

Final Report on Releases 1.5.0, 1.5.1, and 1.5.2

Thomas Haines

Olivier Pereira*

January 9, 2026

Summary

This report covers changes made to the Swiss Post e-voting system in releases 1.5.0, 1.5.1, and 1.5.2; we provide an update on the status of outstanding issues from our previous reports (Section 1) as well as the status of various items in the Catalogue of Measures (Section 2). We did not exhaustively check the changes to the specification and code since these exceed 100,000 lines; however, we did examine certain changes and new releases which seemed to us to be of importance (Section 3).

Overall the quality of the implementation continues to improve with a slower but still positive quality change in the specification.

Summary of Issues Raised in Prior Reports

Most of outstanding issues from our prior reports were not addressed in 1.5 and are scheduled to be addressed as part of 2.0.

Summary of Catalogue of Measures Items

Based on our understanding of the kickoff documents from this round of review, one of the main claims of version 1.5 was to resolve six of the items in the Catalogue of Measures; we comment on the status of these measures below:

A.10 Reduce dependencies on external software While the dependencies have been reduced in some parts of the systems, others still contain a very large number of dependencies. (Section 2.10)

A.11 Disclose source code of software for generating PDF files We agree that this measure has been fulfilled. (Section 2.11)

A.15 Object-Oriented Design in crypto-primitives We did not analyse this issue. (Section 2.15)

A.21 Implement the dispute resolver We agree that this measure has been fulfilled. (Section 2.21)

*O. Pereira only contributed to the review within Scope 1.

A.22 No operational tasks for Auditors We agree that this measure has been fulfilled. (Section 2.22)

A.25 Improvements in specification and code This measure includes about 50 subpoints detailed in the Annex to the Catalogue of Measures; many of this do not seem to have been addressed. We understand from discussions with Post and the Chancellery that this measure should have been listed as partially resolved rather than resolved, since Post did not believe that all subpoints had been addressed. (Section 2.25)

Summary of New Examinations

The only item of note from our additional examinations was a vulnerability which impacted individual verifiability, which would be detected during tally (Sec. 3.10). This issue is already covered by measure A.26 of the Catalogue of Measures and noted as not covered by the security analysis in the computation proof.

We also found some minor vulnerabilities in the VOTING Stimmunterlagen (Sec. 3.7) and Direct Trust Tool (Sec. 3.8), which have since been patched.

Contents

Summary	1
1 Status of Previously Raised Issues	5
1.1 (Missing) Description of protocol	5
1.2 Alignment of protocol with ordinance	5
1.3 Context vs Input	5
1.4 Auditor instructions	6
1.5 Inconsistent naming of setup/configuration Verification	6
1.6 Usage of algorithms and chunking	7
1.7 Alignment of Voting Card Print Service	7
1.8 Minor issues raised in 1.4.5	8
2 Catalogue of Measures Reviews	9
A.10 Reduce dependencies on external software	9
A.11 Disclose source code of software for generating PDF files	10
A.12 Further develop symbolic proofs	10
A.15 Object-Oriented design in crypto-primitives	10
A.21 Implement the dispute resolver	10
A.22 No operational tasks for Auditors	12
A.24 Improvements in computational proof	13
A.25 Improvements in specification and code	13
3 New Examinations and Issues	14
3.1 Update to the architecture document	14
3.2 Changes to ZKP proof verification	14
3.3 Simplify verification of Maurer’s proofs	15
3.4 Change to allow voters to explicitly abstain from elections and votes	15
3.5 Agreement Algorithms - in system specification	16
3.6 Data Integration Service	17
3.7 VOTING Stimmunterlagen	17
3.8 Direct Trust Tool	17
3.9 Examination of id consistent within the system	18
3.10 Complex ballots and individual verifiability	19
3.11 Points Arising from Revised Security Analysis	19
3.12 RecursiveHashOfLength efficiency	20
3.13 Add voting system version into GetHashElectionEventContext	21
3.14 Include key index in Schnorr proofs?	21
3.15 Ask the CCR/CCM to run VerifyConfigPhase?	22
3.16 Explanation of failure modes in the dispute resolution	22
3.17 Miscellaneous observations	23
4 References	24

A	A.22 Analysis	26
A.1	Is the implementation sufficient to ensure the public key is adequately validated?	26
A.2	Is the specification/implementation sufficient to ensure the encryption group is consistent and correct?	27
A.3	Is the specification/implementation sufficient to ensure the verification card set ids and verification card ids are consistent across the control components?	27
A.4	Do the control components adequately verify the prime mapping table? .	27
A.5	Is VerifyTotalVotersConsistency unnecessary?	28

1 Status of Previously Raised Issues

Most of the issues we previously raised, which are still open, are captured in the Catalogue of Measures. For these issues we will defer discussion to Section 2.

1.1 (Missing) Description of protocol

In our report on 1.4.* [2] we noted that:

The system specification covers a lot of the protocol in reasonable detail including the algorithms and digital signatures. However, the description of how the components should manage state and what protocol flows are valid is communicated principally through the sequence diagrams and stateful lists. These leaves open to interpretation many questions.

At present the verifier specification treats all checks required on the same level of abstraction and so cryptographic checks are described at the same level as validations of ensuring all the data has been received or that file names are correct. This not ideal in that presentation of the core protocol checks are cluttered by many which exist at a much lower level of abstraction than exists in the system specification. Moreover, many of the non-core protocols checks are underspecified and consist principal of text rather than pseudocode.

– *Status* –

This issue has *not* been addressed in version 1.5.0 and is currently planned for 2.0.

