Vestigial Vulnerabilities in Deployed Verifiable E-Voting Systems

Thomas Haines
School of Computing
Australian National University
Canberra, Australia
Email: thomas.haines@anu.edu.au

Jarrod Rose
School of Computing
Australian National University
Canberra, Australia

***Abstract*—Electronic voting systems claiming to provide verifiability are seeing increased adoption. Previous work on analyzing these systems has focused on vulnerabilities arising in the specification and implementation of the core protocol and primitives; once the system has been analyzed for these vulnerabilities and appropriate fixes deployed, one might have hoped that the systems would provide the claimed security.**

In this paper, we discuss two categories of vulnerabilities which still seem prevalent in otherwise carefully designed, implemented, and audited systems. We present ten examples of vulnerabilities or weaknesses in these categories drawn from the SwissPost and Belenios systems. Our discussion covers why vulnerabilities in these categories maybe escaping detection and what can be done about it; all the solutions we considered are unsatisfactory and our aim is to highlight this area as an important open problem.

1. Introduction

Although verifiable electronic voting has been an area of research for over four decades, efforts at implementing these techniques have only accelerated in the last two. The implementations in the late 2000s and 2010s can be broadly split into relatively transparent efforts and less transparent efforts. More transparent systems include Helios [1], Scantegrity [2], vVote [3], CHVote [4], and STARVote [5]. Whereas the systems provided by commercial vendors supporting the bulk of deployed systems tended to be very closed to scrutiny; non-disclosure periods of five years were common if access could be secured at all.

Increasing transparency, at least in part as a result of government regulation, in the commercial vendor products in late 2010s saw a flurry of (often long concealed) vulnerabilities discovered and (in some cases) rectified [6], [7], [8]. In this paper we do not aim to discuss the limitations in the state-of-the-art of academic protocols but rather vulnerabilities in deployed systems which invalidate the claimed security, at least until the system is patched.

Only in the last few years are we seeing systems like ElectionGuard [9], SwissPost Voting System [10], and Belenios [11] which have seen the level of sustained effort and investigation to achieve a measure of maturity. This leaves open the question of what kinds of verifiable voting specific vulnerabilities we might find in these systems once the more common vulnerabilities have been addressed.

1.1. Voter Verifiable Electronic Voting

Voter verifiable electronic voting systems aim to tolerate a high number of malicious insiders. The two key properties are verifiability and privacy; informally, verifiability says that if the election result is not consistent with the votes of the honest voters this will be detected. This is often expected to hold even if a single auditor is honest. Privacy is often expected to hold if, in addition to a single auditor, a single of the tally authorities is honest. Deployed systems often make stronger assumptions on the trustworthiness of the authorities, but still at least some of authorities are normally assumed to be corrupted. While the threat model is strong, the notation of verifiability is weak in that it only says attacks must be detected not prevented or attributed; most of the weaknesses we present, while not attacks on verifiability, are not prevented or recovered from by the protocol.

Formalizing these notations of security has been difficult; there are nearly as many definitions of security as there are systems. Prominent examples include Cortier et al. [12] on verifiability and Bernhard et al. [13] on privacy. Getting definitions which capture both the kinds of adversaries we wish to consider and admit security proofs which require a reasonable amount of effort to construct and verify is an ongoing challenge. Both of the two deployed systems we present vulnerabilities on have proofs, both paper proofs and formalized proofs of security in model checkers (such as Tamarin and ProVerif) [14], [15], [16] and/or interactive theorem provers (such as EasyCrypt) [17]. These proofs missed the vulnerabilities, in neither case does this seem inherent to the tool; we discuss further in Section 7.

There seems to be a very strong correlation between the quality of the governance process by which an electronic voting system is designed, implemented, and audited and the number of vulnerabilities. This is a key takeaway from various trials and experiments in the area and has been summarized in numerous papers [4], [18], [19]; this correlation between the quality of the governance of the process by which the system is designed, implemented, operated, and

Weakness/Vulnerability	Impact and Severity	Category	Ref
V1. API free for all	Extra voter credentials created	**3.2**	(Sec. 4.4.1)
V2. Shuffling the Shuffles	*Universal verifiability breach*	**2**	(Sec. 4.4.2)
V3. Spoofing Elections	*Universal verifiability breach*	**3.1**	(Sec. 4.4.3)
V4. Triggering an Early Tally	*Privacy breach (later detected)*	**3.2**	(Sec. 4.4.4)
V5. Messing with Zip Files	Invalidates security claims on auditing	**2**	(Sec. 4.4.5)
V6. Extra Voters	Extra voter credentials created	**3.3**	(Sec. 4.4.6)
V7. Permuting IDs and Keys	DoS (and possible more)	**2, 3.3**	(Sec. 4.4.7)
V8. No Filtering	*Universal verifiability breach*	**1.1**	(Sec. 4.4.8)
V9. Audit to Late	*Privacy breach (later detected)*	**1.1, 3**	(Sec. 5.3.1)
V10. Ghost Voters	Voters issued invalid credentials	**3.3**	(Sec. 5.3.2)

TABLE 1. SUMMARY OF ANALYSIS RESULTS (FOR DESCRIPTION OF CATEGORIES SEE FIGURE 1)

audited and security seems to be stronger than with any particular technical tool or technique. A few key points about this governance, which are now enshrined in Swiss legislation [20], are:

1) the system needs to have clear security claims;
2) the way in which the design, implementation, and operation meets these claims need to be clear;
3) sufficient details need to be (publicly) available to audit the design, implementation, and operation;
4) there needs to be incentives to audit the system.

Currently deployed systems roughly satisfying at least the first two or three of these points include:

- SwissPost Voting System [10]
- Belenios [11]
- Systems based on ElectionGuard [9][1]

Studying vulnerabilities in systems that do not match the points above is important when the systems are used for important elections. However, it is also like studying why a bridge fell over when there was no design process; the fact that it was not fit for purpose is hardly surprising and almost always the discovered reasons, or to return to our specific case vulnerabilities, are well-known in the literature; this does not help us to design good future systems. In contrast, the study of vulnerabilities in systems which are more mature can provide valuable feedback on the limitations of the current design process. This distinction has motivated our choice of systems and vulnerabilities to analyze.

1.2. Current Layers of Abstraction

The current norm for verifiable electronic voting systems is to represent the core protocol and primitives in a specification. This specification sometimes has a reference implementation in the form of a Software Development Kit (SDK), as occurs in ElectionGuard [9]. In other cases, implementation of the core protocol and primitives is left to the system implementers. The core protocol and primitives alone do not constitute a deployable system. As well as additional e-voting specific code, deployed systems also include many security dependencies on supporting libraries, operating systems, and hardware but these are orthogonal to the issues we discuss in this paper.

The concern we raise in this paper is that improving security in the core protocol specification and implementation seems to be leaving a surprising number of exploitable vulnerabilities in the deployed systems. The vulnerabilities we present to this point are taken from the SwissPost Voting system [10] and Belenios [11]; we speculate that similar vulnerabilities exist more broadly.

In Oded Goldreich's book on the complexity theory [21] he attributes a quote to Turing award and Abel prize winner Avi Wigderson which reads "We are successful because we use the right level of abstraction." The conjecture of this paper could be summarized as "We are unsuccessful (see discovered attacks) because we use the wrong level of abstraction." In unpacking this point Goldreich suggests that for a level of abstraction to be good it must be workable at the theoretical level and adequately capture the intuitive concept; in our case this is the concept of security in electronic voting. The difficult in proposing changes to abstraction layers used in e-voting system specification and formal analysis is maintaining the former property while enhancing the latter. We will elaborate on this point in Section 7; we do not have any solution which we think is good, rather our aim is to convince the reader that this is an important open problem.

1.3. Contribution

In this paper we present eight weaknesses on the Swiss Post Voting System [10] and two on Belenios [11], listed along with impact in Table 1; these weaknesses have not previously appeared in an academic publication. Five of these were exploitable vulnerabilities that violated the claimed universal verifiability or privacy; the remaining five would

1. There is a strong caveat here that we are really only claiming the first two of the properties for the ElectionGuard SDK part of the system; we make no claim as to how these systems fair on the last two properties.

have been detected but could not be recovered from within the protocol. We classify these into two categories:

- Flaws in handling input and output
- Flaws in handling state

To our knowledge, there is no existing taxonomy of vulnerabilities which has been created for, or applied to, verifiable electronic voting systems. Our categorization, see Section 3, could serve as useful basis for such a future endeavor. Because we focus on vulnerabilities outside of the core protocol and specification, we do not discuss in any depth vulnerabilities which are inside. A good taxonomy would want to be much more fine-grained on these cases.

Taking these weaknesses as case studies we construct hypotheses as to why they might be occurring and possible solutions. Our discussion of mitigations captures both easy adjustments to processes which would mitigate the most common current examples and a more speculative discussion of enhanced methodologies which might catch attack categories of which we are currently unaware. We are fairly certain in our easy adjustments and would strongly advocate their adoption; in contrast, our discussion of enhanced methodologies is speculative and at present we do not see any solution which seems wholly satisfactory.

1.4. Limitations

Our vulnerabilities are drawn from an extensive review of the SwissPost system and less detailed review of Belenios, see Section 3.2. However, this is still a small sample of verifiable voting systems; we mitigate this by our wider categorization in Section 6. At present, there are relatively few deployed verifiable e-voting systems which are mature, this limits the number of examples in scope.

Our discussion of what can be done is narrow in scope. We are interested in why exploitable security vulnerabilities in the e-voting specific code[2] of deployed systems are occurring, and not being caught by the security analysis of the specification. This focus is motivated by the kinds of vulnerabilities we found in our broader analysis. However, in suggesting the need to refocus the way in which these systems are specified and analyzed, we do not mean to detract from all the other areas which contribute to the security of the deployed systems.

