Second Addendum on the Swiss Post e-Voting System*

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

February 13, 2023

Contents

*This is a slightly expanded version of our original second addendum

1 Summary

The system continues to improve and the December 2022 release fixes many of the outstanding issues, but not all. The system has reached a level of maturity where we no longer readily find exploitable vulnerabilities in the code by looking in the obvious places. From our perspective, the main risk which might result in the system not offering the security required by the Ordinance is the ad-hoc nature of the mitigations; the system has been patched significantly during the review process and now protects against the attacks we, and others, have found. However, it is not clear that the system does provide the required level of security consistently, especially when considering edge cases where normally trusted components are untrusted; for example, universal verifiability where the setup component is untrusted.

The security proofs have also been vastly revised, documented, and improved in the December 2022 release. While some proofs can now be evaluated thanks to being sufficiently documented, this is not the case of others that have not been revised in the same way. The security reductions that have now been added in some places are particularly important for the evaluation. However, this task remains largely ongoing: the latest changes in the protocol remain only partially reflected in the security proofs and, whenever security reductions are available, we found difficulties/gaps in the proposed arguments. In both cases, we suspect that lemma/theorem statements will need to be revised. The problems we found fortunately do not seem to translate into attacks against the system. But, it is most likely that similar issues arise in the other security proofs where reductions remain currently unavailable, and their impact is hard to predict. As such, we cannot say that the security proofs bring the level of confidence that they should provide.

All documents referred to in this report belong to the December 2022 release unless otherwise clarified.

2 Update on Scope 1

There has been much improvement within Scope 1, particularly to the amount of detail given about game hops in the proof document. There have also been improvements to the other documents, but gaps remain and are discussed later in this section. The main outstanding issue with regards to Scope 1 is, in our opinion, the compliance with requirement 2.14.1 of the Ordinance.

> 2.14.1 A symbolic and a cryptographic proof of compliance must demonstrate that the cryptographic protocol meets the requirements in Numbers 2.1–2.12.

We will discuss why we think the cryptographic proof does not demonstrate the compliance of the protocol with the requirements below.

2.1 Alignment between security model required by Ordinance and one used in proofs

The quality of the proof document version 1.1.0 has vastly improved over version 1.0.0. We will not provide an exhaustive list of all the improvements but will focus on the remaining issues.

2.1.1 Role of auditors

We have questioned if the use of the Auditors in the setup phase of the system is in line with the Ordinance. Post's most recent response was,

> We agree that future versions of the protocol could delegate more verification work to the control components. However, we believe - after discussion with the Federal Chancellery and the cantons – that we can use the auditors to check the configuration phase's proofs and that our current protocol is in line with the Ordinance.

We still cannot see how the use of the auditors in the system aligns with the written text of the Ordinance; the communication channels described in 2.2 for example don't seem to allow this use. A similar issue is noted, see quotation below, in Section 11.4 of the proof document, which notes that the privacy game assumes a trusted Auditor even though this contradicts the Ordinance.

> The computational models for individual verifiability and vote privacy assume a faithful execution of the VerifyConfigPhase algorithm. However, this assumption does not align with the trust model for individual verifiability (see section 2.2) and vote privacy (see section 2.4), which assume a trustworthy setup component, but an untrustworthy auditor... The cantons run the setup component and the verifier in a similar, secure offline environment. Thus, we consider the verifier and the setup component equally secure and trustworthy in the configuration phase.

We consider there to be a clear discrepancy as Post acknowledges in their documents. It remains unclear to us how the auditors will be chosen and will operate; this being a Cantonal responsibility the Ordinance and system are both fairly vague on the details. Our concern here remains that the threat model is unclear at this point, as is the distinction between the Auditors and Setup Component; we note for example Figure 22 and more generally Section 7.2 of the architecture document which seems to suggest that the Verifier, Setup Component, and Tally Control Component are all controlled by the canton. Given the crucial role of both the Setup Component and Auditors in the system, we suggest that bringing clarity to this issue would be of value.

2.1.2 Roles of tally control component and electoral board

We had previously noted the inconsistencies in the treatment of the Tally Control Component and Electoral Board in the proof document. This has now been

clarified with the new version of the proof document saying:

> For voting secrecy, we model the tally control component, together with the Electoral Board, as a single Control Component.

