmBallot}$ and $\mathcal{O}\mathsf{VerifyVCC}$ oracles, the $\mathbf{cc}_{\mathtt{id}}^{\beta}$ and $\text{VCC}_{\mathtt{id}}$ variables are not defined. Why does one of them get a β and not the other?

Top of p. 106 – *Improvement* – It is claimed, on p. 106, that an adversary who would make sure that only one ballot is confirmed can trivially win the privacy game. Why is it the case, since the announced tally is always the one corresponding to $\beta = 0$?

B.21 Section 19

Game vc.4 – *Recommendation* – Game vc.4 considers the possibility that a maliciously generated $lpCC$ might match something in the table despite $(pCC_{id,i}) \neq p_i^{k_{id}}$. The proof states, "the bad event happens ... only if there is a hash collision of some adversarial $(pCC_{id,i})$ with some value in the L_{pCC}." But the bad event could also happen if the adversarially-generated $(pCC_{id,i})$ matches *something* in the L_{pCC} table, but not the value it should match. There needs to be an argument that this cannot happen, and it should depend on the algebraic properties of the code generation - the elements of L_{pCC} are exponents of primes, and it should not be possible to generate a new value of that form by multiplying others (at least, that is what needs to be proven). This proof will also depend on the proper construction of the L_{pCC} table.

To repeat the notation of our earlier attack, there needs to be an argument that the adversarially-generated $(pCC_{id,i}) = (pCC_{id,2})(pCC_{id,3})/(pCC_{id,1})$ can never match a valid $(pCC_{id,i})$ *including ones that the voter may not expect*. The adversary must be assumed to have access to L_{pCC} and hence be able to guess and check whether certain values are valid.

Assuming that the $(pCC_{id,i})$ values are properly generated, they have the form $p_i^{k_{id}}$, so this requirement resembles the SGSP assumption. However, the reduction is not immediate because the adversary contributes to choosing k_{id}, since they have a role in *constructing* the table. For example, if they could arrange matters so that $k_{id} = 0$, then $(pCC_{id,i}) = (pCC_{id,2})(pCC_{id,3})/(pCC_{id,1})$ would hold. (We do not think this is actually possible, but the proof should detail why not.)

We do not think there is a real attack here (though we are not sure), because we think the prime-powers cannot be created in this way, but there needs to be a proof of this.

Theorem 1 (sent-as-intended) obviously depends on this.

Game mRTC.3 – *Recommendation* – The reduction to the kdf function security does not work, contrary to what is claimed here. This is because the kdf function is called multiple times with k'_h as only secret input, while the kdf security game only allows for one single call. This does not lead to a practical attack, because any standard KDF offers much more security than what the form of security that is required in Section 6. But it would be easy to build an artificial KDF that would satisfy the security definition given there, and would completely break the security of the voting scheme.

Game bpriv.1 – *Recommendation* – There is a mention of a canonical generator $\mathcal{S}_{mix}$ here, but we do not see how it is defined. Section 7.2.1 and reference [25] only define canonical generators for standard 3-passes Σ-protocols, but the shuffle argument does not fit that structure. It would help, for completeness, to define $\mathcal{S}_{mix}$.

Game bpriv.7 Public system parameters are not checked. This could make it possible to generate the primes table so that the discrete log relation were known. As a result, we would not have an instance in which the SGSP problem can be expected to be hard, which breaks the proof.

Game bpriv.7 Even if the previous point is addressed, it is claimed that the only way of distinguishing $\mathtt{CK}_{\mathtt{id}}$ and $\mathbf{pCC}_{\mathtt{id}}$ (among others) from random group elements would lead to an adversary against the ESGSP problem. We do not see how this can be true, since $\mathtt{CK}_{\mathtt{id}} = (BCK_{\mathtt{id}})^{2k_{\mathtt{id}}}$ and $(BCK_{\mathtt{id}})^2$ is not a prime integer. It turns out that this is not just a gap in the proof: there is a potential privacy attack here that is not captured by the bounds of the theorem and would succeed with much higher probability than what the theorem guarantees, even if that probability remains relatively low.

Let us suppose, for the simplicity of the exposition, a very simple election with two possible choices, one encoded with a prime $p_1 = 3$ (it could be any prime in $\mathbb{G}_q$ strictly smaller than 10^9), and the other with a prime $p_2 \in \mathbb{G}_q$ larger than 10^9, which is the highest value that $BCK_{\mathtt{id}}$ can take. (The attack can work with many more realistic choices, but the point is to show the simplest version of the attack that contradicts the security proof.) An honest voter submits his ballot, with a choice encoded as a prime p (which is either 3 or the large prime, we would like to know). The CCRs decrypt E2 and the adversary obtains $pCC_{\mathtt{id}} = p^{\mathtt{k}_{\mathtt{id}}}$. The return code is shown to the voter, who is happy and submits his confirmation key $\mathtt{CK}_{\mathtt{id}} = \mathtt{BCK}_{\mathtt{id}}^{2k_{\mathtt{id}}}$. Now the voting server tests if $pCC_{\mathtt{id}}^2 = \mathtt{CK}_{\mathtt{id}}$. If the test succeeds, then $p = p_1$ since $p_2 > \mathtt{BCK}_{\mathtt{id}}$, and the attacker knows how the voter voted. Otherwise, we say that the attack fails. So, the attacker learns a vote in clear with probability 1 over the size of the BCK space, which is 10^9 according to Section 20.5. That is not quite practical (even though there are various ways to improve the attack quite a bit). But is definitely much better than what the privacy theorem says.