1.5. Outline

In the next section, we will cover various of the techniques used in the SwissPost and Belenios e-voting system. We present a rough taxonomy of vulnerabilities in Section 3 based on prior work and our investigations; we also discuss the methodology of our investigations. In Sections 4 and 5, we present the ten weaknesses with an intuition of the issue but without providing full technical details. However, we provide links to the disclosure reports that provide further technical details beyond what we have space for here. We also discuss how other recently discovered vulnerabilities would fit into these categorization in Section 6. In Section 7, we discuss possible reasons why vulnerabilities in these categories are more common and what steps can be taken to manage them. Finally, in our conclusion (Sec. 8) we summarize the takeaway points of our paper to various stakeholders. In appendix A, we provide further details on the weaknesses and vulnerabilities.

2. By this we mean the code of system itself rather than inclusive of dependencies.

2. Background

In this section, we will cover various core techniques for electronic voting systems. Verifiable electronic voting systems can be broadly separated into in-person voting systems and remote voting systems; most examples in the second category are online voting systems but this is not inherent. Many of the core techniques are used in both the in-person and online systems. While both the systems we present vulnerabilities on are online systems, this seems largely orthogonal to vulnerabilities.

2.1. Bulletin Board

It is common when designing an e-voting system to assume the existence of a (web) bulletin board. This board is essentially an append-only broadcast channel with memory; this is used to ensure a consistent view of the main election information including parameters, cryptographic key material, encrypted votes, and tally output. The SwissPost system is notable for not using a bulletin board which in some ways simplifies the system but also contributed to the state vulnerabilities we present. The Belenios system has the voting server act as the bulletin board and relies upon auditors to regularly check it to ensure it behaves honestly.

2.2. Tallying Methods

In most (online) electronic voting systems the voter, or rather their device, will submit a ciphertext to the voting server whose plaintext is the voter's ballot. There are two principal techniques for anonymizing these (encrypted) ballots before decryption: mixnets and homomorphic tallying. The Belenios system uses both of these techniques depending on which best fits, whereas the SwissPost system exclusively uses mixnets.

2.2.1. Homomorphic Tallying. Homomorphic tallying makes use of a (threshold) homomorphic encryption scheme to (at least partially) evaluate the tally function on the ciphertexts before decryption. For example, consider a referendum where each vote is either no (denoted by zero) or yes (denoted by one) and where the ciphertexts are encrypted using an additively homomorphic version of ElGamal [22]; anyone can compute the product of the ciphertexts to produce a ciphertext encrypting a sum of the yes votes; this product ciphertext can then be jointly decrypted by the tally

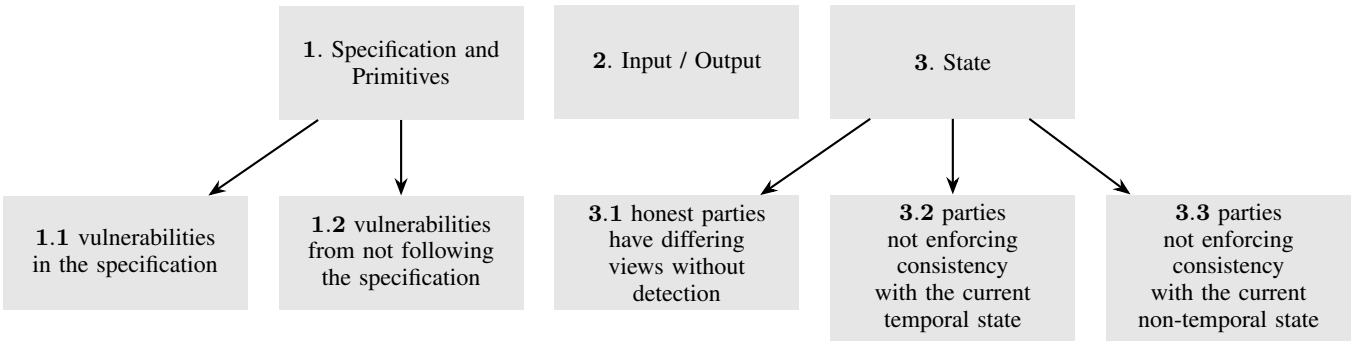

Figure 1. Rough Categorization of Vulnerabilities in Verifiable E-Voting Systems

authorities, revealing only the number of yes votes. This approach was pioneered by Benaloh, Fischer and Yung [23], [24] and first appears in essentially the modern form in the work of Cramer, Gennaro, and Schoenmakers [25].

2.2.2. Mixnets. A mixnet is a kind of anonymous channel introduced by Chaum in "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms" [26]. The application of mixnets to electronic voting is immediate; we can use them to hide the correspondence between voters and votes. A synchronous mixnet essentially takes in a set of ciphertexts and outputs the corresponding plaintexts in a random order; privacy can be informally summarized, as the adversary cannot learn anything about the association between ciphertexts and plaintexts unless all of the mixers are corrupt. This process is normally made verifiable using a kind of zero-knowledge proof called a proof of shuffle.

2.3. Cast-as-Intended Methods

One aim of many works in the electronic voting literature is to reduce trust on the voter's device; most of these works are focused on providing the voter (human) a mechanism to tell if their vote was cast as they intended even when their device is corrupted. There are several major families of related techniques but the one used by the SwissPost system is return-code voting.

2.3.1. Return-Code Voting. Code voting first appears in the SureVote system by Chaum [27]. In a code voting system, the voter receives a code sheet in the mail, or other channel, which is used as a private code book between the authority and the individual voter. What distinguishes a return-code voting system from a code voting system is that the user does not enter the codes; rather, the codes are displayed after the voter has selected and submitted their choices. This provides weaker privacy than code voting (where the voter enters the code into the device) but better useability.

3. Categorization and Methodology

In this section, we will unpack both the categories we will be using to classify vulnerabilities and the methodology used in our analysis of the deployed systems.

3.1. Categorization

A rough categorization of vulnerabilities in verifiable E-voting systems is given in Figure 1; it has three top level categories, which will discuss below. We are not claiming that any of these categories are novel; the categories appear implicitly in many prior works. For example, Cortier et al. [28] discussion of managing complex state and Hirschi et al. [29] on consensus. However, drawing them into a coherent whole and analyzing which categories are most frequent in practice seems valuable.

1. Specification and Primitives The first category combines all vulnerabilities visible in the specification and description of core primitives into category **1.1**, and vulnerabilities from not following the specification in **1.2**; these collectively account for almost all known vulnerabilities. Many deployed verifiable E-voting systems do not have a specification which details the protocol in any appropriate detail, in such cases we still include their vulnerabilities in this category if it would have been visible in a hypothetical specification written at the level of detail expected of a mature system.

2. Input / Output These vulnerabilities occur when the code handling the input and output of data to the core protocol deviates from the expected behavior under adversarial input. This can be thought as the adversary causing the honest parties to deviate from the expected protocol. These vulnerabilities are in some sense trivial and we suspect the number of them produced by professional software engineers is counter-intuitive to many of the designers of the core protocols. As we elaborate on in our discussion, we suspect the software engineers are unused to building systems for this kind of threat model and tacitly view the input as honest.

3. State These vulnerabilities occur when the core protocol is called either at times, or on input, which could never occur in the security model which is envisioned in the specification. As we elaborate on in our discussion, many of the obligations on the implementers are implicit in the security definition. Having crucial security constraints expressed implicitly is unideal, and is a plausible candidate explaining why these vulnerabilities are appearing in deployed systems.

The two categories *input/output* and *state* are the ones we are primarily interested in this work. As we shall, some vulnerabilities sit at the intersection of these categories; in such cases, we view them as simultaneously belonging to all relevant categories.

3.2. Methodology used to Discover Vulnerabilities

The methodology, in common to both analyses, consisted mostly of manual review of the specification and code. We did make use of dynamic code testing when we needed to confirm the presence of a vulnerability, either by creating new unit tests where applicable or malicious payloads to input to the system. We include the steps needed to recreate the attacks in Appendix A. The process we used to discover bugs was in some regards simple, having observed that the honest parties were triggered to act by API calls we examined the following questions:

- What API calls are available?
- Who is authorized to make the calls?
- When can the calls be made?
- How is the input to the calls validated?

Analyzing the first three of these is fairly straightforward, it is normally clear what each call does and hence who should be allowed to make it and when. API calls were easily identifiable either by annotations, in the SwissPost case, or because they are concentrated into API files in Belenios. Restriction on calls, in the form of signature verification or timing constraint checks, normally occur immediately which makes checking these easy.

The last point is more complicated, it requires knowledge of what validations are required. We largely used tacit knowledge in evaluating this; however, the essence was to check that the validation constraints did not allow the adversary to meaningfully deviate from the honest protocol. This is not a principled approach, to our knowledge, no satisfactory principled approach is known; our discussion in Section 7 is intended as first step toward developing a principled approach for future investigations.

Our analysis of the SwissPost system is significantly deeper than that of Belenios. We have spent significant time analyzing the specification, security proofs, code, and details of operation against the many requirements specified in the Ordinance on Electronic Voting [20]; the ordinance regulates the authorization of electronic voting for government elections in Switzerland.

4. The SwissPost System

The SwissPost voting system is the successor to the sVote system which was withdrawn from use in 2019 after numerous vulnerabilities were uncovered [6], [7]. A summary of the revised system and some early errors can be found in the paper "Running the Race: A Swiss Voting Story" [31]. The system is used for elections, alongside other voting channels, in the Swiss cantons of Basel-Stadt, St. Gallen, Graubünden, and Thurgau.

