# Analysis of the Swiss Post e-Voting System

## Audit Scope 1: Cryptographic Protocol

Aleksander Essex

Department of Electrical and Computer Engineering
Western University, Canada
aessex@uwo.ca

November 26th, 2021

Submitted to the Swiss Federal Chancellery

# Table of Contents

## Summary of Findings

I was retained by the Swiss Federal Chancellery to examine the e-voting solution of Swiss Post in relation to the Chancellery's ordinances on electronic voting (OEV), specifically in regards to the cryptographic components and their respective security properties.

Based on my previous experience in cryptographic election verification, my main goal was to assess how or whether the various players could manipulate the constituent zero-knowledge proofs to produce an unintended outcome as a result of common issues like improper parameter testing, insufficient hashing context, parameter trapdoors, etc.

Although I found the Swiss Post system to be highly complex, overall it appears well designed and documented. Furthermore, I believe a serious effort was made by the designers not only to address the specific vulnerabilities found by Teague et al. [11] in the system's predecessor, but also to address the underlying process, design methodology, and culture that led to them.

Acknowledging that I cannot conclude the system is free from vulnerability, my analysis did not find any serious vulnerability. Similarly, I did not find any obvious instance in which the system substantively diverges from the apparent intent of the OEV. I do, however, make several recommendations for improving the system.

## Key Versions of this Report

1. ***Preliminary* Analysis of the Swiss Post e-Voting System**.
   - **Date**: September 1st, 2021
   - **Description**: A preliminary analysis of Swiss Post e-Voting system as of August 2021. Referred to throughout this document as the *preliminary report*.
2. **Analysis of the Swiss Post e-Voting System** (this document).
   - **Date**: November 26th, 2021
   - **Description**: An extension of the preliminary report to include comments on changes made by Swiss Post in response to the preliminary report. Also contains a preliminary analysis of the verifier specification, which was not available for review at the time of the preliminary report.

# 1 Introduction

This report examines Swiss Post's e-voting proposed in relation to the cryptographic components employed to address the Swiss Federal Chancellery's ordinances on electronic voting (OEV).

The cryptographic components of the proposed Swiss Post e-voting system are extensive and complex–likely too complex for most individuals to internalize completely. Nevertheless, this complexity (especially due to Bayer-Groth mixnets and multi-recipient ElGamal) appears necessary to address the unique functional requirements of the electoral system of the prospective users (i.e., cantons) and the specific requirements security requirements of the OEV. As a result, however, the system poses challenges to independent review due to the extent of its technical depth and its documentary breadth.

This review focuses primarily on the individual zero-knowledge proofs made by the system and players (less on the overall proofs of security). Based on my previous experience in cryptographic election verification, my main goal was to assess how or whether the various players could manipulate these zero-knowledge proofs to produce an unintended outcome.

In particular, I examined the degree to which these proofs involved sufficient context in the Fiat-Shamir heuristic hash challenge [2], whether sufficient checking was done of inputs to algorithms [6], and whether the possibility of trapdoored parameters was adequately mitigated [11],[8].

**General Impressions.** My overall impression is that Swiss Post has made a serious and substantive effort to address the OEV requirements and be fully transparent and responsive to the independent examiners. I commend them for dedicating themselves so thoroughly to this process.

My overall impression of the Swiss Post system is that, while complex, it is serious and substantive in terms of the evidence it would generate toward convincing voters and auditors of a genuine election outcome. In particular the approach to the zero-knowledge proofs (Algorithms 6.1-6.9 of the primitives specification) seems more constrained from the attacker's perspective than textbook sigma protocols (e.g., Schnorr, Chaum-Pedersen).

## 1.1 Structure of This Report

This report is an extension to my preliminary report of September 1st. In the interim, Swiss Post continued to make updates to its algorithms and documentation and responded to selected issues raised in the preliminary report.

### Timeline of Events

– **September 1st**: Preliminary report examined the Swiss Post protocol documentation as of August 2021.

– **November 19th**: Swiss Post returned my preliminary report annotated with issue IDs (numbered 1–137 inclusive).[1] They directly responded to issues 73–75, 78, 92, 103, 105, 107, 113–7, 120–2, 123, 133–7.[2] They also included several general responses.

– **November 26th**: This report responds to their responses and makes additional recommendations. It also examines algorithms introduced in updated versions of the documentation, including the verifier specification published after the preliminary report.

