Running the Race: A Swiss Voting Story

Thomas Haines[1(✉)], Olivier Pereira[2], and Vanessa Teague[3,4]

[1] The Australian National University, Canberra, Australia
thomas.haines@anu.edu.au
[2] UCLouvain – ICTEAM – B-1348, Louvain-la-Neuve, Belgium
olivier.pereira@uclouvain.be
[3] Thinking Cybersecurity Pty Ltd., Fairfield, Australia
[4] Australian National University, Canberra, Australia
vanessa.teague@anu.edu.au

Abstract. On the 29th of March 2019 the Swiss Federal Chancellery launched a review of the procedures surrounding e-voting after numerous flaws were discovered in the Scytl-Swiss Post system sVote. On the 5th of July 2021 an independent examination of the revised Swiss Post system began, with some cantons planning to launch new trials with this system.

We summarize and reflect on our experience with the examination of the cryptographic protocol so far and muse over the future. We find that the protocol specification considerably improved over the last 3 years, both through changes in the protocol itself and through clarifications of missing elements in its specification. The clarifications also shed a new light on shortcomings of the protocol, in terms of both verifiability and privacy, including in the latest version of the system, which remains incompletely specified.

We believe that these findings illustrate virtues of the examination requirements set by the Swiss Federal Chancellery: problems can be fixed before deployment rather than being exploited by malicious parties during an election. They also illustrate the tremendous challenges of creating a secure Internet voting system, and the long road ahead.

1 Introduction

Switzerland has a long history in internet voting in political elections spanning nearly twenty years. It has also been a leader in regulating internet voting, particularly since the introduction of the Federal Chancellery Ordinance on Electronic Voting (VEleS) in 2014. This ordinance, particularly the revised version of 2018 [14], details not only security requirements for the system but requirements for the processes around the use of e-voting. Particularly crucial are the requirements which relate to transparency, for example the requirement that "Anyone is entitled to examine, modify, compile and execute the source for ideational

This paper is based on a review performed with the financial support of the Swiss Federal Chancellery.

R. Krimmer et al. (Eds.): E-Vote-ID 2022, LNCS 13553, pp. 53–69, 2022.
https://doi.org/10.1007/978-3-031-15911-4_4

purposes, and to write and publish studies thereon." (Art. 7b.4) We shall see that this requirement has been crucial in revealing issues in systems deployed in Switzerland.

There have historically been several different e-voting systems used by different cantons; the most prominent of these have been the CHVote open source system [3] backed by the canton of Geneva and the sVote proprietary system by Scytl and Swiss Post. Version 1.0 of the sVote protocol, which is the precursor of the current Swiss Post system [20], was used between 2016 and 2019. The system has been required since the beginning to provide individual verifiability, which it aimed to achieve through a technique called return-code voting. In the Swiss return-code voting systems, each voter receives a paper sheet containing random secret verification codes for each candidate before the election. The voter votes online by ticking their choices on web page and, in return, the browsers must show the codes that match those shown on their paper sheet. This should allow a malicious voting client to be detected should it change the voter's choices. sVote 2.1 was announced in 2018 and was designed to also provide universal verifiability. sVote 2.1 progressed through the certification process until the system was made public; at that point external experts found a large collection of errors which affected all aspects of the security of the system from privacy to verifiability. Interested readers may wish to peruse the reports by Haines et al. [4] and Locher et al. [8]. The system was withdrawn from use following these findings.

On the 26th of June 2019 the Federal Chancellery was commissioned to redesign the trial phase of e-voting with the aim to establish stable trial operations. This redesign was to have four major objectives [17]:

1. Further development of the systems
2. Effective control and oversight
3. Increasing transparency and trust
4. Closer cooperation with the academic community

The first stage of this was a dialog with various stakeholders across academia, industry and government. Based on this, the legal basis is being amended and the independent test trials have been relaunched. On the 5th of July 2021 independent experts, of which we were part, were commissioned to examine the compliance of the system with the requirements under federal law [16]. While reports from the first round of examination are available [1], the examination is still ongoing and will serve as a basis for the Federal Council's decision on whether to allow cantons to conduct e-voting. This paper summarises the situation based on Release 0.8.5.0 [18], and does not incorporate improvements made by SwissPost in their updated releases of June 24, 2022.