Most of the vulnerabilities we present in this paper are on the SwissPost system, but this does not mean this system is less secure than others; rather, the governance process around the SwissPost system is much better than any other system we are aware of and so the level of auditing has been higher. The vulnerabilities are also increasingly being caught in the (mandated) auditing immediately following release of the software version in which they are introduced; crucially, this auditing occurs before the system is used.

The Post system differs from the standard academic imagination of a system by its use of micro-services. These services are independent and communicate over well-defined APIs. This has many advantages, including:

- Independently deployable which aids scalability
- Small in size and highly modular
- Highly maintainable and testable

However, the separate services as designed to have as little state as possible which is very different from how e-voting systems are normally conceived of and modeled in the academic literature. In the SwissPost system, the average protocol participant in the specification is implemented as roughly ten micro-services in the implementation. Moreover, many of these micro-severices will process their input in batches rather than all at once.

The other large deviation from the standard representation of e-voting protocols in academic papers is the presence of multiple digital ballot-boxes. An election event within the context of the SwissPost system is a particular election at the level of the canton; a federal election will, in the context of the voting system, consist of multiple elections; one election in each canton using the SwissPost system. However, different voters within each of the system's elections may have different voting rights; that is, they are authorized to vote on different questions, for example on more local issues.

These different voters are grouped into sets based on what questions they are allowed to vote on, each of these sets has their own digital ballot-box. This complexity does not occur in Belenios but does occur in both the French Legislative Election E-Voting Protocol (FELP) [32] and CHVote [33] systems, which we discuss in Section 6. To summarize, the academic security definitions largely consider a single election while the deployed system often support multiple concurrent elections.

The use of micro-services, the batching of operations, and the presence of multiple digital ballot-boxes are not modeled in the security analysis. As we shall see, this significant increase in complexity in the deployed system compared to what was formally analyzed means that there is significant space for exploitable vulnerabilities to fall outside of the security analysis.

Version 2.0 of the Swiss Post system is currently in development, and a specification has been produced [34]. Once this new version is implemented it will be interesting to see what kinds of vulnerabilities are discovered in it.

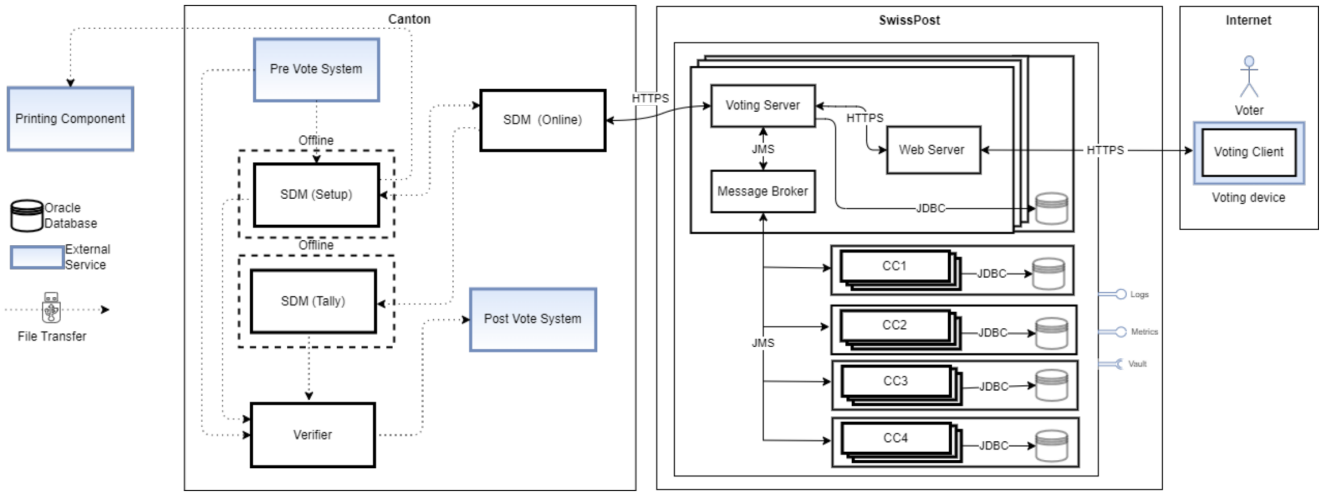

Figure 2. Context Overview of Swiss Post Voting System from [30]

4.1. Scope of Analysis

Our analysis covered most of the specification and implementation of the SwissPost system. Our discussion in this report is based on examinations covering versions `0.8` to `1.4` of the system. We include dates on the weaknesses where possible but in some cases the weaknesses date back to the previous sVote system, which was used as a basis for the SwissPost. Our analysis covered the following items:

- The System, Verifier, and Crypto-primitives specifications [10].
- The E-Voting Source code [35] and supporting Libraries [36].
- Crypto-primitives Source Code [37]
- Verifier Source Code [38]
- Computational proofs [10]
- The typescript library implementing the main functions for the voting client

However, we did not analysis in any significant detail the following items:

- Symbolic proofs [14]
- The software for printing the voting cards
- The wider voting client source code (beyond the core functions)

4.2. Protocol Overview

The principal participants in the protocol are the `voters`, their `devices`, the `setup component` (labeled as `SDM Setup` in Fig. 2), four `online control components` (`CC1`, `CC2`, `CC3`, `CC4`), offline `tally control component` (`SDM Tally`), and the `verifier`. Figure 2, reproduced from the architecture document of the system [30] shows the principal participants along with the various untrusted components. We present a simplified description of the protocol below.

Candidate	Return Code
Spock	1395
Kirk	5340
Bones	6417
Uhura	5876
Confirmation Code	4199
Finalisation Code	3432

Figure 3. A simplified voting card

Setup Phase The `pre vote system` configures the `setup component` and `verifier` with the parameters of the election including:

- the number of voters,
- the questions which will be asked and the allowed responses.

The `setup component` and the `online control components` jointly generate a voting card (see Fig. 3) for each `voter` which is then printed and sent to the `voter` by mail; each voting card contains a unique set of codes. This is done such that all `online control components` need to cooperate with the `voting device` to recover the return codes on the voting card.

Voting Phase In the voting phase, `voters` can instruct their `devices` to construct a ballot which is submitted to the `voting server`. The `voting server` checks the ballot is valid and then the `online control components` jointly compute the return codes. The `voting server` sends the return codes to the `device` which displays them to the `voter`; the `voter` checks that they match the codes on the voting card for the options they selected. If so, the `voter` submits the confirmation code which is required for their ballot to proceed to counting.

	Setup Component	Online Control Components	Tally Control Component	Verifier	Voter's Device
Individual Verifiability	✓	1 of 4	✗	✗	✗
Universal Verifiability	✗	1 of 4	✗	✓	✗
Voting Secrecy	✓	1 of 5	1 of 5	✗	✓
Effective Authentication	✓	1 of 4	✗	✗	✗

TABLE 2. SWISS SECURITY REQUIREMENTS

Tally Phase Once voting is over, the `online control components` shuffle and decrypt the confirmed ballots, as does the `tally control component`. The `verifier` then checks the data provided to it by the other participants and publishes the election result to the `post vote system`.

4.3. Security Requirements

The security requirement for the SwissPost system, and any other system used for electing Swiss governments or deciding referendums, are defined in Federal Chancellery Ordinance on Electronic Voting [20]. The Ordinance contains a long list of requirements but the key requirements are "complete verifiability" and "voting secrecy". The latter is essentially the standard notion of privacy, the adversary doesn't learn anything more than the result, with a few caveats. One area in which this definition is stronger than some others is that privacy attacks must be prevented; it is not sufficient to just detect them.

"Complete verifiability" is broken down into individual verifiability and universal verifiability, the former has various sub-requirements but mainly captures that if the voter (human) believes that their vote was correctly cast, recorded, and confirmed then it was. The later captures that the announced result should be the correct tally of the confirmed votes. In addition, the protocol must provide "effective authentication" which says that ballots can only be confirmed if the adversary has control of the corresponding voters. More formally, the property requires that it should not be possible to cast a valid ballot on behalf of a voter without knowing their (secret) credentials; while the voter's device is listed as untrustworthy for this property, the explanatory report to the legislation [39] emphasizes that an attack where the voter's device leaks the private credentials is out of scope of the threat model.

We show in Table 2 a slightly simplified version of the threat model. A ✓ denotes that the participant is trusted for the property whereas an ✗ denotes it is untrusted; 1 of 4 and 1 of 5 mean that one of the control component is trusted either among all the `online control components`, of which there are four, or including the offline `tally control component`.

4.4. Vulnerabilities

We will now proceed to sketch eight weaknesses in the SwissPost system, four of which if exploited would have violated the main security properties; exploitation of any of the weaknesses would be possible to detect (though in most cases only once the attack vector is known) and there is no evidence of exploitation on the actual system. However, the protocol does not include recovery from any of the weaknesses; a forensic audit of undefined format would have been launched had these weaknesses been exploited.

4.4.1. V1: API free for all. The protocol implemented by the system, like most in e-voting, consists of `setup`, `voting`, and `tally` phases. Prior to July 2023, the micro-services implementing the functionality of the authorities (mainly the `online control components`) in the `setup phase` did not have restrictions preventing them being called during the later or earlier phases; so, `setup` algorithms could be triggered during voting or even during the process of tallying. This behavior is clearly not envisioned by the specification; figuring out if it is possible to exploit this is non-trivial due to various other checks performed by the system. With the exception of the specific case covered in V4 below, we did not find a way to exploit this to break the main security properties. However, it does allow the adversary to tamper with the `online control components` view of the eligible `voters` and what constitutes a well-formed ballot. This weakness is an example of a lack of state enforcement, specifically around the current phase of the election (Category **3.2**). Details of this vulnerability can be found in Section 5.2.3 of the examination report by Haines, Pereira, and Teague - July 2023 [40] and in Appendix A.1.