Note: Since the preliminary report was written, several algorithm numbers have changed in more recent versions of Swiss Post's documentation. This report maintains the algorithm numbering from the document versions cited in the preliminary report unless otherwise stated.

**Changes Made Since the Preliminary Report**
This report preserves the structure and content of the preliminary report, but adds Swiss Post's responses (and my responses to their responses) inline with the relevant issue. Section 1.2 makes several new recommendations. Section 9.3 has been added discussing the recently added verifier specification.

### 1.2 Key Recommendations

**Key Recommendations of the *Preliminary* Report:** Acknowledging that I cannot conclude the system is free from vulnerability, the analysis in my preliminary report did not identify any serious vulnerabilities or issues. With that said, the preliminary report identified several opportunities for improvement. These recommendations, as well as the progress made since the original report are as follows:

1. Make documentation more explicit about the requirement to ensure all inputs to cryptographic algorithms have the expected form, especially algebraic groups.

   – **Progress:** Some progress has been made, although Swiss Post is hesitant about the computational overhead of explicitly checking the validity of every argument at the input of every algorithm. I have, in turn, suggested better labeling of validity assumptions.

2. Make sure the expected algebraic groups produce the desired outcomes, especially with regard to the distinction between $\mathbb{Z}_q$ and $\mathbb{Z}_q^*$, and the distinction between $\mathbb{G}_q$ (which includes 1) and the *generators* of $\mathbb{G}_q$ which does not.

   – **Progress:** This report contains additional discussion of this point. In essence, Swiss Post is taking an algorithm-by-algorithm approach, which appears to be acceptable.

[1] Due to time, I have not incorporated these IDs into this report.
[2] This reference to these issue IDs is intended for Swiss Post's benefit. This report can still be understood without explicitly mapping IDs to issues.

3. Increase the *default* and *extended* security level. Remove the insecure *testing-only* security level.

   – **Progress:** Some progress has been made, with Swiss Post committing to developing more efficient algorithms supporting the extended key size. The trivial *testing-only* security level has been retained, however.

4. Provide more detail about the consequences of violating major trust assumptions (e.g., print office, quantum computation, physical attacks against the mail system, etc.).

   – **Progress:** Little progress has been made, although these points have been acknowledged and may be addressed in future versions.

**Key Recommendations of *This* Report:** This report includes the responses made by Swiss Post to the preliminary report with some additional responses from me. These responses appear beside the relevant findings below.

1. **Under-specified authentication.** Section 2.8 of the OEV requires "effective authentication." Swiss Post does, however, not fully address the question of voter identity and authentication, especially the question of: (a) what is sufficient to allow someone to cast a vote on another voter's behalf, and (b) how feasible is it. Their explanation is that this authentication is ultimately out their control, and different cantons take different approaches to the problem. At a minimum, however, Swiss Post should still prescribe approaches it considers acceptable.

2. **Labeling trusted parameters in algorithms.** For efficiency reasons, Swiss Post wants to avoid checking the cryptographic integrity and validity of inputs to each and every algorithm. This may be risky. Different algorithms are called by different parties at different times, and there needs to be some guarantee that all data ingested by each algorithm will be have been rigorously checked at the current end-point/execution environment prior to use. If this does not happen within the algorithm itself, it can be harder to track parameter validity across successive function calls. Developers (incl. future Swiss Post developers and independent verifiers) *cannot* be left to their own devices in this regard. Given the industry's [8] and even Swiss Post's [11]) track record of parameter hygiene in the discrete logarithm setting, I recommend that, at a minimum, every algorithm explicitly label which input parameters carry a validity assumption.

3. **Response follow-through**. Swiss Post generically acknowledged many of the minor recommendations I made in my preliminary report, and agreed to address them in future versions of the documentation. However, of the 137 issues they acknowledge in the preliminary report, Swiss Post only directly responded to 22. Most of the other issues still appear to be unaddressed in the updated versions of the documentation. As a concrete example: the crypto primitives specification contains a small mistake in

Algorithm 6.9 (Version 0.9.5). Swiss Post labeled this as issue #130 in its response. This issue, however, is still present in the current version (0.9.8).

4. **Integrity verification.** The verifier specification describes an *integrity verification* category which ensures "data elements correspond to the specification" (i.e., have the correct range, type, set memberships, etc.). However, the verifier specification does not explicitly and exhaustively prescribe data integrity checks for the relevant data elements (e.g., checking group membership, etc). Since the verifier specification may be used by independent (non-Swiss Post affiliated) developers, there can be no room for interpretation.