1.2 Alignment of protocol with ordinance

In our report on 1.4.* [2] we noted that:

We remain of the view that a system which conformed to the requirements of the ordinance would be able to provide a mapping of participants which was consistent. That the current solution by Post is unable to do this without creating undue confusion is a problem. This does not need to be immediately resolved but should be taken into account when measures like A.05 (where the protocol will be significantly re-engineered) are undertaken.

– *Status* –

This issue has *not* been addressed in version 1.5.0.

1.3 Context vs Input

Our early reports [4, 5] contain much discussion of the distinction between input vs context. One key point we raised was that the proposed distinction between trusted and untrusted components did not work very well when no component of the system is completely trusted. There is also currently roughly half a dozen JIRA tickets open, mostly

created by the examiners from BFH, which detail more specific issues related to this point.

– *Status* –

Post has indicated that this issue will see a major overhaul in version 2.0.

1.4 Auditor instructions

In our report [5] on version 1.3.* we stated that

We think that it is crucial to amend the documentation with a detailed description of the use of the verifier by the auditors at the operational level. The verifier’s sufficiency for universal verifiability is quite sensitive to how it is used. The voting system, like any engineered system, exhibits a tolerance between the specification, implementation, and operation; in other words, while the implementation and operation are largely the logical consequence of the specification there is a degree of imprecision, as seen in the code and the documented procedures. The level of variation in the voting system is such that without knowing ahead of time how the verifier will be operated there is an unacceptable risk that it will not ensure the universal verifiability of the system. Until this is addressed we believe that the systems should be considered non-compliant with Art.3.a and Art.3.c of the OEV.”

– *Status* –

This issue has been largely addressed with the exception that it remains unclear what the check that

the number of real and test voters corresponds to the expected one

is intended to achieve, we raised this specific point with Post again.

Post has stated that checking the total number of voters, rather than the number in each verification card set, is intentional. They also noted that the number of voters in each verification card set can be found in the election result report; which we confirm is correct. Analysis of the impact of *not* requiring this data to be verified requires more knowledge than we have regarding the relation between election events and verification card sets in the way in which the system is actually used.

– *Summary* –

The issue at stake is that an adversary (in control of the `setup component`) can move voters between voting card sets without this necessarily being detected. This is not, in our judgment, a violation of the required verifiability but does place additional trust in the `setup component` though simplifying the manual checks of the auditors.

1.5 Inconsistent naming of setup/configuration Verification

In our report on 1.4.* [2] we noted that:

It is unhelpful that the verification before voting commences is referred to as `VerifyConfigPhase` in system specification but predominantly as some variation of `Setup Verification` in the verifier specification and implementation. It would be simpler to either use the word configuration or setup but not both.

– *Status* –

This is planned for version 2.0.

1.6 Usage of algorithms and chunking

In our report on 1.4.* [2] we noted that:

The implementation, however, deviates from the specification by implementing chunking; this renders the information in Table 7 incorrect with regards to the implementation. For example, the implementation calls Alg. 4.7 once for each chunk making up the verification cards sets. This is an important factor in the security of the system since the sorting of the `CMtable` is done inside the algorithm. For clarity we believe that sorting `CMtable` on a per voter basis should be fine for security and consequently sorting per chunk should also be secure in practice but specification should be clear on this point.

– *Status* –

Chunking no longer affects the verifier since the relevant payloads are not sent anymore starting from version 1.5.0; however, the issue remains in the system specification and implementation.

1.7 Alignment of Voting Card Print Service

In our report on 1.4.5 [3] we noted that:

A2.2 of the OEV [8], lists the channel between `Setup component` and `Print component` as untrusted. The untrusted channel means the messages need to be encrypted to avoid leaking the relationship between candidates and return codes to the adversary; this would break both individual verifiability and privacy.

The question is, at the protocol level, is the `Voting Card Print Service` part of the `Setup component` or `Printing component`? In our discussions with Post they argued for the former and in our discussions with the Chancellery they argued for the latter. Post's position is contradicted by the channel security section of the system specification [11] but aligns with the current deployments we are aware of.

– *Status* –

We received an answer that the **Voting Card Print Service** is definitively part of the **Setup component**; however, if cantons were to change their printing office then different software achieving the same function might be used as part of the **Print component**. One concern is that an alternate deployment would need not only new software to replace the **Voting Card Print Service** but also changes to the **Secure Data Manager** which would need to start encrypting the relevant files.

1.8 Minor issues raised in 1.4.5

The following minor issues were raised in our report on 1.4.5, we briefly discuss the status.

Configuration Tool - README - Encryption Authentication

This issue has been rectified.

Configuration Tool - README - Broken link

This issue has been rectified.

Configuration Tool - Broken test data

This issue has been rectified; the test data now runs without error.

Configuration Tool - Signature status

This issue has been rectified; signature verification is enabled by default from version 1.5.0 and required from version 1.5.1.

2 Catalogue of Measures Reviews

In this section, we will go through the Catalogue of Measures items listed as resolved or partially resolved in the kickoff document for this round of review.

A.10 Reduce dependencies on external software

This is now the second examination round in which A.10 has been presented as resolved. We previously examined this in detail in version 1.4.* the results of which can be found in Appendix A of our corresponding report [2]; we only noted minor issues in the previous round.

Our recommendation to remove jSurfer from the verifier dependences has been implemented. There has been some turnover in the dependencies of the e-voting system but the total number of dependencies seem relatively stable since our last review. We note that the **Data Integration Service** is relatively liberal in its use of dependencies including obscure libraries like `swisspush:super-machine`.