4.4.2. V2: Shuffling the Proofs of Shuffle. The system uses a mixnet [41] operated by the `online control components` and offline `tally control component`, to anonymize the encrypted ballots. This mixnet is secured by a zero-knowledge proof of shuffle by Bayer and Groth [42]. No security vulnerabilities have been found in the implementation of the proof of shuffle in the SwissPost system, though the commitment parameters were produced incorrectly in the older sVote system [6], [7]. Not withstanding the security of the cryptographic primitive implementation, the proofs of shuffle did not provide the verifiability desired prior to 2022.

The vulnerability occurred in the code which loaded verification data from disk. The code used to load the data for the `online control components` did this based on the *contents* of the verification data; however, the code loading the data for the `offline tally component` does this based on the *filename*. By introducing an inconsistency between *filename* and *contents* the adversary can cause these two functions to return inconsistent results.

Making this exploitable in the system requires some extra work; the main complexity is that the SwissPost system does a partial decryption between each shuffle and naive exploitation of the vulnerability above would result in decrypting the ciphertexts under an incorrect secret key and hence a gibberish result; this issue can be avoided by carefully choosing the public/secret key pairs of the malicious authorities.

The high level intuition is that the proofs of shuffle should ensure a chain of custody where each party proves they did not tamper with the ballots in their care; the inconsistent data handling in the verifier breaks this chain. This is an example of an input/output vulnerability because the verifier has all the data they need to perform the check correctly, they simply parse the data incorrectly. This vulnerability could be exploited to break the verifiability of elections, in the worst case allowing a complete reversal of the correct election outcome without detection. Details of this vulnerability can be found in the examination report of Haines, Pereira, and Teague - 2022 [43] and expanded upon in addendum to that report [44]. Additional details can also be found in Appendix A.2.

4.4.3. V3: Spoofing Elections. The verifiability of the elections is achieved by the `verifier` checking the result is correct based on verification data it receives from various participants. This verification data is signed which prevents tampering, or insertion of new messages, but this does not prevent messages being dropped.

Both the `verifier` and the `online control components` receive a file detailing the election configuration from the `setup component`; the `verifier` has the means to check the correctness of this information but the `online control components` does not. If the `setup component` told the online control components there were more ballot boxes than it told the `verifier`, the adversary could drop the verification data relating to these ballot boxes, and hence the votes would be dropped and not counted. This was patched by having the control components and `verifier` check they have the same view of the election configuration.

This was an exploitable vulnerability on universal verifiability which in the worse case could have allowed an adversary to drop all the ballot boxes containing the ballots of the honest voters and substitute them for ballot boxes containing entirely adversarially controlled voters. This an example of a state vulnerability since it exploits the `verifier` and `control components` have different view of the election configuration (Category $\mathbf{3.1}$). Details of this vulnerability can be found in Section 4.2 of the examination report [45] and in Appendix A.3.

4.4.4. V4: Triggering an Early Tally. The system was reworked between 2023 and 2024 to ensure that each control component agreed on which ballots to tally before tallying commenced; this occurred after vulnerability V1 had been patched. This was implemented by introducing a new micro-service which committed to the votes which would be tallied; the votes were still shuffled and decrypted by the existing micro-service used for this purpose but rather than using the current state of ballot box it would use the set of votes saved by the new micro-service.

This new micro-service had no restriction on when it could be called. For example, if this service was called when only a single vote had been cast, it would commit to decrypting only that ballot. Though, the control component would not actually do the decryption until voting had closed.

Thankfully, this privacy vulnerability would have been detected if exploited, but it still falls within the threat model. This is an example of a state vulnerability, specifically Category $\mathbf{3.2}$. That this vulnerability occurred even after V1 was known motivates the need for a more principled rather than ad-hoc addressing of these kind of vulnerabilities. Details of this vulnerability can be found in Section 4.1 of the examination report [45] and in Appendix A.4.

4.4.5. V5: Messing with Zip Files. The verification data was provided to the `verifier` in the form of three zip files, though this has reduced to two in version 1.5 of the system. The first contains data relating to the election context, the second to the `setup phase`, and the third to `voting` and `tally` phases. The `verifier` runs twice, once at the end of the `setup phase` with the first two zip files and once at the end of the `tally phase` with the first and final zip files. The auditor (human) using the `verifier` (software) is instructed to check the hash of the election context zip file is the same in both runs; this was designed to ensure that the files relating to election context were unchanged as required by the verifier specification.

Despite the auditor's manual check, since the zip files are extracted into the same folder, this mechanism is easily avoided by moving the files from the election context zip file into the other zip files. Now different election context files can be used in each run but, because they are not located in the election context zip file, the hash of that zip file will remain unchanged. We were somewhat astounded that we could not find any way to exploit this weakness; it seems that the consistency checks performed during verification are sufficient to prevent this weakness being exploited. We classify this as an input/output vulnerability because the verifier has the data to check but does not. Details of this vulnerability can be found in Section 4.4 of the examination report [45] and in Appendix A.5.

4.4.6. V6: Adding Too Many Voters. The `online control components` receive the election configuration including the number of voters at the start of the election;

during the setup phase, they create the cryptographic materials the voters need. This functionality is exposed by an API which allows voters to be processed in batches.

Prior to 2023 there was no check that the total number of voters processed was less than or equal to the total number of voters in the election. This weakness on its own is not exploitable because the verification of the setup phase would catch an attack; however, it has an interesting interactions with V1 which made analysis more complicated. Specifically, if these voters were added after setup phase and hence verification of the setup phase had already occurred, there was no check in the system which would catch this vulnerability being exploited. Only once the auditors tried to verify the election at the end would any votes arising from these additional voters be identified as problematic. This is an example of a state vulnerability (Cat. **3.3**), because the component does not check the consistency of the current action with prior information. Details of this vulnerability can be found in Section 5.2.3 of the examination report [40] and in Appendix A.6.

4.4.7. V7: Permuting the Relationship between IDs and Keys. As already noted, the construction of the cryptographic material for each voter is accomplished by the setup component and the online control components working together. Prior to 2022 when the setup component received responses from the online control components it would check that the set of voters ids it received back matched the set it had sent out but *not* that the *order* matched; it would then override it's prior view of the order of voter ids with those of the first online control component (CC1).

This allowed a corrupt online control component to permute the relationship between cryptographic material and voters halfway through the setup phase. Understanding the implications of this weakness, and if it could be exploited, is tricky. It was much easier to get the vendor to patch the weakness than work through the details, which would have been only of historical interest once it was patched. This is an example both of an input/output and a state vulnerability (Cat. **3.2**) because the component does not check the input's internal consistency or consistency with prior information. Details of this vulnerability can be found in Appendix B of the examination report [44] and in Appendix A.7.

4.4.8. V8: No Filtering. The requirements for universal verifiability in the Swiss legislation include filtering out invalid ballots during tallying; the system did not do this. There is a mechanism based on trusting the setup component which filters some invalid ballots but the setup component is not trusted for universal verifiability. It seems likely that the vote counting software, which falls outside the e-voting system and also tabulates ballots cast over other channels, would have complained about these invalid ballots. We classify this vulnerability as Category **1.1**, since the specification should have required filtering to comply with the requirements and did not. Additional details can be found in [40] and in Appendix A.8.

4.4.9. Other Vulnerabilities. We also uncovered other vulnerabilities and non-confirmations with the legislated requirements in the SwissPost system; these can be found in the examinations reports linked and available on the Swiss Federal Chancellery website.The common theme for not including them in this paper is that they are tied to requirements specific to Switzerland and don't seem to generalize to other contexts.

4.4.10. Summary. We have presented eight weaknesses in the SwissPost voting system, four of which (V2,V3,V4,V8) could have been exploited to break the main security properties claimed; once the vulnerabilities are known, it is possible to rule out that they were exploited. Four of these vulnerabilities occurred because of a lack of verification of state, two due to failures in the handling of input, and one due to both; Table 1 in our introduction summaries the categorization.

5. The Belenios System

Belenios [11] is built upon Helios [1]; both systems are publicly available on their respective websites and anyone can setup an election. Belenios improves upon Helios in eligibility verifiability; like most long standing e-voting systems, Belenios has several variants, including BeleniosRF [46] and BeleniosVS [47]. Even within the main version, there are differences between the version described in the paper [11], the specification [48], and the implementation [49] which go beyond the natural expansion of details one would expect to see; we emphasize that all the examples of divergence we encountered seemed well motivated, but we think it is important to note that description of Belenios that follows is designed to be closer to the code and may therefore differ from the paper and specification. When describing the vulnerabilities, we will describe the issues that appear in the implementation, though they may also arise in the specification.

5.1. Protocol Overview

The details below are based on the version of Belenios contained in commit 91d8ca1201702a1cd7a36a0de1c78a3196fbf8ce of the Belenios git repository [49].

The principal participants in the protocols are the voters, voting server, credential authority, tellers, and auditors. The voting server functions as the bulletin board and most components interact directly with it, see Figure 4. The protocol proceeds in a number of phases as we discuss detail below; for simplicity we will present the case where ballots are tallied homomorphically even though the Belenios implementation also supports shuffling.