## 2 Scope of Engagement

I was engaged by the Swiss Federal Chancellery to examine the security of Swiss Post's proposed e-voting solution. They asked me to provide this report setting out my views and recommendations in accordance with Scope 1 (Cryptographic Protocol) outlined in the Chancellery's Audit Concept document.[3] Specifically, I was asked to comment on the Swiss Post system's compliance with the following requirements of Chapter 2 of the OEV Annex:

- **Section 2.5** (Individual verifiability)
- **Section 2.6** (Universal verifiability)
- **Section 2.7** (Ballot secrecy)
- **Section 2.8** (Authentication)
- **Section 2.11** (Proof soundness)
- **Section 2.13** (Definitions and Descriptions)
- **Section 2.14** (Compliance with protocol requirements)

## 3 List of Documents Examined

In preparing this report, I reviewed and relied upon the following documents in addition to any other documents specifically noted elsewhere in this report:

1. **Federal Chancellery Ordinance on Electronic Voting (OEV)**, Swiss Federal Chancellery.
   - **Version**: Draft of 28 April, 2021

[3] Audit Concept for Examining Swiss Internet Voting Systems. Swiss Federal Chancellery. Version 1.3, May 18, 2021.

- **Available**: https://www.bk.admin.ch/dam/bk/en/dokumente/pore/OEV_draft%20for%20consultation%202021.pdf.download.pdf/OEV_draft%20for%20consultation%202021.pdf
- **Description**: Primary legal document outlining relevant technical requirements, particularly the cryptographic requirements for complete verifiability. Referred to throughout this document as the "OEV." All references to "Section XX of the OEV" refer to the Technical and Administrative Requirements of Electronic Voting in the OEV's *Annex* (as distinguished from the OEV's *articles*.)

2. **Cryptographic Primitives of the Swiss Post Voting System**, Swiss Post.
   - **Version examined in the preliminary report**: 0.9.5, 2021-06-22
   - **Version examined in this report**: 0.9.8, 2021-10-15
   - **Available**: https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives/-/blob/master/Crypto-Primitives-Specification.pdf
   - **Description**: Pseudocode specifications of cryptographic functions used by the Swiss Post system. Referred to throughout this document as the *primitives specification*.

3. **Swiss Post Voting System Specification**, Swiss Post.
   - **Version examined in the preliminary report**: 0.9.6, 2021-06-25
   - **Version examined in this report**: 0.9.7, 2021-10-15
   - **Available**: https://gitlab.com/swisspost-evoting/documentation/-/blob/master/System/System_Specification.pdf
   - **Description**: Document describing the steps, phases and procedures of setting up, executing and verifying an election using the Swiss Post system. Referred to throughout this document as the *system specification*.

4. **Swiss Post Voting System architecture document**, Swiss Post.
   - **Version examined in the preliminary report**: 0.9, 2021-07-01
   - **Version examined in this report**: 0.9.1, 2021-08-17
   - **Available**: https://gitlab.com/swisspost-evoting/documentation/-/blob/master/System/SwissPost_Voting_System_architecture_document.pdf

– **Description**: Document describing the software engineering methodology, practices and requirements. Referred to throughout this document as the *architecture specification*.

5. **Protocol of the Swiss Post Voting System**, Swiss Post.
   – **Version examined in the preliminary report**: 0.9.10, 2021-06-25
   – **Version examined in this report**: 0.9.11, 2021-10-15
   – **Available**: https://gitlab.com/swisspost-evoting/documentation/-/blob/master/Protocol/Swiss_Post_Voting_Protocol_Computational_proof.pdf
   – **Description**: Document describing the cryptographic primitives and protocols of the Swiss Post system and the various security proofs thereof. Referred to throughout this document as the *protocol specification*.
6. **Swiss Post Verifier Specification**, Swiss Post.
   – **Version examined in the preliminary report**: Unavailable at time of writing.
   – **Version examined in this report**: 0.9.1, 2021-10-15
   – **Available**: https://gitlab.com/swisspost-evoting/verifier/verifier/-/blob/master/Verifier_Specification.pdf
   – **Description**: Document describing the algorithms and checks that an independent verifier of the Swiss Post system would undertake to verify the correctness and validity of the election parameters, proofs, and ultimately, the election results. Referred to throughout this document as the *verifier specification*.