There is much to be applauded about how Switzerland is handling this process. However, breaking new ground is not without its difficulties. What is being attempted has never been done before and the time required to complete the process may be longer than certain stakeholders would like [13]. It is important to remember that a good certification process should *not* prematurely certify a system that does not meet requirements. It is a design feature *not* to deploy a system, even if people are expecting it to be ready, if it is not in fact ready yet.

There is no guarantee that a sufficiently secure, practical and usable system will be created in the expected timeline, if at all.

1.1 The Swiss Post Protocol

The Swiss Post e-voting system consists of numerous components which are housed either within the relevant Canton or within Swiss Post, see Fig. 1. We will now summarise the protocol, introducing the components as they become relevant by name and by the abbreviation used for the component in Fig. 1. Our protocol description is deliberately incomplete, focusing on the elements that will be useful in our further discussions. The current specifications are not particularly coherent when it comes to the components of the system. While we do our utmost to be clear, some confusion as to the participants is unavoidable in our paper since it exists in the specifications. In particular, the protocol roles include two main groups of control components, the Return Codes control components (CCR)s and the Mixing control components (CCM)s; in Fig. 1, the components denoted CC refer to a component combing the functions of a CCR and CCM. The trust assumption is that at least one member of each group of control components remains honest.

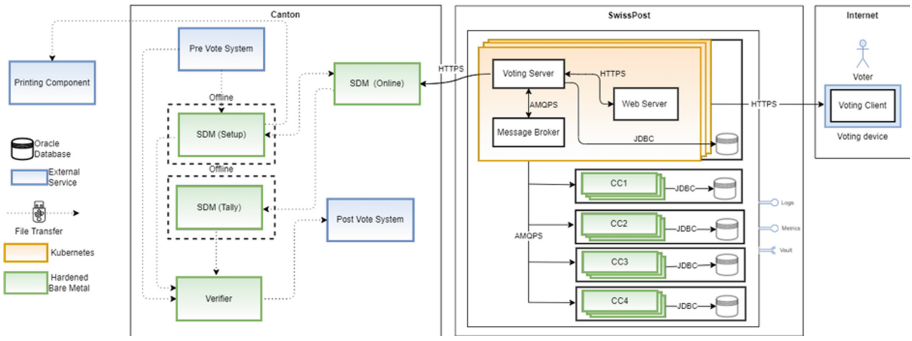

Fig. 1. System deployment - Fig. 12 from the Swiss Post E-Voting Architecture Document 1.0.0 in accordance with permitted use.

Configuration Phase. The system begins with a trusted setup component, depicted as SDM (Setup), creating the global parameters for the system. The CCMs then jointly generate a public key by running a protocol called SetupTally, at the end of which the CCMs all have a share of the secret key as does the electoral board (which does *not* appear in Fig. 1). The trusted print office together with the CCRs run a protocol called SetupVoting which generates voting cards containing that contain voter credentials and verification codes and will be sent to the voters, and stores the cryptographic information (denoted CMtable) that will be needed to recover the return codes with the help of the CCRs.

Voting Phase. The voter, having received their voting card by mail, uses the web-based voting client to create their vote. This vote is then sent via the untrusted voting server to the CCRs who, if the vote is valid, jointly compute, using the CMtable and the information received from the voter, the return codes to be returned to the voter. If the voter accepts their return codes, they submit the ballot casting key printed on their verification card. The CCRs check that this confirmation code is valid and if so jointly compute and return a vote cast return code.

Tally Phase. First the ballot box is cleansed to remove unconfirmed ballots and all information except the encrypted votes. The online CCMs, hosted by Swiss Post, then in sequence mix and partially decrypt the ballots.

Audit. The auditors verify the proofs generated by all the control components using VerifyVotingPhase and VerifyOnlineTallyPhase, before the CCM hosted by the canton does the final mix and decryption. The auditors then check the proofs generated by the canton CCM using VerifyOfflineTallyPhase.