This model is consistent with the proofs but provides a substantial weakening of the claimed security property. By modeling the Tally Control Component, together with the Electoral Board, as a single Control Component the analysis only considers what happens if all these parties are honest and not what happens when some of these parties misbehave but others don't. For example, if the Electoral Board or the Tally Control Component is honest, but not both, then the system does not provide privacy.

2.1.3 Weakness in the security definitions

There remain many points where details of the system (even at the specification level) have been simplified out of the security models and proofs. We summarise a few below:

Chunking The setup phase is run in chunks rather than all at once as the proof assumes; Specifically, algorithms 4.3 GenVerDat and 4.4 GenEncLongCodeSharesj are called several times, on a subset of eligible voters each time. The results are then combined before algorithm 4.5 CombineEncLongCodeShares is called. This means that some tables and allow lists are sorted on a per chunk basis rather than globally as the proof model assumes; this does not appear to break anything in practice.

Verification Card Sets Both the specification and proof do not adequately capture that multiple verification card sets with different voting rights are being processed at the same time. For example, in the definition of `CompliantVote` both the correctness id and allow list depend on the verification card set id, but this is not made explicit.

This simplification also has an impact on privacy; the proof document presents a definition which shows that the system does not leak much more than the set of all confirmed votes, but depending on how the ballot boxes are set up, these sets (of votes) may be tiny. We are insufficiently familiar with the current status quo in Swiss elections to evaluate the real world impact of this but we encourage it to be carefully considered.

2.2 Missing elements in verifier specification

In our previous reports we mentioned missing elements in the verifier specification; we were particularly concerned around the lack of description of the manual checks the auditor was required to do and the resulting security vulnerabilities if these checks were not completed. Post has addressed this by adding algorithm 0.01 which specifies (informally) the checks the auditors need to manually perform.

The current list of manual verification checks seems to be heading in the right direction but is too rough and informal to be evaluated for security. We would like to see an algorithm which clearly specifies where the inputs come from and what checks are required; for example, what specific fields in what files need to be checked and what checks need to be made on those fields. It would be particularly useful to be clear about what inputs come from the existing files and what manual information the auditors need to receive in order to complete the checks.

2.3 Context verification

In our most recent report we summarised our position on input and context verification by saying:

> We would...encourage making clear what the source of the input is supposed to be and any corresponding consistency checks but the main step forward require(d) is to ensure the code actually...does ensure the context comes from a trusted source.

We received the following specific responses from Post:

> We improved all the mentioned algorithms and made sure that the context variables stem from a trusted source. Moreover, we implemented a validation of all identifiers (electionEventID, verificationCardSetId, BallotBoxId, verificationCardId) in the control components and the setup component.

That would certainly be a good step forward and we will comment on the current status of the context and input verification in Section 3.2. In addition, Post has said:

> The GenKeysCCR algorithm is the first algorithm of the control components to receive the encryption parameters. Hence the control components cannot validate them at this point. However, they will use the same encryption parameters in all subsequent algorithms.

This appears to be a fairly clear problem, albeit one with the documentation rather than the code. If the algorithm is not intended to receive the input from a trusted source or an internal view then it should be listed as input not context, or at least marked in some way as inconsistent with the norm; Algorithm 4.1 still lists this as context.

In summary, we consider there to be room for significant improvement in describing the source and verification checks of context and input in the pseudocode algorithms; however, the more pressing issue is to ensure that the code actually does perform the explicit checks and the more obvious (currently implicit) consistency checks. We will comment on the status of the code issue in Section 3.2 of our update on Scope 2.

2.4 Vote confirmation agreement

In our most recent report we expressed concern about the informal description of the agreement process in the event that the CCRs have different views of which votes have been confirmed. This has been substantially improved in the new release.

The main changes are:

- a new trusted role, the *dispute resolver*, described in Section 6.2.7 of the system specification and referred to in Section 12 of the Computational Proof,
- a much clearer description of the dispute resolution process, in Section 6.2.7 of the system specification,
- a much simpler verification step (8.01), which simply says that the verifier checks that the set of confirmed votes is identical for all CCRs,
- improvements to the proof, to adapt it for these changes.

Although the changes seem reasonable, and we cannot see any way for the system's security goals to fail at this point, the specification and proof still need improvement in order to be convincing.