	Tellers	Credential Authority	Voting Server	Auditors
Verifiability	✗	1 of 2	1 of 2	✓
Privacy	$\leq t$	✗	✗	✓

TABLE 3. BELENIOS REQUIREMENTS

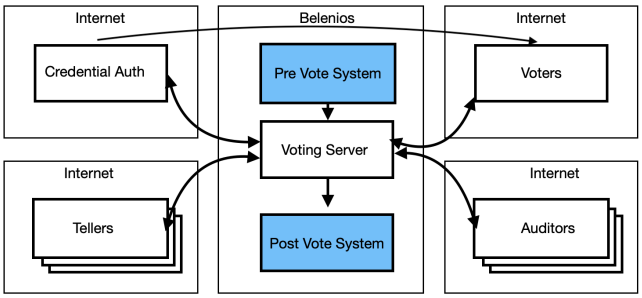

Figure 4. Overview of Belenios Voting System

Setup Phase The `pre vote system` configures the `voting server` with the parameters of the election including:

- the number of voters and tellers, and
- the identities of the `credential authority`, `tellers`, and `voters`,
- the questions which will be asked and the allowed responses.

The `credential authority` then generates private credentials for all `voters` which it distributes to them. It provides the `voting server` the list of public credentials paired with the corresponding voter identities. The `tellers` generate key shares which are combined to form the election public key. Once these two steps have occurred the election can be finalized with the public key material being made publicly available by the `voting server`. Optionally, the `voting server` also sends passwords, it generates, to the `voters` at this point but by default these passwords are not sent until a vote is received from the `voter` as explained below.

Voting Phase In the voting phase `voters` can construct ballots using their private credential which they submit to the `voting server`. The `voting server` checks the ballot is valid with respect to the claimed public credential and then emails the `voter` a password to use to confirm their vote. Each confirmed ballot is added to the list of public election data. It is assumed that the `auditors` are contentiously polling the public data to check that it is valid and that no data previously present has been removed.

Tally Phase Once voting is over, the `voting server` homomorphically combines the ballots and each of the `tellers` contributes their decryption shares which are combined to produce the decrypted tally. This election outcome is then used in some other system, which we label `post vote system`, which sits outside the scope of the Belenios system; unlike the SwissPost system where the `post vote system` is essentially the election result dissemination system for Swiss elections, the use of Belenios by a wider variety of users running different elections means that the `post vote system` varies greatly between election instances.

Belenios, as normally deployed, is unusual in that the authorities, and particularity `tellers`, run light clients in a web browser. The system moves much of the verification checks that they would normally do to the `auditor(s)`. We will spend more time covering the Belenios vulnerabilities since there is no existing writeup to reference.

5.2. Security Requirements

Belenios has fewer security requirements than the Swiss Post system; they are also of a different kind being self-imposed rather than legislated. A derivation of Belenios used for French Legislative Elections called French Legislative E-Voting Protocol (FLEP) does need to operate within a legal framework. We will comment on how recent attacks on FLEP [32] fit into our categorization later (Sec. 6).

The basic variant of Belenios aims for roughly similar privacy and universal verifiability to the SwissPost system; it does not aim for individual verifiability but does provide stronger eligibility verification. The system also has a slightly unusual security model when it comes to the `voting server` which will be relevant for vulnerability V10; specifically the `voting server` is not generally considered to be trusted for either privacy or universal verifiability but its actions which are monitored by the `auditors` throughout the election are essentially taken as honest, because deviation from the honest behavior would cause the `auditors` to complain.

Table 3 presents a simplified version of the security claims in [11]. For verifiability it assumed that either the `credential authority` or `voting server` is honest; if neither was honest they could stuff the ballot box which breaks Belenios' verifiability property. For privacy, it is assumed that a threshold of the `tellers` is honest. Both properties require that the auditors are honest.

5.3. Vulnerabilities

We will present two weaknesses on the deployed Belenios implementation; one of which was exploitable to break privacy. If the attack had been exploited it would have been detected but not until the privacy breech had occurred.

5.3.1. V9: Auditing too late. The main role of the `talliers` is jointly generate the key used to encrypt votes and then to jointly decrypt the anonymized ballots during the `tally phase`. It is crucially important that they decrypt only the anonymized ballots, anonymized either because they have been homomorphically combined or shuffled, and not another ciphertext of the adversaries choosing; this is normally accomplished by the `teller` checking the ciphertext(s) they are decrypting has a valid chain of evidence from the submitted ballots. However, in Belenios, as implemented, these checks are deferred to the `auditor` and the `teller` decrypts the ciphertexts it is instructed to by the `voting server`; this ought to be fine because the `auditors` have already checked that these ciphertexts are correctly computed.

The vulnerability arises because, while the `auditors` did receive the full bulletin board, they only checked the information related to tallying once the `tally phase` is over; this occurs in both the implementation and specification. This means that while the `auditors` could check that the ciphertexts which were going to be decrypted where valid they didn't until after decryption occurred; this was patched by having the `auditors` check this data as soon as it becomes available.

This vulnerability is hard to categorize because it arises from a shift in the expectations on the `auditors` without following through the change to their actions; we, nevertheless, categorize this as a vulnerability in the specification (Cat. $\mathbf{1.1}$) and a state vulnerability because it arises from the `auditor` not taking actions which it needed to for security at the correct phase of the election.

5.3.2. V10: Ghost Voters. The `voting server` is configured with a list of `voters` from the `pre vote system`. The `credential authority` later provides a list of public credentials paired with voter identities; various checks were preformed but it was not checked that all `voters` received a public credential. The `voters` without a public credential were unable to vote but would still receive a password by email if the system was configured to send passwords before votes were received. The error message received by the voter when trying their password suggested that the `voting server` rather than the `credential authority` had misbehaved; in short, while an attack would have been detected, voter confidence could have been badly damaged. We consider this vulnerability to fall into the state category, specifically Category $\mathbf{3.3}$ since the `voting server` does not check the consistency of the input with the previous data.

6. Categorizing Other Recent Vulnerabilities in Deployed E-Voting Systems

In this section, we will discuss how recent vulnerabilities discovered in other deployed schemes relate to the categories we have given. In doing so, we aim to restrict ourselves to bugs found in systems that either fairly mature or closely related to either the SwissPost system or Belenios. As we discussed in the introduction, while the number of relatively mature systems is increasing it is still small; consequently, the number of relevant attacks to categorize is not large.

6.1. A Privacy Attack on the Swiss Post E-voting System

Cortier et al. presented several related privacy weaknesses on the SwissPost system at Real World Crypto 2022 [28]; to our knowledge, this is the only exploitable (against the main security properties) vulnerability found in the SwissPost voting system in versions `0.8` to `1.4` other than the four we present here. The most basic version of the attack exploited the fact that the specification did not require the talliers to check the zero-knowledge proofs of the previous mixers before decrypting resulting in a Category $\mathbf{1.1}$ vulnerability (in the specification). This attack was made worse by the dishonest mixer creating fake ballot boxes which the honest control component would nevertheless process; this is a Category $\mathbf{3.3}$ vulnerability since it exploits the honest control component not checking the input against its own view of the election state.

6.2. Reversing, Breaking, and Fixing the French Legislative Election E-Voting Protocol

Debant and Hirschi show two vulnerabilities on the FLEP protocol used for French legislative elections [32]; this protocol is derived from Belenisos. We will follow their convention by labeling these vulnerabilities V_1 and V_2.

- V_1 can be exploited to break the individual verifiability of the system by exploiting missing checks on the ballot references the voter receives; these references are used to check that the ballot was collected. The missing checks mean that the reference which is displayed to the voter may have no relation to the ballot which was cast.
- V_2 exploits the fact that ballots are not tied to any specific ballot box. An adversary can utilize this vulnerability to move ballots between ballot boxes; this is similar in effect to vulnerability V2 we found on the SwissPost system.

Categorizing these vulnerabilities is complicated because FELP does not properly meet any of the four key marks of maturate we described in the introduction, though the surrounding legislation does partly cover the security claims aspect. In particular, there was no complete system specification. Ideally, both of these vulnerabilities should have been caught at the specification level but we would expect V_2 to have been missed in the formal analysis which historically has not captured multiple ballot boxes. On this basis we categorize V_2 is a state vulnerability; the categorization of V_1 would depend on what details of the ballot references appeared in the specification.

6.3. Attacks in Breaking Verifiability and Vote Privacy in CHVote

Cortier, Debant,and Gaudry [50] discovered nine attacks on the CHVote system which was to be used for elections in the Swiss canton of Geneva. We will adopt the convention of referring to these attacks by the section of their paper in which it is described. CHVote is a much more mature system than FELP but is not complete in that some components required to deploy the system are not implemented. Unlike Belenios and the SwissPost system, the formal analysis of CHVote is absent and the informal analysis has been much less extensive; this is reflected in the vulnerabilities that Cortier, Debant,and Gaudry found.

The first several vulnerabilities (3.1, 3.2, 3.3) relate to weaknesses in the use of oblivious transfer [51] for individual verifiability; these vulnerabilities would have been caught if the protocol described in the specification been analyzed for security, consequently we categorize these in $\mathbf{1.1}$. Likewise, the privacy vulnerability 3.6 would have been caught by an analysis of the specification.

Vulnerability 3.5 and 4.2 involves the implementation failing to check the proof of shuffle of one of the authorities; we would classify this as failing to follow the specification (Category $\mathbf{1.2}$). There is no vulnerability in the proof verification but rather the code responsible validating the proof fails to do so; if this failure to check was dependent on adversary behavior we would have instead classified this as an input/output bug.

Vulnerability 4.1 involves a dishonest election authority claiming to have received a different ballot than that submitted by the voter in way which does not provide recourse to resolve disputes. Cortier, Debant, and Gaudry claim that this vulnerability would be covered by SwissPost bug bounty under individual verifiability but it is unclear what part of that legislated security requirement is being violated. There is a requirement in OEV (11.11) relating to the ability to resolve certain kind of disputes and we would consider this vulnerability to violate that requirement. This vulnerability would fall within Category $\mathbf{1.1}$, since the system specification does not ensure the required property.