In the following sections we will regularly use the same symbols as the Swiss Post specifications to facilitate interested readers making comparisons. In all cases we will first provide an explanation of the symbol in prose.

1.2 Outline

The remainder of this paper proceeds in four sections. In Sect. 2 we summarise the security required by the current draft ordinance. We have split the results of our examination of the Swiss Post system into two sections: in Sect. 3 we discuss the state of the documentation and security proofs before discussing attacks on the system in Sect. 4. Finally, we conclude in Sect. 5 and discuss possible directions that the examination process may take.

2 The Requirements

The requirements detailed in the draft ordinance are extensive and we will focus on those that pertain to the cryptographic protocol and system implementation.

Art. 3 of the April 28, 2021 draft ordinance [15] outlines four high level requirements which follow under our own headings:

Secure. "The system is designed and operated so as to guarantee verifiable, secure and trustworthy vote-casting."

Usable. "The system is easy to use for the eligible voters; account must be taken of the special needs of all voters wherever possible."

Clear. "The system and the operational procedures are designed and documented so that the details of the technical and organisational procedures can be checked and understood."

Transparent. "The general public have access to information appropriate to the addressees on how the system works and its operational processes, and there are incentives for specialists among the general public to participate."

We focus on the security and clarity: our impression is the process has the desired level of transparency and we are not qualified to assess the usability.

Requirements for complete verifiability. Complete verifiability captures the notion that it should not be possible to manipulate the result of the election without detection under certain trust assumptions on the system. To capture the latter, the ordinance considers parts of the system to be trustworthy. The notation of complete verifiability is based on two subnotations which we detail below:

Individual Verifiability. ensures that it is possible to detect manipulation of the ballot on the user's device. It should also ensure that the ballot is correctly recorded by the trustworthy part of the system. Furthermore, it should be possible for a voter who did not cast an electronic ballot to receive a proof that no ballot was received on their behalf. Individual verifiability corresponds to what is often refereed to as cast-as-intended and collected-as-cast verifiability in the literature, up to differences in the trust assumptions.

Universal Verifiability. captures that the result contains all registered votes and only those cast in conformity with the system. This corresponds to what is referred to as counted-as-collected verifiability, up to differences in the trust assumptions.

Preservation of voting secrecy. The requirements require that the secrecy of votes should be preserved provided that at least one of the control components of each group is honest and the voter's device follows the protocol and doesn't leak the vote. (This is challenging in practice, since the voter's device is expected to obtain the JavaScript code that it uses to prepare its ballot from the untrusted voting server.)

2.1 Comments on the Requirements

Positively, the draft ordinance clarifies many of the issues in the previous version. However, it continues to align more and more closely with a properly implemented version of the Swiss Post protocol. We would encourage including incentives to design stronger systems; for example by assigning grades to systems to facilitate decision making by cantons. We have heard numerous stakeholders indicate their desire to develop stronger systems after the current system meets the current requirements. But competition would be extremely hard if competition was based only on price, because stronger security does not bring an added value.

The notions of verifiability required are weaker than those common in the academic literature, which are incompatiable with a trusted print office or trusting one of the control components. In some cases, these differences allow for better usability; in others, it is unclear why the system should not be required to achieve a higher level of security. For example, return-code voting provides a tradeoff between usability and trust assumptions for which no strictly better solution is known. On the other hand, some forms of trust allowed in the setup components are unnecessary.

3 The Specification and Proofs

The Swiss Post e-voting system's protocol design is captured in the two documents entitled, "Protocol of the Swiss Post Voting System," [19] and "Swiss Post Voting System – System specification" [21]. The information about the protocol is slightly less detailed in the Protocol document than the System Specification, but the former also includes security games and proofs. We will first discuss issues with the scope of the protocol specification before discussing its alignment with the VEleS.