The specification Although the main idea of the `ConfirmVoteAgreement` algorithm seems correct, the exact follow-on process needs to be more carefully specified. Does the dispute resolver return the output to the CCRs, and do they update their copy of $\mathtt{L}_{\mathtt{confirmedVotes}}$? We assume so, but it doesn't explicitly say. Is the dispute resolver obliged to provide evidence of proper inclusion to the CCRs, in the form of the $\mathtt{hlVCC}_{\mathtt{id},j}$ values? Should the CCRs check them? If a CCR has included a certain vote in $\mathtt{L}_{\mathtt{confirmedVotes}}$ but hears from the dispute resolver that it should not be confirmed, should it remove it, even if it has received the evidence? We acknowledge that some of these cases may be unreachable for a perfectly trustworthy dispute resolver, but we are not certain. The CCRs' algorithm for receiving information from the dispute resolver and updating their state should be specified.

The proof Theorems 2 & 3 (the proofs against vote insertion and deletion by breaking the confirmation process) do not exactly prove the right thing, i.e. that the vote cannot be included if it has not been confirmed, and cannot be rejected if it has been confirmed. They need to be extended to do so, by referring to the dispute resolution process and the unanimity required by the verification spec. The current changes are a good start, but suffer a little from confusion (at least our confusion) about whether the lists of confirmed votes are explicitly updated after the agreement process. For example, it isn't clear whether the bad conditions $\mathtt{bad}_{\mathtt{rej}}$ and $\mathtt{bad}_{\mathtt{inj}}$ refer to the honest control component's list of confirmed votes before or after the dispute resolution.

To take one concrete example, suppose someone comes along later and decides on a flawed dispute resolution process: take a simple majority's view on whether the vote was confirmed, without examining the evidence. Then (we think) Theorems 2 and 3 remain true if we interpret $\texttt{bad}_{\texttt{rej}}$ and $\texttt{bad}_{\texttt{inj}}$ as applying to the honest CCR's state *before* the confirmation process. However, the vote could be accepted or rejected inappropriately (by 3/4 majority) even if the bad cases defined in the Theorems never happen. Conversely, if we interpret the bad events as applying *after* the dispute resolution process, the bad events could certainly happen with the flawed dispute resolution process: the honest CCR could be persuaded to change their mind because the majority said otherwise. This is why the dispute resolution process needs to be a part of the proof, and why it is critically important to clarify whether the bad events refer to the state before or after dispute resolution. (In Theorem 3, the proof clearly refers to the dispute resolution process, but the game in Figure 31 does not model it.)

Currently the vote insertion and vote rejection games do not model the adversary's opportunity to attempt to disrupt the dispute resolution process, but they should. For example, the adversary might send different values to the dispute resolver from those it sent to honest participants during the `ConfirmVote` execution.

One approach would be to split each proof (vote insertion and vote rejection) into two distinct steps.

The first step (for Theorem 2 and 3 respectively) would interpret the bad event ($\texttt{bad}_{\texttt{rej}}$ and $\texttt{bad}_{\texttt{inj}}$ respectively) as applying to the honest CCR's state *before* dispute resolution. The Theorem statements and proofs would be quite similar to their current form (without the references to the Section 12 dispute resolution process). They would not model the dispute resolution process, instead arguing about the security properties of the `ConfirmVote` protocol as they currently do. (It might work better in the case of Theorem 3 to define $\texttt{bad}_{\texttt{inj}}$ as the event in which the voter did not confirm the vote and *any* CCR got enough evidence to prove to the dispute resolver that the vote should be included.)

The second step would be to explain why the dispute resolution process implies that the right consensus prevails. A possible pair of bad events would be:

- for rejection, that the voter successfully confirmed, but the dispute resolver did not cause the vote to be unanimously accepted by all 4 control components (and verification passed),
- for injection, that the voter did not confirm the vote, but the dispute resolver caused the vote to be unanimously accepted by the CCRs (and verification passed).

This argument would involve a game that explicitly allows the adversary to attack the resolution step. It would reference `ConfirmVoteAgreement` to prove that the impossibility of these bad events follows from Theorems 2 and 3.

An alternative proof strategy would be to keep the argument as it is, but update the games in the preambles to Theorems 2 and 3 to include the dispute

resolution protocol.

Typos: at the bottom of Algorithm 6.9, `SendVoteAgreement` in the System specification, we think the Output should be $\top$ if the vote was *sent*, (not confirmed).

2.5 Security reductions and their underlying assumptions

In our previous reports, we pointed out the need to produce more explicit security reductions proving the indistinguishability between security games: we were often trying to guess how these reductions would work and, in several places, we just could not see how to support the security statements that were made.