We extended our analysis of dependencies to the `voting client`. Our analysis showed that the following dependencies:

- `ev-argon2-wasm@1.0.0`
- `ev-jssha@3.3.1`
- `rv-utf8@3.0.0`
- `vts@1.1.0`
- `@angular/router@19.2.14`
- `@ng-bootstrap/ng-bootstrap@18.0.0`
- `@ngx-translate/core@16.0.4`
- `@popperjs/core@2.11.8`
- `bootstrap-icons@1.13.1`
- `bootstrap@5.3.6`
- `marked@15.0.12`
- `ngx-markdown@19.1.1`
- `rxjs@7.8.2`
- `simplebar-angular@3.3.1`
- `tslib@2.8.1`
- `zone.js@0.15.1`

This is a much longer list than we were expecting and we reached out to Post to clarify. Post confirmed that while the dependencies into the cryptographic part of the voting client have been minimised, the dependencies into the UI part are significant.

– *Summary* –

We remain satisfied with the dependencies in the core part of system ; however, there appears to be significant room for improvement in the `Data Integration Service` and `Voting Client`.

A.11 Disclose source code of software for generating PDF files

We examined the disclosed `Voting Card Print Service` as part of the examination of 1.4.5; the issues raised have been resolved.

A.12 Further develop symbolic proofs

We did *not* analyse this issue.

A.15 Object-Oriented design in crypto-primitives

We did *not* analyse this issue.

A.21 Implement the dispute resolver

The aim of the dispute resolver is to allow resolving of disputes among the control components about what votes should be counted.

2.21.1 History of Issue

We raised extensive issues with dispute resolution in our first report [4]. These issues were significantly addressed by the addition of the `Long Vote Cast Return Codes allow list` which allows the `CCs` to check that the other `CCs` have received the correct confirmation key before they release their share of the `long Vote Cast Return Code`. There were two main attacker scenarios we considered either:

- to return the correct VCC_{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 BCK_{id} .

We also commented on the possibility of message reordering attacks and divergence in the logs.

2.21.2 Currently Implemented Solution

The dispute resolver is now detailed in the specification [12]. It is triggered if the `MixDecryptProcessor` aborts due inconsistent `ControlComponentVotesHashPayloads`.

- each `CC` sends their view of the election event and verification cards calculated by `ExtractElectionEvent` and `ExtractVerificationCards`;
- the dispute resolver checks consistency of election events by comparing the hash computed by `GetHashExtractedElectionEvent`;
- the dispute resolver checks that the received ballots are the same by computing the hash of the verification card id, verification card, set and ballot `E1`;
- the dispute resolver creates a new list of confirmed ballots including all ballots for which evidence has been provided matching the allow list;
- the dispute resolver sends this list back to the `CCs`;
- each `CC` goes through the new list and adds to its own list any missing ballots, provided evidence matching the allow list is provided.

2.21.3 Under-defined specification and mistaken vulnerability

During the course of our investigation we erroneously reported a vulnerability in the dispute resolution process based on a misunderstanding of the behavior of `GetHashExtractedElectionEvent`. Part of that report also covered issues in the specification of the dispute resolution protocol. Specifically, we asked for the following changes to 6.4 `CheckExtractedElectionEventConsistency`:

- the extracted election events should be listed as input rather than context since they are not deterministic in nature, have not yet seen mutual acceptance among participants, nor are they compared against external public data. *Post made this change in 1.5.1.*
- on line 2, `eeej` should be listed as input to `GetHashExtractedElectionEvent`.

2.21.4 Summary

The formalization of the dispute resolver makes use of the `Long Vote Cast Return Codes allow list` in the expected way to check if evidence is available that a vote was confirmed. This is essentially the working out of Option 2 in our original recommendations. The basic logic of the solution is that:

- the voter can only see the `VCCid` if the honest control component releases it's share and
- the honest control component will only do this once they have sufficient evidence that the ballot should be counted as confirmed (by all the control components).

Minor Comments The implementation of dispute resolution would abort if no vote is marked as confirmed by `CheckVoteConfirmationConsistency`. While it might be unusual for no ballot to be confirmed in a verification card set, it is unclear why this should be a failure of dispute resolution except so much as it shows that one of the control components listed ballots as confirmed while they should not have been. However, if the aim is to check that no control component listed ballots as confirmed without valid evidence this should be checked directly.

– *Status* –

Post has updated 6.4 `CheckExtractedElectionEventConsistency` with some of our suggested changes while leaving others to 2.0; where the distinction between context and input will be removed. Post also made changes to the implementation to address our minor comments.

There was significant discussion by the examiners regarding the role and purpose of the dispute resolver, particularly as—in the current form—it would be possible to have the control components directly implement this functionality without involving a third party. We agree that the current approach seems suboptimal in terms of the complexity of the dispute resolution process relative to the scope of disputes it can resolve but we would encourage consensus on the role and scope of the dispute resolver before further implementation changes.

A.22 No operational tasks for Auditors

In Version 1.5, the tasks performed by the auditors during configuration have decreased but some tasks remain. It was unclear to us why only the tasks removed should be considered “operational tasks” since the tasks which remain still relate to “cryptographic parameters for the ballot.”

We created a ticket to clarify the intent of A.22 with the response that:

If the verifier is used for verifications that are not strictly needed (given the trust model), that is fine. If the verifier is used for verifications that must be done in order to satisfy security requirements for which the verifier may not be considered to be trusted INSTEAD of using a trusted component for that verification, this would contradict A.22.

We did not see any concrete issue in our analysis (Appendix A); nevertheless the gap between the specification and implementation is unusually high in regards to the control component actions at the end of the setup phase; see our recommendation to address this below. We also note that there is significant room to constrain the actions of a malicious `setup` component though this is not required by the security properties.

– *Recommendation* –

Consolidate the various checks that the control components should run to finalise the setup/configuration phase into a pseudocode algorithm.