## 4 Individual verifiability

Section 2.5 of the OEV addresses requirements for individual verifiability:

> *The voter is given a proof … to confirm that the attacker has not altered any partial vote before the vote has been registered as cast in conformity with the system;*

> *The voter is given a proof … to confirm that the attacker has not maliciously cast a vote on the voter's behalf which has subsequently been registered as a vote cast in conformity with the system and counted.*

The Swiss Post system relies on the server responding to a voter's selection with confirmation codes (*choice return codes*) and upon the voter's casting of the ballot (*vote case*

*return code*). See e.g., §20.5 in the protocol specification. This approach to "code voting" mostly common and consistent with the literature. Due to time, I was unable to closely examine this aspect of the system.

## 5 Universal verifiability

Section 2.6 of the OEV addresses requirements for universal verifiability.

> *The auditors receive a proof ... to confirm that the attacker, after the votes were registered as cast in conformity with the system, did not alter or misappropriate any partial votes before the result was determined.*

> *The auditors receive a proof ... to confirm that the attacker did not insert any votes or partial votes not cast in conformity with the system which were taken into account in determining the result.*

The Swiss Post system relies on various zero-knowledge proofs (§6 of the primitives specification) and the verifiable mixnet of Bayer and Groth (§5 of the primitives specification). Due to time, I was unable to closely examine the verifiability of the mixnet. The other zero-knowledge proofs, however, are largely consistent with standard approaches in the literature. Detailed comments on these proofs are given in §9 below.

As a comment on the OEV's definition, the term *universal verifiability* has traditionally been applied in the literature in reference to a proof *anyone* could check/verify, not only a set of auditors. Additional comments are provided below.

## 6 Ballot secrecy

Section 2.7 of the OEV addresses requirements for voting secrecy and absence of premature results.

> *2.7.1: It must be ensured that the attacker is unable to breach voting secrecy or establish premature results unless he can control the voters or their user devices.*

The trust assumptions in the protocol specification document (see, e.g., Tables 1, 2) states that the print office, which "prints the code sheets", is trusted with maintaining the privacy of the voter–candidate–code associations. However, unlike other code-voting systems (e.g., Scantegrity [3]), the code associated with the voter's chosen candidate is encrypted and never directly revealed to any party outside the voter's physical environment or device, which is consistent with requirement 2.7.1.

> *2.7.2: With the exception of the person voting and his or her user device, system participants who have enough information to breach voting secrecy or to collect premature results are not considered protected against the attacker.*

Since Swiss Post presumably would be responsible for mailing the code sheets, I suggest that Swiss Post clarify if they envision acting as the fully trusted Print Office, or as an independent party with comparable trust assumptions.

> *2.7.3: It must be ensured that the attacker cannot take control of user devices unnoticed by manipulating the user device software on the server. The person voting must be able to verify that the server has provided his or her user device with the correct software with the correct parameters, in particular the public key for encrypting the vote.*

The Swiss Post solution uses a browser-based approach for vote casting as opposed to a mobile app-based approach. The pros and cons of each are discussed in the documentation (See §4.1 of the architecture specification). I see the merits of both approaches, however, with browser based voting, verifying the URL and the authenticity of the TLS connection is slightly more the user's responsibility, and in practice could lead to issues resulting from phishing, TLS stripping [4], weak TLS configurations [12] and TLS proxying [7]. However, our survey of the TLS configurations of 100 election websites in 2018 actually found Swiss Post was the only election agency fully protecting against TLS stripping [5].

## 7 Authentication

Section 2.8 of the OEV addresses requirements for voter authentication.

> *It must be ensured that the attacker cannot cast a vote in conformity with the system without having control over the voters concerned.*

Authentication of the voter to the voting server (beyond what is written on the code sheet itself) appears to largely unaddressed by the documentation (see, e.g., protocol specification §13.1). It was unclear whether a voter logs into and authenticates to the voting server (e.g., username, password.) Would attacking the mail system and stealing the voter code sheet package provide sufficient information to cast a vote on a voter's behalf? I did not see anything in the documentation seemingly addressing this possibility.

**SP Response**. "The documentation does not directly address this point since authentication differs from one canton to another." SP highlights two scenarios:

1. Cantons integrating e-Voting into an existing e-Government portal. Authentication elements include: username, password, and a second factor (e.g., authentication app).