The latest Proof document shows major improvements here: proofs have been modified in many places, addressing the specific concerns that we raised about their validity, and the reductions that are now proposed for some of the proofs are much more detailed.

We think that these changes are of major importance as it now makes it possible to review proofs that were essentially infeasible to review before, and we believe that pursuing the work in this direction would be vital in order to address the requirement of the Ordinance regarding the existence of a computational security proof.

The currently proposed reductions still raise difficulties: some of them are related to issues that we discussed previously, some of them only appeared now. We review below the proof of Lemma 1. The proof of this lemma has 4 games, and reductions are proposed for 3 of these games. We found difficulties in all 3 reductions. Fortunately, all of these difficulties seem to be reasonably easy to solve.

However, most of the other proofs are much less detailed and follow similar patterns, so we suspect that similar problems will appear. As long as the reductions are not written, it will not be possible to assess whether they will be as easy to solve, or if they will rather lead to the discovery of real protocol vulnerabilities, as it is often the case when one examines the intricacies of security proofs.

2.5.1 Extraction of secrets

In our report dated March 24, 2022,[1] we raised concerns, on page 15, that:

> It is unclear why the extraction of the secret keys from the adversarial CCRs can work as it is described. Extraction is assumed to work for one proof, but this may not be enough to guarantee that it can be performed for many proofs.

The Proof document now offers considerably more discussions regarding this issue, including in Sections 13.2 and in the reduction of Game rMTC.1 on page 72.

[1] https://www.newsd.admin.ch/newsd/message/attachments/71147.pdf

About the proposed extraction strategy However, we do not believe that the extraction strategy that is explained in the latest Proof document works.

Our understanding of the proposed strategy is as follows: there are $\mathtt{N_E}$ proofs to be extracted ($2 \cdot \mathtt{N_E} \cdot (m-1)$ proofs, more precisely, which is even more demanding, but we simplify a bit here), and the witnesses of these proofs are extracted one by one, in the order in which the adversary queried the random oracle in order to make these proofs. The extraction itself proceeds, for each proof, by rewinding the adversary to the point at which the RO query with the commitment of that proof was made, provide a different answer to this RO query, and extract from the corresponding response (given that one can extract the secret from two transcripts that share the same commitment but have different challenges).

We, however, believe that this extraction strategy will fail with overwhelming probability when $\mathtt{N_E}$ grows, rather than with negligible probability, as would be needed for the proof.

Consider the following basic example as an illustration of the problem that can happen. Suppose that, when the adversary computes a ZK proof, it decides to complete that proof only if it received from the RO a challenge ending with 10 bits set to 0 (or, more generally, $\log(\lambda)$ bits set to 0, where λ is the security parameter – of course, the condition based on which the proof will be completed can be much more complex and unknown to the extractor). Concretely, every time the adversary needs to compute a proof, it repeatedly picks random commitments and queries the RO on these commitments until it gets a challenge of the expected form, and completes the proof in that case only.

As a result, a single rewind of the adversary, in which the challenger provides a fresh random answer to the adversary's RO query, will result in the adversary completing the proof on the same commitment with a probability $1/1024$ (or $1/\lambda$ in the more general case).

This is already quite far from the overwhelming probability that would be needed in the proof. However, it gets much worse when we need to extract $\mathtt{N_E}$ proofs. Indeed, since the adversary could very well choose his commitment for each proof as a function of the challenge that it received for the previous one, each single proof requires its own rewinding step (as it is already done in the current proof outline). As a result, the probability of a successful extraction of all the proofs with a single rewind per proof becomes $2^{-10\mathtt{N_E}}$ (or $\lambda^{-\mathtt{N_E}}$), which decreases exponentially fast with $\mathtt{N_E}$ and would simply never happen as soon as we have more than half a dozen voters.

We expect that this issue can be solved by increasing the number of rewinds. Here is a sketch of how this could go (we did not check the details). Suppose that we have an adversary that, on a random challenge, completes a proof with probability p, and assume that p is lower bounded by some polynomial inverse in the security parameter λ. If we rewind a single proof λ/p times, we will be able to extract at least once with a probability $1-(1-p)^{\lambda/p} \approx 1-e^{-\lambda}$ when $p \to 0$ (remembering that $\lim_{n\to\infty}(1+x/n)^n = e^x$). So extraction would fail with probability $\approx e^{-\lambda}$, which is negligible.