– *Status* –

Post has acknowledged the issue and plans to address it as part of 2.0.

A.24 Improvements in computational proof

This measure requires the update of the cryptographic (computational) proof in order to show a sketch of a valid reduction from the security of the e-voting protocol to the underlying cryptography. We are unsure as why this was listed as partially addressed since the changelog indicates no progress regarding this recommendation. We remain of the view that delaying the addressing of this measure until A.05 is implemented is a sensible decision.

A.25 Improvements in specification and code

A.25 calls for an improved specification, specifically with regards to a list of known issues. Various changes have occurred, particularly around the verifier, which make the specification more complete. We extracted the list of the subpoints to this measure from the Annex to CoM.

At present it appears that many of the recommendations have not been implemented and so this measure should not be viewed as resolved. There were a number of specific issues which were claimed to be resolved and in these cases we agree that resolution has indeed occurred.

– *Status* –

We understand that this issue is still open and will delay our analysis until Post believes it is resolved.

3 New Examinations and Issues

In this section, we will discuss various new examinations; in some cases these relate to new functionality or changes in the system, whereas in others the item examined existed in prior releases but we had not examined it.

3.1 Update to the architecture document

We provide discussion on various points related to the architecture document.

3.2 Technical Constraints This list includes various actual constraints such as testability and reproducibility with various choices of programming languages and frameworks; the inclusion of the latter as constraints seems unwarranted. To be clear we think that these are fine design choices, just not constraints.

Figure 3 This figure appears outdated. `Domain` should not exist as an entity independent of `E-voting-libraries`; even if there was a strong reason to do so `Verifier` would then depend on `Domain`. This issue also occurs in Section 5.1.1 and Figures 4 and 10.

9.2.4 BigIntegers It is not immediately clear why the less precise `BigInteger.signum() >= 0` should be preferred over `BigInteger.signum() == 0` when comparing with zero; of course, these should be equivalent since we don't expect any biginteger to be negative.

9.3.6. Use appropriate data type and structures It appears that the words “don't” and “do” in the example are flipped from the intended order.

Springboot vs Spring boot Currently this framework is referred to inconsistently, we suggest choosing one and using it consistently.

The performance objective of SetupTally The choice of 4 hours as the performance objective seems strange since the vastly more computational intensive `SetupVoting` aims for 8 hours.

– *Status* –

Post has informed us they intend to address these points in version 1.6.

3.2 Changes to ZKP proof verification

In version 1.5.0 the division of responsibility of ZKP proof verification was changed with proofs produced in `GenEncLongCodeShares` now being verified by the `setup` component in `CombineEncLongCodeShares` rather than the verifier in `VerifyConfig`. Our aim here is to analyze the security impacts of this transfer.

Recall that the setup phase is responsible for generating the various codes and key material which will be required later in the election; this is particularly crucial for the privacy and individual verifiability of the system. The transfer of these tasks to `setup` component was part of measure A.22. We analyze this issue by going through each check previously performed by the verifier and arguing why it is no longer necessary:

VerifyChunkConsistency Checked that the chunk ids received were monotonically increasing. This same check by the setup component occurs in `VerificationCardSetDataGenerationService`.

VerifyEncryptedPCCExponentiationProofsVerificationCardSet Checked the exponentiated, encrypted, hashed partial Choice Return Codes and associated exponentiation proofs. These are now checked in `CombineEncLongCodeShares`.

– *Recommendation* –

In Alg. 4.6 `CombineEncLongCodeShares`, `sksetup`, `cpcc`, and `cck` should be listed as context since they come, or at least should, from the internal view of the `setup` component; this would also mean updating Alg. 4.7 and Alg. 4.8 accordingly. This seems to be a typo which emerged from copying the algorithms from the verifier specification to the system specification without fully making the corresponding changes.

3.3 Simplify verification of Maurer’s proofs

Finding “EX-324: 3.2.1. Crypto-Primitives Clarified” states that, for efficiency reasons, and using notations from Section 10.1 of the Cryptographic Primitives document (v. 1.5.0) [9], the computation of $c' = x \otimes y^{-e}$ is realized as $c' = x \otimes (y^{-1})^e$, which of course makes sense when e is much smaller than the order q of \mathbb{G}_1 .

– *Recommendation* –

It could be more direct to modify the proposed implementation of the Maurer protocol by computing $z = b \star w^{-e}$ in \mathbb{G}_1 and then $c' = x \otimes y^e$ in \mathbb{G}_2 . In practice, this saves the computation of a multiplicative inverse modulo p and replaces it with a subtraction in place of an addition modulo q .

This could benefit Algorithms 10.2, 10.5, 10.8, 10.11.

This change appears to be fully compatible with the paper of Maurer from which the current protocol is designed. The Maurer paper talks about a challenge space C , which must be big enough but can be any subset of $[0, q]$ (see [7, Sec 6.1]). The proposed change effectively defines a challenge space $[q - 2^L, q(= 0)]$ instead of the current $[0, 2^L[$ (where L is the output length of the hash function).

– *Status* –

Swiss Post informed us that this will be considered for future versions.

3.4 Change to allow voters to explicitly abstain from elections and votes

We have been advised that version 1.5 would include the ability for voters to explicitly abstain from the election; this feature has been delay till version 1.6.