2. Cantons running e-Voting from a website (e.g., `evoting.ch`). Authentication would require "additional information such as date of birth."

**My Response**. Swiss Post's response ultimately does not address Section 2.8 of the OEV. Although SP's response suggests that, strictly speaking, the voter code sheet is

not sufficient to cast a vote on a voter's behalf, perhaps a better question is to ask how feasible it is for someone *already* with another voter's code sheet to gather the remaining credentials.

Dates of birth, for example, are of dubious value for authentication purposes. They (a) cannot be changed in the event of a breach, (b) are low entropy even under ideal circumstances, (c) are not often well-protected from public view[4] and are well known to many other individuals in a voter's immediate social circle. Plausible threat actors in this setting include friends, family and even local community leaders engaging in credential *theft* or *gifting* such we have observed in Canada's e-voting experiences [4].

The OEV itself may need additional language. Currently, authentication is posed in terms of an attacker's ability to maliciously access a voter's account without "having control over" the voter. The notion of control needs additional defining language.

A relevant question seemingly not addressed by the OEV is how easily credentials can be transferred to another individual. For example, I spoke to several candidates in the Ontario Municipal election who described instances of what could be described as vote *gifting* whereby a voter gave a friend of neighbor the mail envelope containing their login PIN. A voluntary act obviously transcends the notion of control, yet intuitively it seems like a weak approach to authentication that such an "attack" could be so casually accomplished.

Consider two e-voting systems: System A requires a simple username and password. System B additionally requires a mobile device to take a 3D picture of the voter using technology to detect presentation attacks.[5] In System A, physical entity and digital identity are trivially separable and a voter's credentials can be easily transferred to arbitrarily many remote parties. The credentials of System B, however, is non-trivially connected to the voter's live, physical presence.

## 8 Proof soundness

Section 2.11 of the OEV addresses requirements for the soundness of the cryptographic proofs.

> *2.11.1: The probability of the attacker being able to falsify a proof under Number 2.5 if he changes a partial vote, suppresses a partial vote or casts a vote in someone else's name must not exceed 0.1%.*

Falsifying a proof of individual verifiability in the Swiss Post system would require the attacker to provide a valid choice return code of another selection (candidate). The codes are 4-digits. Assuming that (a) the codes are uniformly and randomly generated, (b) the attacker is not able to directly decrypt a return code (i.e., is relegated to guessing a valid code), (c) the attacker was not able to obtain the code sheet from the print office or by attacking the mail system responsible delivering it to the voter, and (d) assuming that

[4] Even Swiss Post's website discloses their CEO's birth year (1971).
[5] see e.g., ISO 30107-3 https://www.iso.org/standard/79520.html

voter would reliably identify an incorrect choice return code and take appropriate action (contrary to research [13]), then yes, the probability of successfully falsifying the choice return code in the Swiss Post system would not exceed $1/(10)^4 = 1/10,000$ or 0.1%.

> *2.11.2: The probability of the attacker being able to falsify a proof under Number 2.6 if he causes the calculated result to deviate by 0.1% from the correct result by altering and suppressing votes cast in conformity with the system or by entering votes not cast in conformity with the system may not exceed 1% per proposal, list election or candidate election.*

The probability that a proof of universal verifiability could be falsified in the Swiss Post system exceeding the defined threshold is based on computational hardness assumptions, e.g., not being able to compute the discrete logarithm of one generator relative to another in the Pedersen commitment scheme, as well as the general soundness of the zero-knowledge proofs. To the latter point, particular attention must be paid to testing/ensuring all function inputs have the expected algebraic form/set membership. See comments pertaining to the primitive specification made in §9 below.

> *2.11.3 If the probability of the attacker being able to falsify a proof under Number 2.6 is not negligible in the cryptographic sense, it must be possible to reduce the probability of success as desired by repeated tallying, by providing the auditors with an additional, independent proof under Number 2.6 for each count.*

This provision appears to not appear to apply to the Swiss Post system.

## 9 Definitions and Descriptions

Section 2.13 of the OEV addresses requirements for definitions and descriptions of the cryptographic protocol.