The second step is that one needs to successfuly extract $\mathtt{N_E}$ proofs. If we

repeat the same process $\mathtt{N_E}$ times, the probability that we extract every time would again be taken down to $(1-e^{-\lambda})^{\mathtt{N_E}}$. If $\mathtt{N_E}$ were a constant, then we should be asymptotically ok. Further adjustments on the number of rewinds may be needed when $\mathtt{N_E}$ is considered to be polynomial in λ (we did not check).

Moving forward We essentially see two ways of moving forward here:

1. Either an extraction strategy can be fleshed out, possibly inspired from the one outlined above, and then a complete proof of extraction can be made, resulting in revised security bounds in Lemma 1. We note that this strategy does not make any use of the weak simulation extractability (WSE) property of the proof. More generally, it may question the usefulness of the WSE definition in the document, and may suggest the introduction of specific lemmas explaining how to extract from multiple ZK proofs in Section 7.

2. Alternatively, the authors may wish to stick to strategies based on the WSE property discussed in Section 7. However, the reduction should then be based on the WSE definition, which is not the case here (as it directly explains the underlying rewinding strategy). It should also be explained why the security bounds resulting from the use of WSE are negligible.

2.5.2 On IND-CPA security

In various places, the IND-CPA security of ElGamal is used. We are not sure of what definition of IND-CPA security is used: there are a few equivalent ones in the literature, and it sometimes looks like different versions are used within the Proof document.

For instance, the statement of Lemma 1 has a factor $2 \cdot \mathtt{N_E}$ next to the IND-CPA bound, which is what one would expect when the standard IND-CPA security notion is used (i.e., the single challenge one, as found on Wikipedia or in the Katz-Lindell textbook for instance) and there are $2 \cdot \mathtt{N_E}$ messages from which the adversary can distinguish. The multiplicative factor then comes from the so-called hybrid argument that is used to adapt from a single challenge to multiple challenges.

But, in Game $\mathsf{rMTC}.3$, the reduction $\mathcal{B}$ seems to query the IND-CPA oracle with multiple pairs of messages. This is not consistent with a single challenge definition, and suggests that a multi-challenge one is used. But, if a multi-challenge definition is used, then we wonder why the $2 \cdot \mathtt{N_E}$ factor would be needed in the Lemma statement.

We would suggest to state the IND-CPA security definition that is used in the Proof document, possibly in Section 3, and to make sure that it is used everywhere.

2.5.3 What is constant?

Related to the previous observation, we find it surprising to see that, in the reduction made for the rMTC.4 game for instance, no hybrid argument is used to build the reduction. As it is, the reduction looks correct if $\mathtt{N_E}$ is constant. The common practice in the literature seems rather to consider that $\mathtt{N_E}$ may grow with the security parameter: more voters prompt for a larger security parameter. However, if $\mathtt{N_E}$ is not constant, then the argument proposed in the Proof document appears to become incorrect – see for instance the discussion in `https://eprint.iacr.org/2021/088.pdf` in order to see why.

We would suggest adapting the proofs to a setting in which $\mathtt{N_E}$ may be polynomial (or to argue otherwise).

3 Update on Scope 2

We provide an update below on various issues within scope 2. In summary, the implementation has matured past us readily finding exploitable vulnerabilities but we have not had time to systematically review the security of the implementation.

We have carefully looked for many common issues in the code and spent significant time carefully following the logic of certain security requirements. Many important improvements have been made in response to our feedback. However, the implementation of the system involves 10 micro-services taking responsibility for various parts of the protocol across the Setup Component, Online Control Components, and Tally Control Component; in addition, there are 51 verification tasks within the verifier each containing a multitude of checks. By our count, there are dozens of stateful services between the various components—which manage databases and file repositories. In the context of this complexity, we have assumed that much of the underlying services are appropriately implemented without systematic checking.

3.1 Undocumented architecture decisions

In our previous report we commented that

> There are numerous architecture decisions which have been made which have various advantages and disadvantages. This is to be expected but we think many of (the) decisions should be better described in the documents to ensure a clear view, particularly of the disadvantages. In our previous report we highlighted micro-services (which describes both the online system and the verifier)...this has not been addressed.

Post has now undertaken to address this by updating the architecture document; we understand that Section 9.1.2 and 9.2.3 are the main updates to reflect this.