3.5 Agreement Algorithms - in system specification

Agreement among the participants is crucial and we think that raising this issue to its own subsection of the system specification is very beneficial for security. In our previous report [5] we argued that:

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

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

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

– *Status* –

The documentation of the data sent is now largely present in Section 7 of the system specification. We briefly repeat the relevant data below:

- `ElectionEventContextPayload`
- `ControlComponentPublicKeys`
- `SetupComponentPublicKeysPayload`
- `SetupComponentVerificationDataPayload`
- `SetupComponentTallyDataPayload`

The new `GetHashElectionEventContext` algorithm ensures that the parties share a consistent view of the the `ElectionEventContextPayload` with the following exceptions:

- the `seed`
- the `small primes`
- the `votesTexts` and `electionsTexts`

Since none of these values appear to be directly used by the the control components, this does not seem to have any impact.

3.5.1 Remaining Scope for Improvement of Consensus

The current solution seems sufficient to address the issues we had previously raised. However, there are still more complicated consensus issues which could occur. As an example, we raised the possibility of different honest components having a different view on what version of the protocol is being used for a specific election. This is further discussed in Section 3.13.

3.6 Data Integration Service

The `data integration service` is used to generate `anonymised-configuration.xml` which is used by the e-voting system. Much of the information produced by this tool is crosschecked later in the system. Our relatively shallow investigation did not find any issues in this tool.

3.7 VOTING Stimmunterlagen

We examined the VOTING `stimmunterlagen`, specifically the following three repos:

- <https://github.com/abraxas-labs/voting-stimmunterlagen-offline-client-docs>
- <https://github.com/abraxas-labs/voting-stimmunterlagen-offline-client-app>
- <https://github.com/abraxas-labs/voting-stimmunterlagen-offline-client-shared>

Our focus was mainly on the cryptographic libraries, we had the following findings:

- The security of the signature verification in `Voting.Stimmunterlagen.OfflineClient.Shared.Cryptography.Decryption` depends upon the default fallback behavior of the data converters. A similar issue occurred in older version of the Post code but has since been patched. We recommend using the data converters with the explicit fallback behavior of throwing an exception; the vendor has stated they plan to implement this in the next release.
- At present the logic in `ValidateFileSignature` method of `FileDecryptor` does not validate the length of the signature. Depending on the permissiveness of the implementation of the signature verification in the underlying library, this may allow message extension attacks on the `cryptoFileBytes`; we have not tested to see what the rest of the `FileDecryptor` would do with this malicious payload. The vendor has consulted their expert and concluded that there is no vulnerability here; we were not provided with any reason to substantiate this claim and have not investigated further.

3.8 Direct Trust Tool

In previous reports, we have detailed examinations of the `file-cryptor` and `xml-signature` tools; we also examined the `dispute-resolver` in this report. The remaining tool we have never examined is the `direct-trust-tool`.

The `direct-trust-tool` is used generate the keystores used to secure communication in the rest of the protocol. We examined the following functionality:

keystores-generation

public-keys-sharing-download

public-keys-sharing-import

public-keys-fingerprint

keystores-download

The only possible issue we saw was in the key fingerprints displayed to the user, to facilitate checking that the correct keys were imported; these fingerprints were based off the contents of `componentStorageRepository` and it was not immediately clear if that will actually be consistent with the contents of the keystores.

– *Status* –

Post has confirmed the issue and addressed it in version 1.5.1 with the fingerprints now based off the imported values.

3.9 Examination of id consistent within the system

One complexity of the system not addressed in the security proof is the support for multiple concurrent elections; we examined a few adhoc questions around the consistency of the ids denoting which election particular voters and ballots belonged to.

What enforces that the context ids in an encrypted verifiable vote entity match the verification card entity?

These are crosschecked by the `IdentifierValidationService` when the `PartialDecryptProcessor` receives the encrypted verifiable vote entity.

There doesn't seem to be any agreement on which verification card ids belong to which verification card set, does it matter?

The control components generate their view of the verification card ids based on payloads received by the `GenerateEncryptedLongReturnCodeSharesProcessor` from the setup component; the verifier receives similar information in the `SetupComponentTallyDataPayload`.

A dishonest setup component could send inconsistent payloads to the components and there is no direct check which would detect this; however, since the setup component is only untrusted for universal verifiability we only need to consider attacks in that category. The verifier runs `VerifyVerificationCardIdsConsistency` which checks that verification card ids with confirmed ballots are consistent with the view it received from the setup component. This seems to suffice for universal verifiability.

3.10 Complex ballots and individual verifiability

It is often implicitly assumed when defining individual verifiability that voters know what a valid ballot looks like and intends to cast a valid ballot; this assumption does not work well in situations where the ballots are complicated and the voter may not even have a predefined intent of what complete ballot to cast. The individual verifiability formalisation used in Version 1.4.0 of the “Protocol of the Swiss Post Voting System - Computational Proof” document assumes that the voter is trying to verify the submitted ballot against a well-formed partial vote; while the formalisation makes this assumption, Post is clearly aware of the issue and argues separately about vote compliance, which we will discuss again shortly.

An adversary might attempt to break individual verifiability by:

- encouraging a voter to submit an invalid ballot (or simply cast one on their behalf),
- letting a voter successfully check that the ballot matches their intent,
- having the voter’s ballot filtered out later as invalid.

Since the definition of individual verifiability used in Switzerland also requires the protections to apply on partial votes, the attack above can be simplified to the voting device casting an invalid ballot without detection by a voter who doesn’t check the part of the ballot which is invalid.

There seems to be two main ways to prevent such an attack:

1. the honest component storing the ballot can check that the ballot is valid before storing it,
2. the verification mechanism can warn the voter that the ballot is invalid.

The first of these two options is used in the Swiss Post system with the honest control component checking the received ballot against the allow list. However, at present this mechanism has a caveat. It does not stop an invalid vote for a party list lacking candidates, an issue which can only occur in certain elections.