The final vulnerability 4.3, which is not exploitable in the OEV threat model, allows the online election authorities to stuff the ballot box without detection; this is an example of a state vulnerability since the authorities modify data which should have been fixed at an earlier time point.

7. Discussion

We will now begin a discussion of why these vulnerabilities maybe occurring and what can be done about it.

7.1. Why Are These Vulnerabilities Occurring?

We speculate that the categories of vulnerabilities have very different causes.

7.1.1. Category 1 (Specification). These vulnerabilities seem to be strongly correlated with immature systems. It seems that the kind of audit process already enshrined by the Swiss legislation is sufficient to handle this errors.

7.1.2. Category 2 (Input/Output). Recall that we define the Input/Output category to cover vulnerabilities which arise because the code handling the input and output values to the core protocol and primitives deviates from the expected behavior under adversarial input.

Of the weaknesses we present, those pertaining to input/output are V2,V5, and V7. Of these only V2 is known to have been exploitable; we don't actually know that V5 and V7 aren't exploitable, these are the only weaknesses that we couldn't decide either way, highlighting the (at least to us) unexpected complexity of this category. In all three cases the system will not have any correctness issues under honest usage because of these bugs; they only occur if input is given to participants which the provided implementation of the other participants would never produce. We concluded from this that implementers likely wrote something which was correct without adequately accounting for the behavior of the code under adversary input.

7.1.3. Category 3 (State). Of the weaknesses we discovered, those pertaining to the state are V1,V3, V4, V6, V7, V9, V10. Of these V3, V4, V9 are known to have been exploitable. We can further breakdown the classification of these weaknesses into three main subcategories:

1) honest parties have differing views without detection (V3,V10),
2) parties not enforcing consistency with the current temporal state (V1,V4,V9)
3) parties not enforcing consistency with the current non-temporal state (V6,V7)

Evaluating the contributing factors to this category is more difficult. We will detail, for each of the subcategories individually, why the corresponding vulnerabilities were not caught by the formal security analysis and any obvious contributing factors to the error occurring.

Subcategory 3.1 With the first subcategory, V3 (spoofing of elections) is beyond the scope of the formal security analysis which only considers a single ballot box. V10 (ghost voters) is more complicated, the paper implicitly requires that the number of public credentials provided be checked but both the specification and code do not. The (principal) cause of V3 seems fairly straightforward, the implementation had state which was not adequately captured or verified in the specification; this is also true of V10.

Subcategory 3.2 This subcategory has the clearest explanation. Looking at the specifications for these systems, checking the current state of the election is not part of the algorithms described; this is intended to be obvious from the wider description of the protocols' phases. We note that V9 (the privacy vulnerability in Belenios) actually does occur at the level of the specification,

which did not require the auditors to check the product ciphertext until the tally was complete. However, the issue does not arise in the formal security analysis of privacy which assumes that the tellers compute the product ciphertext for themselves.

Subcategory 3.3 These vulnerabilities are again missed by the formal security analysis because it does not cover batching which is exploited in V6 (adding extra voters) and in the case of V7 (permuting of the association between voters and key material) because the sending of the verification card ids is not covered in the specification. This seems to have similar causes to subcategory 3.1 which is not surprising since the two categories are similar; the first capturing inconsistencies between components and the second inconsistencies within components.

7.1.4. Summary. The root cause of the input/output errors seem to be a lack of "professional paranoia" to the part of the developers, to borrow a phrase from the book "Cryptographic Engineering" [52]. In other words, the developers thought too much about what the code was supposed to do and not what it would do, under adversarial input. For the state related vulnerabilities there seems to be two primary issues at play:

- implementations which involve protocol complexities not covered by the specification and analysis,
- and requirements which are implicit in the specification not being included in the implementation.

In other cases multiple layers/versions of specification exist with those the implementation follows differing from those which were subjected to formal security analysis.

Both systems we analyzed differ from the standard academic model of how e-voting schemes are presented. In the SwissPost case, by the use of microservices and in the Belenios case by pushing more verification to the auditors. Both of these different layouts could be modeled and formally analyzed but both systems seem to have analyzed assuming a more standard deployment layout. We do not have a complete picture of how e-voting systems are deployed; from what we do know, we would assume that most deployed systems deviate from the academic norm and hence from the norm of security analysis.

Most of the vulnerabilities we discuss do not appear at the level of the cryptographic primitives and core algorithms, even if they did appear in the specification. This means that in systems like ElectionGuard [9] such vulnerabilities are likely to appear in how the Software Development Kit (SDK) is used rather than in the part of the system which has seen the most security auditing.

7.2. What Can Be Done? - Easy Wins

The easy changes that could be made to the process of design, documentation, analysis, and implementation to prevent these vulnerabilities from occurring again are different for the two categories.

7.2.1. Input/Output. The input/output vulnerabilities are the smaller of the two categories and less frequently exploitable, though this may reflect on the analysis methodology. The general techniques around sanitizing input would seem to address this issue without need for any change to design, documentation, or analysis of e-voting systems.

7.2.2. State. The vulnerabilities arising from the mismanagement of state are more complicated to resolve. The current norm in the specification and security definitions is to encode implicit constraints on valid flows through the phases, as well as removing complexities of the protocol which needed careful handling of state to secure. There is certainly low-hanging fruit in terms of documenting these implicit assumptions and checking to see if they seem plausible implemented; our subcategories for this class of vulnerability could itself be used as such a checklist:

The parties ensure the consistence of their election views This is the solution now implemented in the SwissPost system where the hash of the election event configuration is included as auxiliary/associated data in various cryptographic operations; this ensures that if the honest parties don't agree the protocol will abort.

Functionality should only be available in the expected phase The honest parties should keep track of what phase the election is up to and each method exposed by the honest parties should only allow itself to be called in the appropriate phase; this is now implemented in both the Belenios and SwissPost systems.

Input should be checked against the party's view Upon receiving some external input, this input should be validated against the existing view of the party whenever possible.

We note that many of the state bugs can be expressed in terms of reachability; particularly, can core functions be reached on input which the specification says should not be allowed. If the specification contains these constraints then fuzzing and related techniques can be used to check the implementation follows the specification.

7.3. Call for Better Formal Analysis

A more robust but complicated option would be to change the specification and security definitions to make less implicit assumptions; the definitions would then capture more attacks. However, there are many tensions that make this difficult and such definitions and specifications would likely be too complex to reason about on paper. We hope that the vulnerabilities we discovered will, at least, help to facilitate an incremental progression in the models being checked to capture these new known attacks. This actually already partially occurred in the SwissPost system where the symbolic models where expanded to capture a privacy attack we reported.

8. Conclusion

In this paper we have presented ten new weaknesses which were present in the SwissPost and Belenios verifable e-voting systems; all the vulnerabilities have now been patched. Fiver of these weaknesses were exploitable in the threat model either to break privacy or to tamper with the election result but the evidence suggests this did not occur. We categorize these vulnerabilities and provide discussion on possible causes and mitigations to the category rather than just the specific instance of vulnerability. Our hopes for this paper are two fold:

- First, that it serve as a guide to vendors and other stockholders to check for vulnerabilities in these categories in their e-voting systems.
- Secondly, that it encourage future security definitions for electronic voting systems to make less implicit assumptions on state.

We conjecture that the framework through which the designers and auditors of the core system interact with the system implementers needs to be revised if we wish to reduce the number of exploitable security vulnerabilities missed in the specification and security analysis. However, actually solving that issue is an open problem.

Acknowledgments

Thomas Haines is the recipient of an Australian Research Council Australian Discovery Early Career Award (project number DE220100595). He is also one of the independent experts commissioned to examine the compliance of the SwissPost system with the requirements under federal law [53]; the views, opinions, and/or findings expressed here are those of the authors and should not be interpreted as representing any other party.

Thomas Haines would like to thank Olivier Pereira and Vanessa Teague for invaluable discussions; for several examination rounds, we worked jointly on analyzing the specification and proofs of the SwissPost system. Thomas worked independently on the source code examination and is solely responsible for any mistakes in this paper.

Early Version

This preprint is being released in order to facilitate discussion and feedback; we may make significant changes to the manuscript in future.

Ethical Considerations

Our research potentially impacts several groups. We principally considered the impact our research according to the principles of Beneficence and Respect for Persons.

The electorates whose elections occurred on these systems and others impacted by the outcomes of the elections. There are two main harms possible arising from our research which we considered, both arising from misuse of vulnerabilities we might discover. The first was the possibility that a vulnerability we discovered could be used against the system; the second was that the vulnerabilities might be used to create distrust in an election result.

The first issue is reasonable straightforward to mitigate by following best practices in operational security and responsible disclosure. Two advantages in this regard specific to electronic voting is that elections are time bound events and so vulnerabilities can often be patched before any chance of exploitation occurs. The other is that vulnerabilities often require breaching significant operational security mechanisms to exploit and in some cases would be detected.

The second issue of using our results to create distrust in election outcomes is harder to mitigate since such distrust is often irrational. We primarily address this through the tone and composition of our writeup. We emphasize all available evidence that the vulnerability wasn't exploited

The developers of the systems We mitigated reputational and other harm to the developers systems by coordinating disclosure and mitigation of the vulnerabilities. We also mitigate the harm by presenting our results in a fair and balanced way which does lead readers to have an unfair view of the work of the developers.

The decision to proceed with the research was taken based on a belief that we could mitigate the possible harm of the research, as detailed above, while it was likely to generate significant benefit. Likewise the decision to publish is based on a view that benefits of the lessons contained in this report outweigh the possible harms.