E-voting systems have traditionally been developed as fairly monolithic applications; reviewers and auditors have got used to the kinds of vulnerabilities

which occur in such systems. There are many advantages to using micro-services over monolithic applications, but they may introduce new errors which were uncommon or unknown in the monolithic systems. Since reviewers most readily find errors of kinds they are most familiar with, we consider this change in architecture style to be a significant point of risk (though not an unwarranted one).

Post has done a good job of highlighting the advantages of the approach in the architecture document; however, the drawbacks in terms of security are only partially considered. For example, the attack on UV described in our previous report (about which we provide an update on Section 3.3) would have been very unlikely to occur if the data was read in once and then passed in memory through to the various verifier checks. There is some discussion of the security mechanisms required to address this sort of issue (for example, exactly once processing) but this discussion is ad-hoc and does not seem to systematically consider the possible security implications.

We consider the main risks of the use of micro-services to be:

- ensuring appropriate state between the services,
- ensuring consistency between the input to the services,
- and the lack of familiarity of the reviewer with the common pitfalls and countermeasures in this approach.

A semantic discussion of how state is managed and input consistency is assured would help to mitigate these risks.

In summary, while improvements have been made to the architecture document they are largely orthogonal to our main concern; specifically, the analysis of the security implications is not sufficiently systematic and detailed to convincingly show that the risks arising from this decision have been mitigated.

3.2 Unvalidated inputs

In our previous report we commented in some detail on the input validation of various of the algorithms; we provide an update on the these issues below. We focus our discussion on the three main points:

- Does the context "stem from a trusted source (for instance an internal view or a trusted component"?
- Does the implementation "check the input variables against the context"?
- Does the implementation "ensure the validity of the identifiers and indices"?

Alg. 4.1 GenKeysCCR and Alg. 4.9 SetupTallyCCM The encryption parameters and election event id are listed as context in Alg. 4.1 but do not come from a trusted source; this is a discrepancy between the specification and implementation. In our view they should either come from a

trusted source in the implementation or be listed as input in the specification. Moreover, it is unclear to us why the encryption parameters are input at all, since they are supposed to be (according to the specification) deterministically derived from the name of the election event.

Alg. 4.4 GenEncLongCodeShares All contexts to Alg. 4.4 come from trusted or internal views, though as we noted last time:

> the encryption parameters were placed in local state after coming from an untrusted party and hence should not really be considered trusted.

In this case the issue is mitigated because the encryption parameters are cross checked with the input from the Setup Component.

The implementation takes as input the verification card public keys but this is not represented in Alg 4.4; this does not seem to matter as far as we can tell.

Alg. 4.5 CombineEncLongCodeShares and Alg. 4.6 GenCMTable In our previous report we commented that:

> the verification card ids should be...checked as consistent within the verification card set id.

This is now done through a method called verifyConsistencyChunk.

Alg. 4.7 GenVerCardSetKeys We had previously commented that:

> It is not clear to us that GenVerCardSetKeys actually ensures that $pk_{CCR_2}, pk_{CCR_3}, pk_{CCR_4}$ are actually ϕ long and not longer; the underlying issue is that GroupVector reports element size as the size of its first element without checking that all its elements have the same size.

Post has correctly pointed out that, while this is true of the private constructor, all public methods to create GroupVectors do check that the elements all have the same size.

Alg. 4.10 SetupTallyEB We had previously noted that:

> The spec lists the maximum number of write-ins as part of the input where similar parameters were considered context in other algorithms, this seems to be an a minor error in the specification.

With the exception of the above, the other issues we raised seemed to have been resolved.

Alg. 5.3 VerifyBallotCCR and Alg. 5.4 PartialDecryptPCC The main change Post has made to the PartialDecryptService, which prepares the input for these two algorithms, has been to add an identifier validator.

This validator at present is checking that all the ids (election, verification card set, verification card) exist in the CCR's local view. This is a good start but it does not appear to address our major concern of inconsistency between these values. We provide below Java code for the checks we believe need to be added to the validation.

```
checkArgument(verificationCardSetService.getVerificationCardSet(verificationCardSetId).
        getElectionEventEntity().getElectionEventId().equals(electionEventId),
        "Verification card set and election event not consistent.");

checkArgument(verificationCardService.getVerificationCard(verificationCardId).
        verificationCardSetId().matches(verificationCardSetId),
        "Verification card and verification card set not consistent.");
```

The code above checks verification card set and election event match in the CCR's view, and the same for the verification card and set.