3.1 The Protocol Specification is too Narrow

One of the hard things in protocol design is choosing a proper layer of abstraction to describe the protocol. This abstraction should not hinder comprehension with unnecessary details but should include sufficient information to conclude the protocol is secure. The latter requirement is captured in 2.14 of the VEleS which states "One symbolic and one cryptographic proof must demonstrate that the cryptographic protocol meets the requirements in Numbers 2.1–2.12. The proofs must directly refer to the protocol description that forms the basis for system development. The proofs relating to basic cryptographic components may be provided according to generally accepted security assumptions (e.g. 'random oracle model', 'decisional Diffie-Hellman assumption', 'Fiat-Shamir heuristic')."

At the time we examined the system the following three areas were particularly noticeable as underspecified:

Authentication: 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.[1] This is detailed in Sect. 4.1., and we believe that this stresses the importance of the completeness of the protocol specification.

Authorisation: The security of the system also depends on when and by whom various processes can be called, which is not detailed.

Error Handling: 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 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 VEleS 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

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

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 VEleS, the potential enormous complexity of interpreting the inconsistencies, the potential that the inconsistencies may create situations in which it is not even possible to decide whether a recorded vote matches a voter intent or not, 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 in Sect. 4.2, but it may be relevant in other parts of the protocol too.

3.2 The Roles and Channels are Incompletely Aligned with the VEleS

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 VEleS.

In particular, the role of the auditors and of the electoral board, as described in the protocol specification, appears to be problematic.

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 VEleS. As such, and following Art. 2.1 of the VEleS 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".

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 (Sect. 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 it 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 VEleS 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 VEleS Appendix.

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, Fig. 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 VEleS 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.

4 The Bugs

Having discussed some high level issues with the protocol and requirements, we now focus on some vulnerabilities which we discovered during our examination.

4.1 Lack of Authentication: Attack on Individual Verifiability

This section of our report refers to a vulnerability disclosed to Swiss Post in March 2021,[2] 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[3] 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.

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.

[2] https://gitlab.com/swisspost-evoting/e-voting/e-voting/-/issues/1.

[3] This vulnerability was detected in version 0.7.

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

This could allow the adversary to impersonate the one honest return code control component starting in the config phase and run 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.

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

At the time of writing, 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.

4.2 Lack of Details in Handling Inconsistencies: Attack on Individual Verifiability

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 a confirmation key $\texttt{CK}_{\texttt{id}}$ 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. For reasons of space our description here is necessarily incomplete—more information can be found in our report from Round 1 of the examination [5].

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.[4] 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 VEleS.

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.

What Inconsistent Logs Should be Permitted? Let us consider CCR logs that are almost, but not perfectly, consistent. This may be due to communication mishaps, a corrupted voting server, or one or more malicious CCRs.

We focus on the confirmation logs ($\texttt{L}_{\texttt{confirmed},j}$) and, in the rest of this discussion, we omit $\texttt{lVCC}_{\texttt{id}}$ 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: (where $\texttt{vc}_{\texttt{id}}$ is the verification card identifier)

$$\texttt{CCR}_j : (\texttt{vc}_{\texttt{id}}, 1, \texttt{CK}_{\texttt{id}}, *, *) \text{ for } j = 1, 2, 3.$$
$$\texttt{CCR}_4 : \text{No record for } \texttt{vc}_{\texttt{id}}$$

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

[4] This was reported to SwissPost as a gitlab issue which is currently private.

2. The voter enters his true Ballot Casting key $\texttt{BCK}_{\texttt{id}}$. The client honestly computes $\texttt{CK}_{\texttt{id}}$ 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 Sect. 12.2.2.2 of the protocol specification correctly, including logging, and return long vote cast return code $\texttt{lVCC}_{\texttt{id},j}$ ($j = 1, 2, 3$) to the Voting Server.
5. Cheating $\texttt{CCR}_4$ computes $\texttt{lVCC}_{id,4}$ 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 Sect. 12.2.2.3.)
7. *The Voting server also computes correctly (but does not log) the value of* $\texttt{lVCC}_{id}$ *derived from a correct execution of 12.2.2.3 using the* $\texttt{lVCC}_{id,j}$*'s received from honest CCRs (*$j = 1, 2, 3$*), plus the* $\texttt{lVCC}_{id,4}$ *it received out-of-band from the cheating* $\texttt{CCR}_4$. 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 Sect. 12.2.2.3.
8. *The Voting Server then sends the (correct)* $\texttt{VCC}_{\texttt{id}}$ *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, 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 Sect. 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 Sect. 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:

$\mathtt{CCR}_j : (\mathtt{vc}_{\mathtt{id}}, 1, \mathtt{CK}_{\mathtt{id}}, *, *), (\mathtt{vc}_{\mathtt{id}}, 2, \mathtt{CK2}_{\mathtt{id}}, *, *)$ for $j = 1, 2, 3$.
$\mathtt{CCR}_4 : (\mathtt{vc}_{\mathtt{id}}, 1, \mathtt{CK2}_{\mathtt{id}}, *, *), (\mathtt{vc}_{\mathtt{id}}, 2, \mathtt{CK}_{\mathtt{id}}, *, *)$

These logs could be the result of various scenarios very similar to the previous ones. For instance, it may be that the voting client was honest, the voter entered a correct $\mathtt{BCK}_{\mathtt{id}}$ value, and a correct $\mathtt{CK}_{\mathtt{id}}$ was sent to the voting server, but the malicious voting server created $\mathtt{CK2}_{\mathtt{id}}$ as well and sent the values $\mathtt{CK}_{\mathtt{id}}$ and $\mathtt{CK2}_{\mathtt{id}}$ to the first three CCRs, and the values $\mathtt{CK2}_{\mathtt{id}}$ and $\mathtt{CK}_{\mathtt{id}}$ to $\mathtt{CCR}_4$. The corrupted voting server may then decide to send the correct Vote Cast Return code to the voting client, after reordering the responses from $\mathtt{CCR}_4$. 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 $\mathtt{CK}_{\mathtt{id}}$ and $\mathtt{CK2}_{\mathtt{id}}$ are incorrect values.

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

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

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.

It is also unclear whether a $\mathtt{VCC}_{\mathtt{id}}$ would be returned to the voter in any of these cases.

What do the Specification Documents Say About These Cases? We inspect the different available documents in order to try to interpret what would happen.

Protocol Specification. The scenarios 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 $\texttt{vc}_{\texttt{id}}$ and confirmation attempts number in $\{\mathsf{L}_{\texttt{confirmed}_j}\}_{j=1}^{m}$. 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 attempts for $\texttt{vc}_{\texttt{id}}$.

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 an 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 Sect. 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 the control components' state and in the list of ballots to be included in the tally.

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—see [5] for details.

Verifier Specification. The verifier specification (version 0.9.1) is more demanding, and it appears from Sect. 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 Sect. 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.

4.3 Lack of ZK Proofs of Correct Key Generation: Attack on Privacy

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 VEleS, 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 VEleS.

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' public key shares $(EL_{pk,1}, EL_{pk,2})$ 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.

5 Conclusion

The Swiss regulations and processes for e-voting are world leading and we strongly advocate adoption of similar processes in countries like Australia, Estonia, and any other jurisdiction using Internet voting for political elections. The Swiss Post e-voting system is continuing to improve, gradually fixing issues it inherited from sVote. However, the system is still not complete and significant security issues are still being discovered.

This experience may feel frustrating for the stakeholders who are looking forward to a swift return of e-voting in Switzerland, especially when e-voting has been used for years.

Our feeling is rather that the process illustrates difficulties that were always there.

- The design of an Internet voting system that would offer security in a context that is suitable for government elections is widely regarded as an open question by the academic community [9,11,22].
- The other countries that decided to open their Internet voting system to public scrutiny (and many that didn't) also faced the discovery of significant security issues – see the cases of Norway, Estonia, Australia and Russia for instance [2,6,7,10,12,23].

Switzerland adopted regulations regarding the review of its Internet voting system that are well aligned with the practices adopted for other high-impact cryptographic protocols. The process is however made quite challenging because of the unique set of requirements adopted by Switzerland on the one hand, and because of the almost complete absence of existing standards regarding e-voting protocols, and on which a Swiss system could rely. As a result, we encourage all stakeholders to allow sufficient time for the system to be properly developed and reviewed before deployment. Remember that *not* certifying a non-compliant system is a desirable goal of a good process.