– *Summary* –

At present for certain elections, the system does not have individual verifiability when the adversary submits a specific kind of invalid ballot. The Catalogue of Measures item A.26 calls for addressing this specific case but does not note the effect of this issue on individual verifiability. This issue is also discussed in Section 11.4 of the Computational Proof, which notes the relationship between vote compliance and individual verifiability.

3.11 Points Arising from Revised Security Analysis

We reexamined the security analysis of the system from our prior work in [2] and [5]; this highlighted the various positive changes in the system which have made it easier to see why the system should be secure.

Here we will include suggestions on improvements based on the analysis; in general we think that implementing these suggestions in version 1.* of the protocol is likely not a good use of time; however, they should be considered when designing version 2.

Improved Key Hygiene One issue which significantly complicates the analysis of individual verifiability is the reuse of keys for different purposes; for example pk_{setup} is reused for encrypting both $\text{pCC}_{\text{id},k}$ and hCK_{id} ; the analysis would be easier if a different key was used for these two purposes. The same is true of the verification card secret keys. A positive example in the code is the `control components` use of different keys for the voter specific Choice Return Code Generation and Vote Case Return Code public key.

Separate Data Structures for Secret Values At present it is common for secret values to be packaged into the return types of algorithms. In most cases, the secret values only need to go to one place but the overall type is handed over to many functions; someone analysing the security of the system needs to check that the secret value is not used in these functions. For example, `persistPreComputationPayloads` passes the voter’s secret key k_{id} to five functions but only one of these needs the key.

3.12 RecursiveHashOfLength efficiency

Algorithm 5.7 `RecursiveHashOfLength`, in the Cryptographic primitives (v.1.5.0) [9] hashes values with an output length ℓ at every step (Lines 8, 10, 12, 17).

This appears to have at least three downsides:

1. It can be inefficient, especially when values must be hashed to a length much larger than ℓ^* (as in `HashAndSquare`, which is used a lot in the voting protocol).
2. It creates the requirement that $\ell \geq \ell^*$, which looks artificial (it could make sense, in some situations, to look for short hashes).
3. This algorithm seems largely redundant with Algorithm 5.5 (`RecursiveHash`).

– *Recommendation* –

Would it make sense to follow the approach of HKDF described in Section 5.4, that is, start by an “extraction” phase that hashes everything to a fixed length, then an “expand” phase that transform this fixed length output into something of the desired length?

For instance, `RecursiveHashOfLength` could be defined as

1. the application of `RecursiveHash`, followed by
2. the application of HKDF-Expand, using the output of `RecursiveHash` as *PRK*.

This seems to address the downsides listed above:

1. Now, at each step, values are hashed to length ℓ , which matches the security requirements

2. The `RecursiveHash` function works with a sufficient security level even if the required final output length is less than ℓ^* .
3. It removes the need to implement two recursive hashing functions, that essentially differ by the choice of hash functions.

– *Status* –

Swiss Post indicated that this will be considered for a future version.

3.13 Add voting system version into `GetHashElectionEventContext`

Algorithm 3.11 `GetHashElectionEventContext` of the System Specification (V.1.5.0) does a good job at hashing lots of elements defining an election event. As we start having many versions of this voting system, and more versions coming, there is a concern that the semantics of some elements may be changing from one version to another.

– *Recommendation* –

Tie `GetHashElectionEventContext` to the version of the system specification (and possibly system code version) against which the hash is computed. This could be as simple as adding a “System Specification Version 1.5.0” string into the inputs that are hashed in `GetHashElectionEventContext`.

Our goal here is to protect from so-called chosen protocol attacks [6], or multi-protocol attacks [1], which do not seem to be envisioned in the current security analysis, and would probably be very complicated to handle if the protocol does not take explicit steps to prevent mixing messages coming from different versions of the system.

A full solution may require changes that are more important than adjusting the inputs of `GetHashElectionEventContext`, but that adjustment would seem to be a simple step in a useful direction.

– *Status* –

This issue is in discussion.

3.14 Include key index in Schnorr proofs?

Algorithm 4.4 `GenKeysCCR` of the System Specification (v.1.5.0) ties every computed Schnorr proof to the index j of the CCR computing the proof, which is a good idea. However, there is nothing tying a proof to a specific key index, and the proofs therefore do nothing to prevent reordering some keys, which could be bad. This may be prevented elsewhere in the system by other means, but there is an easy opportunity to strengthen the operations here, while being consistent with the inclusion of the index j that is already there.

– *Recommendation* –

Modify Algorithm 4.4 by computing, between the current lines 3 and 4, an i'_{aux} value defined as $i_{aux} || IntegerToString(i)$, and using i'_{aux} in the `GenSchnorrProof` function.

– *Status* –

Swiss Post indicated that this will be considered for a future version.

3.15 Ask the CCR/CCM to run VerifyConfigPhase?

Section 4.3 of the System Specification explains that Auditors can optionally run the VerifyConfigPhase at the end of the Configuration Phase. It is a good idea to do so, but it is optional only, which makes sense with respect to the auditors role.

– *Recommendation* –

Would it make sense to ask the CCRs (or the CCMs) to always run VerifyConfigPhase? Given that at least one of them is honest, this would make sure that, under the ordinance threat model, the system does not start collecting votes in a state in which an invalid configuration exists and is undetected because no auditor verified the configuration.

– *Status* –

Swiss Post clarified that, in the current implementation, the auditor’s run of the VerifyConfigPhase algorithm is not optional, and that it is a design choice that the control components do not enforce or perform this verification step, for efficiency reasons.

We are not sure that we understand why it would be really damaging to perform these verification steps, at any time before the election starts, so that a configuration issue can be detected before any vote is cast: we assume that the configuration phase is not executed minutes before the election?