> *2.13.1: Wherever possible, building-blocks are used that are in widespread use worldwide and have been thoroughly scrutinised by experts. Standards, reference projects and scientific publications can be used as a benchmark. Derogations and cases of doubt must be dealt with separately in the context of the risk assessment referred to in Article 4.*

> *2.13.2: Instructions must not be underspecified. Individual instructions must restrict the options for implementation to such a degree that any form of implementation that the instructions allow is also compliant with meeting the cryptographic protocol requirements.*

### 9.1 System Specification

The following remarks pertain to the system specification document in relation to OEV requirements 2.13.1 and 2.13.2:

- The Preliminaries section (§3.1-3.3) of the system spec seems to repeat much of §2 of the primitives specification document and could likely be merged.
  - **SP Response**: This redundancy has been eliminated and now points to the primitives specification document for relevant information.
- §3.4.1: What is the purpose of the distinction between the *verification card* and the *voting card*?
  - **SP Response**: "We have this distinction purely for historical reasons. The voting card ID is printed on the code sheet, while the verification card ID is internal to the voting system."
- §3.4.3 (Voting Options): Please clarify that the product of prime numbers is expected to *not* have experienced a modular reduction, and therefore factoring a value $\alpha$ does not involve attempting to factor $\alpha + iq$ for values of $i > 0$.
  - **SP Response**: The verifier specification (not available at the time of the preliminary report) specifically checks for this.
- Algorithms 3.11 and 3.12 do not seem to enforce an upper bound on products of primes, and it is not clear from Tables 11 and 12 how many candidates can be accommodated by the default parameters.
- §3.4.4: It is not clear how the CorrectnessID's prevent an invalid plaintext from being encrypted. This section would benefit from additional background/explanation.
- Algorithm 4.4, Line 8: The HashAndSquare function does not appear to perform a modulo reduction, meaning that it implicitly relies on the RecursiveHash hash function to output a hash value $h$ such that $|h| < |\sqrt{(p)}|$, otherwise it is possible for HashAndSquare to output a value larger than $p$, and therefore not in $\mathbb{G}_q$. This should not be left as implicit, and I recommend including an explicit modulo reduction on the squared value so the reader does not have to go through a similar crypto self-diagnostic.
  - **SP Response**: Explicit modulo reduction was added in 0.9.8 of the primitives specification.
- Algorithm 8.9: Why is PBKDF parameterized to 32,000 iterations? Cite relevant standard.

– Algorithm 8.10: AES-GCM is initialized with a 96-bit IV, which is smaller than the block and key length (i.e., 128-bits). Please explain why a smaller IV was selected and why it is acceptable (e.g., recommend citing [9], which explicitly addresses this).

### 9.2 Primitives Specification

The following remarks pertain to the cryptographic primitives specification document in relation to OEV requirements 2.13.1 and 2.13.2

**Symbols.** The symbols and notation (Symbols section of the primitives specification) are mostly standard and would be readable by someone with a cryptography background. A few points of clarification are recommended:

– $\mathbb{G}_q$ is used to denote the set of quadratic residues modulo $p$, which forms a group of order $q$, which must be a sufficiently large prime in the Swiss Post system. This requirement should be explicitly stated.

– $\mathbb{N}^*$ is used to denote the set of positive integers. This notation is somewhat overloaded and could lead to confusion in at least two unintended ways:

  • The notation $\mathcal{S}^*$ in the context of a set of strings $\mathcal{S}$ is used to denote the *Kleene star*.

  • The notation $\mathbb{S}^*$ in the context of a set of integers $\mathbb{S}$ is used to denote a cyclic multiplicative group.

  A more standard notation (at least in my experience) would be $\mathbb{Z}^+$.

– $p$ is used to denote the modulus. In this application, it is required that $p$ is a large safe prime. This requirement should be explicitly stated.

– $q$ is used to denote the non-trivial subgroup of $\mathbb{Z}_p^*$. In this application, it is required that $q$ is a large prime. This should be explicitly stated.

– Throughout the document, lowercase $l$ is used (see e.g., Algorithm 3.3), however for visual clarity, the explicit $\ell$ (i.e., `\ell`) is preferred.

**Basic Data Types.** The basic data types (Section 2 of the primitives specification) include standard bit/byte/integer/string type conversions, encodings (base16, base32, base64). The approaches of Algorithms 2.1–2.8 all appear standard and correct (with the following exceptions):

– Algorithms 2.1, 2.3, and 2.5 output a string $S$, respectively in: $\mathbb{A}_{Base16}$, $\mathbb{A}_{Base32}$, $\mathbb{A}_{Base64}$. Algorithms 2.2, 2.4, and 2.6 input a string $S$, respectively in: $\mathbb{A}_{Base16}$, $\mathbb{A}_{Base32}$, $\mathbb{A}_{Base64}$. Each such set $\mathbb{A}_x$ respectively denotes the string's alphabet, implying the string is a single character in length. An arbitrary string over this alphabet should be indicated with the Kleene star notation $(\mathbb{A}_x)^*$.