References

[1] B. Adida, "Helios: Web-based open-audit voting," in *USENIX Security 2008*, P. C. van Oorschot, Ed. USENIX Association, Jul. / Aug. 2008, pp. 335–348.

[2] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. T. Sherman, and P. L. Vora, "Scantegrity: End-to-end voter-verifiable optical-scan voting," *IEEE Secur. Priv.*, vol. 6, no. 3, pp. 40–46, 2008.

[3] C. Burton, C. Culnane, and S. A. Schneider, "vvote: Verifiable electronic voting in practice," *IEEE Secur. Priv.*, vol. 14, no. 4, pp. 64–73, 2016.

[4] R. Haenni, E. Dubuis, R. E. Koenig, and P. Locher, "Chvote: Sixteen best practices and lessons learned," in *E-VOTE-ID*, ser. Lecture Notes in Computer Science, vol. 12455. Springer, 2020, pp. 95–111.

[5] J. Benaloh, M. D. Byrne, B. Eakin, P. T. Kortum, N. McBurnett, O. Pereira, P. B. Stark, D. S. Wallach, G. Fisher, J. Montoya, M. Parker, and M. Winn, "Star-vote: A secure, transparent, auditable, and reliable voting system," in *EVT/WOTE*. USENIX Association, 2013.

[6] R. Haenni, "Swiss Post Public Intrusion Test: Undetectable attack against vote integrity and secrecy," https://e-voting.bfh.ch/app/download/7833162361/PIT2.pdf?t=1552395691, Mar. 2019.

[7] T. Haines, S. J. Lewis, O. Pereira, and V. Teague, "How not to prove your election outcome," in *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2020, pp. 644–660.

[8] M. A. Specter, J. Koppel, and D. J. Weitzner, "The ballot is busted before the blockchain: A security analysis of voatz, the first internet voting application used in U.S. federal elections," in *USENIX Security 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, Aug. 2020, pp. 1535–1553.

[9] J. Benaloh, M. Naehrig, O. Pereira, and D. S. Wallach, "ElectionGuard: a cryptographic toolkit to enable verifiable elections," in *USENIX Security 2024*, D. Balzarotti and W. Xu, Eds. USENIX Association, Aug. 2024.

[10] Swiss Post, "Protocol of the swiss post voting system," https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation, Oct. 2025.

[11] V. Cortier, P. Gaudry, and S. Glondu, "Belenios: A simple private and verifiable electronic voting system," in *Foundations of Security, Protocols, and Equational Reasoning*, ser. Lecture Notes in Computer Science, vol. 11565. Springer, 2019, pp. 214–238.

[12] V. Cortier, D. Galindo, R. Küsters, J. Mueller, and T. Truderung, "SoK: Verifiability notions for E-voting protocols," in *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016, pp. 779–798.

[13] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi, "SoK: A comprehensive analysis of game-based ballot privacy definitions," in *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 499–516.

[14] Swiss Post, "Symbolic proofs of swiss post voting system," https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/documentation-1.6.0.0/Symbolic-models, 2024.

[15] V. Cheval, V. Cortier, and A. Debant, "Election verifiability with ProVerif," in *CSF 2023 Computer Security Foundations Symposium*. IEEE Computer Society Press, Jul. 2023, pp. 43–58.

[16] S. Baloglu, S. Bursuc, S. Mauw, and J. Pang, "Election verifiability revisited: Automated security proofs and attacks on helios and belenios," in *CSF 2021 Computer Security Foundations Symposium*, R. Küsters and D. Naumann, Eds. IEEE Computer Society Press, 2021, pp. 1–15.

[17] V. Cortier, C. C. Dragan, F. Dupressoir, and B. Warinschi, "Machine-checked proofs for electronic voting: Privacy and verifiability for belenios," in *CSF 2018 Computer Security Foundations Symposium*, S. Chong and S. Delaune, Eds. IEEE Computer Society Press, 2018, pp. 298–312.

[18] T. Haines and P. B. Rønne, "New standards for e-voting systems: Reflections on source code examinations," in *Financial Cryptography Workshops*, ser. Lecture Notes in Computer Science, vol. 12676. Springer, 2021, pp. 279–289.

[19] A. Sutopo, T. Haines, and P. B. Rønne, "On the auditability of the estonian IVXV system - and an attack on individual verifiability," in *FC Workshops*, ser. Lecture Notes in Computer Science, vol. 13953. Springer, 2023, pp. 19–33.

[20] 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.

[21] O. Goldreich, *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[22] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *CRYPTO'84*, ser. LNCS, G. R. Blakley and D. Chaum, Eds., vol. 196. Springer, Berlin, Heidelberg, Aug. 1984, pp. 10–18.

[23] J. D. Cohen and M. J. Fischer, "A robust and verifiable cryptographically secure election scheme (extended abstract)," in *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*. IEEE Computer Society, 1985, pp. 372–382. [Online]. Available: https://doi.org/10.1109/SFCS.1985.2

[24] J. C. Benaloh and M. Yung, "Distributing the power of a government to enhance the privacy of voters (extended abstract)," in *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing, Calgary, Alberta, Canada, August 11-13, 1986*, J. Y. Halpern, Ed. ACM, 1986, pp. 52–62. [Online]. Available: https://doi.org/10.1145/10590.10595

[25] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," in *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, ser. Lecture Notes in Computer Science, W. Fumy, Ed., vol. 1233. Springer, 1997, pp. 103–118. [Online]. Available: https://doi.org/10.1007/3-540-69053-0_9

[26] D. Chaum, "Verification by anonymous monitors," in *CRYPTO'81*, A. Gersho, Ed., vol. ECE Report 82-04. U.C. Santa Barbara, Dept. of Elec. and Computer Eng., 1981, pp. 138–139.

[27] ——, "Surevote: technical overview," in *Proceedings of the workshop on trustworthy elections (WOTE'01)*, 2001.

[28] V. Cortier, A. Debant, and P. Gaudry, "A privacy attack on the swiss post e-voting system," in *RWC 2022-Real World Crypto Symposium*, 2022.

[29] L. Hirschi, L. Schmid, and D. A. Basin, "Fixing the achilles heel of E-voting: The bulletin board," in *CSF 2021 Computer Security Foundations Symposium*, R. Küsters and D. Naumann, Eds. IEEE Computer Society Press, 2021, pp. 1–17.

[30] Swiss Post, "E-voting architecture document v1.4.1," https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/raw/master/System/SwissPost_Voting_System_architecture_document.pdf, 2025.

[31] T. Haines, O. Pereira, and V. Teague, "Running the race: A swiss voting story," in *E-Vote-ID*, ser. Lecture Notes in Computer Science, vol. 13553. Springer, 2022, pp. 53–69.

[32] A. Debant and L. Hirschi, "Reversing, breaking, and fixing the french legislative election E-voting protocol," in *USENIX Security 2023*, J. A. Calandrino and C. Troncoso, Eds. USENIX Association, Aug. 2023, pp. 6737–6752.

[33] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis, "Chvote system specification," *IACR Cryptol. ePrint Arch.*, p. 325, 2017.

[34] V. Cortier, A. Debant, O. Esseiva, P. Gaudry, A. Høgåsen, and C. Spadafora, "A practical and fully distributed e-voting protocol for the swiss context," *IACR Cryptol. ePrint Arch.*, p. 1625, 2025.

[35] Swiss Post, "Source code of the swiss post voting system," https://gitlab.com/swisspost-evoting/e-voting/e-voting, Oct. 2025.

[36] ——, "E-voting libaries of the swiss post voting system," https://gitlab.com/swisspost-evoting/e-voting/e-voting-libraries, Oct. 2025.

[37] ——, "Cryptoprimativies of the swiss post voting system," https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives, Oct. 2025.

[38] ——, "Verifier of the swiss post voting system," https://gitlab.com/swisspost-evoting/verifier/verifier, Oct. 2025.

[39] Swiss Federal Chancellery, "Explanatory report of the federal chancellery ordinance on electronic voting - 1 july 2022," Available from https://www.bk.admin.ch/dam/bk/en/dokumente/pore/E_Voting/Explanatory%20report%20PoRO%20and%20OEV%202022.pdf, Jul. 2022.

[40] T. Haines, O. Pereira, and V. 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%201%20and%202%20Final%20Report%20Thomas%20Haines,%20Olivier%20Pereira,%20Vanessa%20Teague%2031.07.2023.pdf, Jul. 2023.

[41] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–88, 1981.

[42] S. Bayer and J. Groth, "Efficient zero-knowledge argument for correctness of a shuffle," in *EUROCRYPT 2012*, ser. LNCS, D. Pointcheval and T. Johansson, Eds., vol. 7237. Springer, Berlin, Heidelberg, Apr. 2012, pp. 263–280.

[43] T. Haines, O. Pereira, and V. Teague, "Report on the swiss post e-voting system," https://www.newsd.admin.ch/newsd/message/attachments/71147.pdf, Mar. 2022.

[44] ——, "Addendum on the swiss post e-voting system," https://www.bk.admin.ch/dam/bk/en/dokumente/pore/E_Voting/Examination_Reports_March2023/Scopes%201%20and%202%20Final%20Report%20Addendum%201%20Thomas%20Haines,%20Olivier%20Pereira,%20Vanessa%20Teague%2021.11.2022.pdf, Nov. 2022.

[45] T. 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%201%20and%202%20Final%20Report%20Thomas%20Haines%2011.07.2024.pdf, Jul. 2024.

[46] P. Chaidos, V. Cortier, G. Fuchsbauer, and D. Galindo, "BeleniosRF: A non-interactive receipt-free electronic voting scheme," in *ACM CCS 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM Press, Oct. 2016, pp. 1614–1625.