3.16 Explanation of failure modes in the dispute resolution

The high level idea of the dispute resolution process, as depicted in Fig. 15 of the System Specification, looks quite natural. We are a bit surprised by the behavior of Algorithms 6.4 and 6.5, though: they seem to declare a failure very quickly, and we guess from the remark on p.119 that such a failure would lead to a “comprehensive forensic investigation”.

There seem to be various failure modes that could be solved in a systematic way, though, and we suspect that OEV 11.11 would require explaining these.

For instance, one compromised CCR could have all its data erased. But, if all the data related to the recorded votes (ciphertexts, decryption proofs, etc.) from this CCR remain available on other CCRs with the proper signatures, and if these other CCRs all agree on their views, then this seems to offer evidence that the CCR that does not report anything is indeed compromised, and that the votes recorded by the other CCRs should be kept for the mixing phase.

(We certainly agree that, if there is evidence that a CCR/CCM is compromised, then a comprehensive forensic investigation would be very important in order to understand how this compromise happened and the extent of it. But this investigation does not seem to remove the question of deciding how the election should be tallied, within the Ordinance security model, which we understand to be the goal of the dispute resolver.)

– *Recommendation* –

Two questions/suggestions:

1. Would it be possible to clarify in the System Specification document what is actually happening if Algorithms 6.4, 6.5 and 6.7 declare a failure (\perp)?
2. If the answer is: a “comprehensive forensic investigation”, would it be possible to explain why the failures that lead to that recommendation cannot be solved by the dispute resolver in a systematic way, based on the protocol data available to him?

– *Status* –

Swiss Post clarified that, in case of a failure of Algorithms 6.4 or 6.5, the recommendation would be to cancel the electronic election event and conduct a forensic investigation.

We recommend Swiss Post to clarify that cancelling the electronic election event is the recommended action when one of these verification algorithms fails. We think that, if such a decision ever needs to be made, a written document supporting it would be very useful.

3.17 Miscellaneous observations

Byte notation consistency We noticed that the notation for bytes used in [9, Algorithm 3.1] was inconsistent. This was fixed in Version 1.5.1.0 of the cryptographic specification [10].

Why shortening the salts? In Algorithms 3.19, 3.20, 5.1, 5.2 and possibly others, when a salt is computed, it is trimmed to 128 bits, which is the default recommendation in the Argon spec. We do not see any advantage in shortening this salt to 128 bits if a 256 bits salt is available. A 256 bits hash makes the code simpler (by removing the need to use the `CutToBitLength` function) and can only add a bit of security and the speed difference should be imperceptible (and even if there were a speed difference, the whole purpose of using Argon is to have a slow hash function, so slowing things down would be good). Swiss Post has indicated its preference to stick to the default recommendation of the Argon specification.

Where are the hPW_i computed? Section 4.2.2. of the System specification document (v.1.5.0) explains that “the setup component sends non-invertible hashes of the electoral board passwords to the Tally control component”. We guess that these passwords are the hPW_i values visible at the bottom of Fig. 9. But we do not see where these hPW_i values are computed and how. Swiss Post largely clarified this in Version 1.5.2 of the Specification.

Algorithm 6.8 UpdateConfirmedVotingCards never returns \top In the System Specification (v.1.5.0), Algorithm 6.8 `UpdateConfirmedVotingCards` is claimed to output \top if the update was successful. However, Algorithm 6.8 does not have a line that returns \top . Swiss Post addressed this problem in Version 1.5.1.

What is implicit agreement? At the bottom of Page 79 of the System Specification (v.1.5.0), it is written that each Control Component “confirms that all parties implicitly agree on the election event context”. What does it mean to “implicitly agree”? More generally, would it make sense to have an algorithm that explains

precisely what is done regarding the 3 items at the bottom of p.79 and top of p.80? In Version 1.5.1, Swiss Post explained what was meant by this agreement, and deferred the other clarifications to a future Version 2.0.

4 References

- [1] Cas Cremers. Feasibility of multi-protocol attacks. In *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES 2006*, pages 287–294. IEEE Computer Society, 2006.
- [2] Thomas Haines. Final report on releases 1.4.0 to 1.4.3. https://www.bk.admin.ch/dam/bk/en/dokumente/pore/E_Voting/Examination_reports_August2024/Scopes%20%20and%20%20Final%20Report%20Thomas%20Haines%2011.07.2024.pdf.download.pdf/Scopes%20%20and%20%20Final%20Report%20Thomas%20Haines%2011.07.2024.pdf, July 2024.
- [3] Thomas Haines. Final report on release 1.4.5. https://www.bk.admin.ch/dam/bk/en/dokumente/pore/E_Voting/Examination-reports-June-2025/Scopes%20%20and%20%20Final%20Report%20Thomas%20Haines%2019.06.2025.pdf, July 2025.
- [4] Thomas Haines, Olivier Pereira, and Vanessa Teague. Report on the swiss post e-voting system. <https://www.news.admin.ch/newsd/message/attachments/71147.pdf>, March 2022.
- [5] Thomas Haines, Olivier Pereira, and Vanessa Teague. Remarks on selected elements of the swiss post e-voting system versions 1.2.3, 1.3, and 1.3.1 final version. https://www.bk.admin.ch/dam/bk/en/dokumente/pore/E_Voting/Examination_reports_August2023/Scopes%20%20and%20%20Final%20Report%20Thomas%20Haines,%20Olivier%20Pereira,%20Vanessa%20Teague%2031.07.2023.pdf.download.pdf/Scopes%20%20and%20%20Final%20Report%20Thomas%20Haines,%20Olivier%20Pereira,%20Vanessa%20Teague%2031.07.2023.pdf, July 2023.
- [6] John Kelsey, Bruce Schneier, and David A. Wagner. Protocol interactions and the chosen protocol attack. In *Security Protocols, 5th International Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 1997.
- [7] Ueli Maurer. Zero-knowledge proofs of knowledge for group homomorphisms. Available from <https://crypto.ethz.ch/publications/files/Maurer15.pdf>, 2015.
- [8] Swiss Federal Chancellery. Federal chancellery ordinance on electronic voting of 25 may 2022. Available from <https://www.fedlex.admin.ch/eli/cc/2022/336/en>, May 2022.
- [9] Swiss Post. Cryptographic primitives of the swiss post voting system – version 1.5.0. <https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives/-/blob/crypto-primitives-1.5.0/Crypto-Primitives-Specification.pdf>, June 2025.