Alg. 5.5 DecryptPCC and Alg. 5.6 CreateLCCShare Adding the checks suggested in the previous item to the identifier validator would also resolve the issue we had with the input validation of these algorithms.

Alg. 5.9 CreateLVCCShare and Alg. 5.10 VerifyLVCCHash The identity validator is again used here and would be improved by our suggested changes. Interestingly in this case the (input validator) checks are preformed as part of the payload verification in the onMessage method of the LongVoteCastReturnCodesShareHashProcessor; this is not entirely consistent with the position of these checks in the cases above; we cannot see a strong reason to prefer either convention.

Alg. 6.1 GetMixnetInitialCiphertexts, Alg. 6.2 VerifyMixDecOnline, and Alg. 6.3 MixDecOnline These three algorithms are used by the MixDecryptProcessor to handle requests it receives; this processor does some validation but largely depends on the MixDecryptService to handle the processing of requests. One of the changes Post has made is to explicit check that the election event and ballot box correspond in the CCR's view, this is an excellent change.

Alg. 6.4 VerifyVotingClientProofs, Alg. 6.5 VerifyMixDecOffline, Alg. 6.6 MixDecOffline, and Alg. 6.7 ProcessPlaintexts

We would like to see an explicit comment about line 123 of MixOfflineFacade to note that the code on the lines ensures that the ballot box and election event match according to the electionEventContextPayload; at present we are concerned that this check is implicit and may be accidentally removed in the future.

In general it seems that the checks performed for these algorithms should be adequate but we had some trouble following the source of some of the data; for example, the data coming from the BallotTallyService.

In summary, with a few exceptions Post has fixed the issues with input validation we were concerned about. It should, however, be noted that the

current approach is heavily reliant on the setup component as a ground source of truth; if in the future the setup component is less trusted, then the issue of input validation will again become a pressing concern.

3.3 Update on attack on UV

In our previous report we detailed an attack vector on universal verifiability which centered around reordering and renaming shuffle payloads to cause the verifiability chain to break. As we understand it, Post addressed this problem by adding a check called VerifyFileNameNodeIdsConsistency; this, in combination with the checks present in verification, prevent every variation of this attack we can think of.

We consider this issue to be resolved.

3.4 Update on possible attack on IV

In our previous report we detailed a possible attack vector on individual verifiability. The issue was a lack of checks by the setup component that the verification card ids received from the CCRs match those it sent out both in order and content; this coupled with weakness in the verification which didn't detect this attack.

We suggested changes to VerifyVerificationCardIdsConsistency which Post has now made; these changes prevent every variant of the attack we can think of.

We consider this issue to be resolved.

4 New issues

We comment below on a few new issues we have detected since our last report.

4.1 Problem with KDF

Several of the algorithms, in cryptographic primitives specification, including 4.8, 4.9, 4.10, 4.12, and 4.13, take input strings which are supposed to be in the Universal Coded Character Set according to ISO/IEC10646. However, unless the input is restricted to UTF_8 it seems that the conversion from strings to bytes performed by ConversionsInternal.stringToByteArray will result in collisions since the "method always replaces malformed-input and unmappable-character sequences with this charset's default replacement byte array." To our knowledge this does not result in any exploitable vulnerabilities in the system but we suggest being more careful about the input domain validation checks and type conversions.

4.2 Error logging

In several cases the system has code like

```
if (decryptVerif.stream().allMatch(VerificationResult::isVerified) &&
  shuffleVerif.stream().allMatch(VerificationResult::isVerified)) {
    return true;
} else {
  decryptVerif.forEach(verificationResult ->
    LOGGER.error(verificationResult.getErrorMessages().getFirst()));
  return false;
}
```

This results in an UnsupportedOperationException if a subset of the checks failed because the system tries to get error messages from passing verifications. We suggest updating the code along the lines below so that only failing checks have their error messages logged.

```
if (decryptVerif.stream().allMatch(VerificationResult::isVerified) &&
  shuffleVerif.stream().allMatch(VerificationResult::isVerified)) {
    return true;
} else {
  decryptVerif.stream().filter(a -> !a.isVerified()).forEach(
    verificationResult ->
      LOGGER.error(verificationResult.getErrorMessages().getFirst()));
  return false;
}
```