References

1. Federal Chancellery. E-voting: Results of the first independent examination available (2022). https://www.bk.admin.ch/bk/en/home/dokumentation/medienmitteilungen.msg-id-88085.html
2. Gaudry, P., Golovnev, A.: Breaking the encryption scheme of the Moscow internet voting system. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 32–49. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_3
3. Haenni, R., Koenig, R.E., Locher, P., Dubuis, E.: CHVote system specification. Cryptology ePrint Archive, Report 2017/325 (2017). https://ia.cr/2017/325
4. Haines, T., Lewis, S.J., Pereira, O., Teague, V.: How not to prove your election outcome. In: IEEE Symposium on Security and Privacy, pp. 644–660. IEEE (2020)
5. Haines, T., Pereira, O., Teague, V.: Report on the Swiss post e-voting system (2022). https://www.newsd.admin.ch/newsd/message/attachments/71147.pdf
6. Specter, M., Halderman, J.A.: Security analysis of the democracy live online voting system (2020)
7. Halderman, J.A., Teague, V.: The New South Wales iVote system: security failures and verification flaws in a live online election. In: Haenni, R., Koenig, R.E., Wikström, D. (eds.) VOTELID 2015. LNCS, vol. 9269, pp. 35–53. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22270-7_3
8. Locher, P., Haenni, R., Koenig, R.E.: Analysis of the cryptographic implementation of the Swiss post voting protocol (2019). https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting.html

9. National Academies of Sciences, Engineering, and Medicine: Securing the Vote: Protecting American Democracy. The National Academies Press, Washington, DC (2018)
10. Pereira, O.: Individual verifiability and revoting in the Estonian internet voting system. In: Proceedings of the 7th Workshop on Advances in Secure Electronic Voting (2022). https://ia.cr/2021/1098
11. Pilet, J.-B., Preneel, B., Erzeel, S., Pereira, O.: Étude sur la possibilité d'introduire le vote internet en belgique (2020). https://elections.fgov.be/informations-generales/etude-sur-la-possibilite-dintroduire-le-vote-internet-en-belgique
12. Springall, D., et al.: Security analysis of the Estonian internet voting system. In: Proceedings of the 21st ACM Conference on Computer and Communications Security. ACM (2014)
13. Swiss Community: Swiss post e-voting to operate from 2023 (2022). https://www.swisscommunity.org/es/news-media/swisscommunity-news/swiss-post-e-voting-to-operate-from-2023
14. Swiss Federal Chancellery: Federal chancellery ordinance on electronic voting (2018). https://www.fedlex.admin.ch/eli/cc/2013/859/en
15. Swiss Federal Chancellery: Federal chancellery ordinance on electronic voting (draft of 28 April 2021) (2021). https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting.html
16. Swiss Federal Chancellery: Federal government launches examination of new e-voting system (2021). https://www.bk.admin.ch/bk/en/home/dokumentation/medienmitteilungen.msg-id-84337.html
17. Swiss Federal Chancellery: E-voting (2022). https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting.html
18. Swiss Post: E-voting documentation 0.8.5.0 (2021). https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/documentation-0.8.5.0
19. Swiss Post: Protocol of the swiss post voting system - version 0.9.11 (2021). https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/documentation-0.8.5.0/Protocol
20. Swiss Post: Swiss post voting system (2021). https://evoting-community.post.ch/
21. Swiss Post: Swiss post voting system - system specification - version 0.9.7 (2021). https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/documentation-0.8.5.0/System
22. U.S. Vote Foundation: The future of voting: end-to-end verifiable internet voting - specification and feasibility study (2015). https://www.usvotefoundation.org/E2E-VIV
23. Espen Zachariassen: Feil i krypteringen av e-stemmer (2013). https://www.tu.no/artikler/feil-i-krypteringen-av-e-stemmer/234436