- [10] Swiss Post. Cryptographic primitives of the swiss post voting system – version 1.5.1. <https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives/-/blob/crypto-primitives-1.5.1/Crypto-Primitives-Specification.pdf>, August 2025.
- [11] Swiss Post. Swiss post voting system – system specification – version 1.4.2. <https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/b5fc2da60f373612bc887138966c984da4487dfa/System>, 2025.
- [12] Swiss Post. Swiss post voting system – system specification – version 1.5.0. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/a81327f165be40b10a2e4c82dc381abcfda3be37/System/System_Specification.pdf, 2025.

A A.22 Analysis

This section contains our analysis of Measure A.22; specifically we aimed to assess if any of the checks performed by the `verifier` in `VerifyConfigPhase` are crucial for a security property where the `verifier` is not a trusted component.

During `VerifyConfigPhase` the `Verifier` receives the following:

- `electionEventContextPayload.json`
- `configuration-anonymized.xml`
- `controlComponentPublicKeysPayload.${j}.json`
- `setupComponentPublicKeysPayload.json`
- `verificationCardSets/${vcs}/setupComponentTallyDataPayload.json`

The completeness and authentication checks on these payloads are not relevant to the rest of the system since the other components which expected to receive these payloads check the signatures for themselves. Since pk_{CCR} is not relevant for security, the related checks are also not security critical. The analysis related to EL_{pk} is complicated by the fact that the specification does not have a single algorithm that the control components run to finalise the setup/configuration phase but rather informal details are scattered across the document (bottom of page 66, bottom of page 77, pages 79 and 80). We would like to see these formalised to reduce the gap between the implementation and specification but the intent is clearly there.

The summary of our investigation is that the measure has been addressed. We go into more detail on the following questions:

A.1 Is the implementation sufficient to ensure the public key is adequately validated?

The aim of this question is to check that `VerifyElectionPublicKeyConsistency` and `VerifySchnorrProofs` are unnecessary.

Specification On page 66, it is stated that all control components should run `VerifyKeyGenerationSchnorrProofs` at the end of `SetupTally` protocol. `VerifyKeyGenerationSchnorrProofs` runs checks equivalent to those in `VerifySchnorrProofs` in the verifier specification. The `SetupTally` protocol (Fig. 8) shows the CCMs receiving both the key shares and the final key. A check similar to `VerifyElectionPublicKeyConsistency` could therefore be performed to ensure the final key is the product of the key shares. We cannot see any part of the system specification which asks the control components to perform this check; neither is there any check in the specification that the control component key shares originate with the claimed control component.

Implementation The omitted consistency check in the specification does occur in the implementation, in the constructor of `SetupComponentPublicKeys` (line 94).

Summary In neither case do these checks appear strictly necessary since the payloads originate with the `setup` component who is trusted for privacy.

A.2 Is the specification/implementation sufficient to ensure the encryption group is consistent and correct?

The aim of this question is to check that `VerifyEncryptionGroupConsistency` and `VerifyEncryptionParameters` are unnecessary.

Specification Differences in the encryption group will be caught no later than `VerifyKeyGenerationSchnorrProofs`. `SetupVoting` (Fig. 7) includes the sending of the seed alongside the encryption group to the `control` components; this suggests an intent to validate the encryption group against the seed. However, we cannot see any check in the system specification which would actually do so.

Implementation We can also see no such check in the implementation.

Summary A weak encryption group could break all three main security properties of the system; however, for individual verifiability and privacy we can assume the `setup` component correctly generated the group and for universal verifiability we can assume the verifier correctly verified the group.

A.3 Is the specification/implementation sufficient to ensure the verification card set ids and verification card ids are consistent across the control components?

The aim of this question is to check that `VerifyVerificationCardIdsConsistency` and `VerifyVerificationCardSetIdsConsistency` are unnecessary.

Specification and Implementation The `GetHashElectionEventContext` used in `VerifyKeyGenerationSchnorrProofs` ensures a consistent view of the verification card set ids.

A.4 Do the control components adequately verify the prime mapping table?

The aim of this question is to check that `VerifyPrimesMappingTableConsistency`, `VerifySmallPrimeGroupMembers` and `VerifyVotingOptions` are unnecessary.

Specification and Implementation The second and third checks in `VerifyPrimesMappingTableConsistency` would require the `configuration XML` to be sent to the `control` components to replicate. We could not see any check in the specification or implementation which mirrors the first check in `VerifyPrimesMappingTableConsistency`. We cannot see any checks analogous to `VerifySmallPrimeGroupMembers` or `VerifyVotingOptions` in the specification or implementation.

Summary Since the `setup` component is trusted, this does not seem to matter.

A.5 Is `VerifyTotalVotersConsistency` unnecessary?

The aim of this question is to check that `VerifyTotalVotersConsistency` is unnecessary. This check is only relevant to universal verifiability for which the `verifier` is trusted.