[47] A. Filipiak, "Design and formal analysis of security protocols, an application to electronic voting and mobile payment. (conception et analyse formelle de protocoles de sécurité, une application au vote électronique et au paiement mobile)," Ph.D. dissertation, University of Lorraine, Nancy, France, 2018.

[48] Stéphane Glondu, "Belenios specification - version 3.0," https://www.belenios.org/specification.pdf.

[49] C. Inria, "Belenios github," https://github.com/glondu/belenios.

[50] V. Cortier, A. Debant, and P. Gaudry, "Breaking verifiability and vote privacy in chvote," *IACR Cryptol. ePrint Arch.*, p. 80, 2025.

[51] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," in *CRYPTO'82*, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds. Plenum Press, New York, USA, 1982, pp. 205–210.

[52] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography engineering: design principles and practical applications*. John Wiley & Sons, 2011.

[53] Swiss Federal Chancellery, "Federal government launches examination of new e-voting system," https://www.bk.admin.ch/bk/en/home/dokumentation/medienmitteilungen.msg-id-84337.html, Jul. 2021.

Appendix A. Details of attack on the SwissPost system

In this appendix, we provide further details on the weaknesses on the SwissPost e-voting system. For each weakness, we desribe:

- the powers the adversary would need to exploit it,
- the security mechanism being circumvented (if applicable),
- the steps the attacker would take,
- the security consequences of exploitation, and
- how the weakness was patched.

A.1. V1: API free for all.

Attacker Powers. To exploit this weakness, the adversary would need to control:

- any component authorized to call the associated API. In most cases this is not restricted but some check that the request comes from the `setup component`.

Attack Steps.

- Call the API.

Security Consequences. Two security consequences specific to particular APIs are listed in V4 (privacy breach) and V6 (extra voter credentials issued); beyond these the general implication is disruption by causing issues which will be detected later in the election but without a clear way to resolve them.

Patch. Have each API check that it is being called at the expected phase of the election.

A.2. V2: Shuffling the Proofs of Shuffle

This is the most technically complicated attack we present. The core vulnerability is straightforward, if the filenames and contents being verified are inconsistent the auditors will verify the proofs of shuffles with respect to the wrong input ciphertexts. Actually exploiting this vulnerability involves the adversary taking several actions to ensure various auxiliary checks do not catch the attack. The simplest case to consider is two parallel referendums, where the adversary will replace the result of one referendum with the other without detection.

We include in Figure 5, the code which loaded proofs of shuffle from the online components. The code once executed creates a map from ballot box ids to lists of proofs of shuffles; it does this by first reading in all shuffles (output ciphertexts and proofs) and then grouping these by the ballot box denoted in the payload. Let N*X*BB*Y* denote the contribution of the shuffle payload contribution of the Xth `online control component` to the Yth ballot box and let the expected shuffle order be implicit in the order of the payloads. This code shown will reorder the malicious ordering shown below back to the expected ordering so that it passes the verification of offline shuffle. Recall that the ciphertexts used to verify the final offline shuffle will be retrieved based on the filename, and hence will come from the other ballot box.

Folder BB1: N1BB1 N2BB1 N3BB1 N1BB2
Folder BB2: N2BB2 N2BB2 N4BB2 N4BB1

Attacker Powers. To exploit this weakness, the adversary would need to control:

- `control components` 2, 3, and 4
- the `offline tally component`
- some of the `voters`
- the `voting server`

Security Mechanism being Circumvented. The zero-knowledge proofs of correct shuffle.

Attack Steps.

- During setup, the dishonest `online control components` choose their keys so that their keys cancel each other out.

```
final Map<String, List<ControlComponentShufflePayload>> controlComponentShufflesByBallotBoxId =
electionDataExtractionService.getAllControlComponentShufflePayloads(inputDirectoryPath).stream()
      .collect(Collectors.groupingByConcurrent(ControlComponentShufflePayload::getBallotBoxId));
```

Figure 5. Code for loading proofs of shuffle from online control components

- The adversary ensurers that both ballot boxes have the same number of ballots; this can be done either by voting or not voting using credentials it controls, or be simulating errors when an honest voter tries to vote.
- The dishonest `tally control component` shuffles and decrypts the first shuffle for ballot box two when it should shuffle the last online shuffle for ballot box one; likewise, it shuffles and decrypts the last shuffle for ballot box one when should decrypt the last shuffle for ballot box two.
- The adversary reorders/renames the payload so that file which should contain the final online shuffle for ballot box one contains the first shuffle for ballot box two.

Security Consequences. The results announced for ballot box one are replaced with the results for ballot box (and vice versa) two without detection, breaking universal verifiability.

Patch. This was patched by checking that file names match file contents.

A.3. V3: Spoofing Elections

Attacker Powers. To exploit this weakness, the adversary would need to control:

- the `setup component`
- channel between the `online control components` and the `auditors`

Attack Steps.

- During setup, the dishonest `setup component` sets up two ballot boxes for each ballot box which should exist; lets call these two sets of ballot boxes `real` and `fake`. It then executes the protocol as normal until it is time to send the data to the printing service, it then sends the `real` data only.
- During both the setup verification and tally verification phases, the adversary only sends data relating to the `fake` ballot boxes in its own transmission to the `auditors`; it also drops all payloads originating from the honest `control components` pertaining to the `real` ballot boxes.

Security Consequences. The announced election result matches the `fake` spoofed ballot boxes while all the `real` ballots are dropped without detection.

Patch. This was patched by having all components include hashes of the election configuration (which includes the number of ballot boxes) as auxiliary information in various cryptographic operations; this will cause the protocol to abort if the components do not agree on the election configuration.

A.4. V4: Triggering an Early Tally

Attacker Powers. To exploit this weakness, the adversary would need to control:

- the `voting server`

Attack Steps.

- The adversary calls the `GetMixnetInitialCiphertexts` API of the honest `control components` causing them to commit to the ciphertexts which they will decrypt and mix
- After the tally phase starts the adversary calls `MixDecOnline` API of the honest `control components` causing them to mix and decrypt the ciphertexts they had previously committed to, rather than all the ballots in the ballot box.

Security Consequences. The adversary can get a partial tally of the election revealing additional information about how people voted.

Patch. The `GetMixnetInitialCiphertexts` API was restricted to only activate during the tally phase.

A.5. V5: Messing with Zip Files.

Attacker Powers. To exploit this weakness, the adversary would need to control:

- The channel between the `auditors` and the rest of the system

Security Mechanism being Circumvented. Verification that the election configuration was consistent between the setup verification and tally verification checks.

Attack Steps.

- During both the setup verification and tally verification phases, the adversary takes the zip files going to the `auditors` and creates new zips where the files are located inside the setup and verification zips files where the configuration is left empty.
- The `auditors` check the hash of the now empty zip file is consistent between the two verification phases.

Security Consequences. The adversary can introduce any inconsistent between the election configuration used in the two verification phases without detection by the intended security mechanism. As previously noted, we could not see any way to actually exploit this weakness; in the setup verification phase most incoming data is provided by the `setup component` who is trusted in the threat model and in the tally phase at least one `online control component` is trusted; the crosschecking of data received against that of these trusted components caught every inconsistent in the election configuration we considered.

Patch. The setup verification phase is no longer part of the security protocol, though it still remains part of the wider system for procedural reasons. Since the security protocol now has a single verification by the auditors, an inconsistent between the configurations is no longer relevant.

A.6. V6: Adding Too Many Voters

Attacker Powers. To exploit this weakness, the adversary would need to control:

- The `setup component`
- The channel between the honest `control components` and the `auditors`

Attack Steps.

- The adversary sends requests to the honest `control components` causing them to generate keys for more voters than are actually eligible,
- if this is not occurring after the setup phase using vulnerability V1, the adversary drops the payloads relating to the extra voters before they arrive at the auditors.

Security Consequences. Extra voter credentials are generated in the system without detection; the threat model for eligibility verifiability assumes the `setup component` is honest so this attack is outside the the threat model.

Patch. The online `control components` now check that the number of voters processed is bound by the number of eligible voters.

A.7. V7: Permuting the Relationship between IDs and Keys

Attacker Powers. To exploit this weakness, the adversary would need to control:

- The first `control component`

Attack Steps.

- The first `online control component` permutes the relationship between the list of voter ids it received and the payloads of cryptographic material it produced
- The `setup component` overrides its view of what keys belong what `voter` with the new mapping provided by the adversary.

Security Consequences. We did not fully analyze the impact of this vulnerability and its severity is therefor unknown; in best case, it would have caused errors which would not have been detected by the setup verification, disrupting the election.

Patch. The setup component now checks that the verification card ids received (in the form of a list) match with those sent.

A.8. V8: No Filtering

Attacker Powers. To exploit this weakness, the adversary would need to control:

- some `voters`
- the `setup component`[3]

Security Mechanism being Circumvented. The `setup component` generates an `allow list` which is used to check if votes are valid; however, for universal verifiability the `setup component` is untrusted.

Attack Steps.

1) The `setup component` generates an allow list allowing invalid votes.
2) The adversary controlled `voters` submit invalid ballots matching the allow list.
3) The honest `control checks` checks the invalid ballots against allow list and they pass.
4) The votes are tallied.

Security Consequences. This is a violation of the universal verifiability requirements; it is unclear what the impact on the wider electoral system would be.

Patch. This was fixed by having the `auditors` (who are trusted for universal verifiability) check that invalid ballots are not present in the output.

3. There is a known issue that the `allow list` does not actually enforces ballot validity completely; in this specific case control of the `setup component` is not required.