– The application of the Kleene star should be in superscript of the set, i.e., $\mathbb{A}^*_{UCS}$, or ideally $(\mathbb{A}_{UCS})^*$, not $\mathbb{A}_{UCS}*$ (see e.g., Algorithm 2.10).

– The *verum* $\top$ and *falsum* $\bot$ truth symbols used throughout (e.g., Algorithms 2.4, 2.6, etc.) should be explicitly defined in the Symbols section.

– Algorithms 2.4, 2.6 seem to draw a distinction between "**return**" and "**output**," which should be defined.

– The output set of Algorithms 2.4 and 2.6 is listed as a byte array $\mathcal{B}^*$, but technically the function's codomain includes the *falsum* symbol, i.e., $\mathcal{B}^* \cup \{\bot\}$

– Apostrophes as a decimal separator for large numbers will likely confuse to North American and many European readers. For example, *one million* is represented in the documentation as $1'000'000$. This notation should be explicitly defined as many European readers would expect the notation 1.000.000 whereas North American readers would expect $1,000,000$.

– The range notation used throughout, i.e., $i \in [0, n)$ is relatively commonplace, but it is recommend that the inclusion/exclusion be made explicit, i.e., that $0 \leq i < n$.

  • **SP Response**: Addressed in §1.1 of updated primitives specification.

– Algorithm 2.8 calculates byte length as $\lceil \frac{|x|}{8} \rceil$, whereas Algorithm 3.1 invokes the redundant `bytelength()` functionality.

**Basic Algorithms** The basic algorithms (Section 3 of the primitives specification) include algorithms for generating random integers, vectors, and strings given a non-deterministic `randomBytes()`. These approaches all appear standard and correct. I did not observe anything in Algorithms 3.1–3.5 that would to lead biases in the output (cf. e.g., [6]). However in terms of specification, I make the following recommendations.

– The input and output sets, as well as the uniformity of the output distribution of the `randomBytes` function, should be explicitly stated.

– The `randomBytes` capitalization should be `RandomBytes` to be consistent with the other functions.

– The ToInteger in Algorithm 3.1 should be ByteArrayToInteger.

– Algorithms 3.1 and 3.2 use the terms "upper bound" and "upperbound" respectively.

– Algorithm 3.1 should more precisely output a random *integer*, not just a number.

– The functionality of the Truncate function invoked in Algorithms 3.3-5 is reasonably obvious but should still be defined.

– Section 3.2 points out that when invoking the recursive hash, it is the "caller's responsibility to ensure only finite inputs are provided in practice." Is there such thing as an infinite input in practice?

– How is the "pseudo-random hash function" of the recursive hash (Algorithm 3.6) defined? Why not just assume the hash function is a random oracle (or computationally indistinguishable from one)?

– The recursive hash requires domain-separation, i.e., RecursiveHash('A', 'B') does not yield the same result as RecursiveHash('AB'). The approach internally seems to be essentially to compute RecursiveHash('A', 'B') essentially as Hash(Hash('A')||Hash('B')), which seems plausible, if non-standard. We know from other related applications (such as HMAC), however, that simple combinations of hashing could be insufficient at preventing certain attacks (such as length-extensions). Can we be certain Algorithm 3.6 explicitly prevents attacks like these? Is there analysis that can be invoked from related constructions (e.g., CHVote)?

– The recursive hash does not seem to enforce type separation, e.g., it appears as though RecursiveHash(255)=RecursiveHash(`0xff`). I do not see an immediate exploit, but it seems as though enforcing type separation (e.g., RecursiveHash(255)!=RecursiveHash(`0xff`)) would be a safer choice in the context of domain separation.

  - **SP Response**: Explicit type separation was added to RecursiveHash by prepending an unique *type* identifier byte to each pre-image. The defined types are: *byte, integer, string.*

– I find Algorithm 3.6 challenging to read. For example, $\mathcal{V}$ is a domain, but $\mathcal{V}^*$ is a set, but it is "recursively defined."

  - **SP Response**: Notation closely follows Algorithm 4.15 of the CHVote specification [10].

**ElGamal.** The ElGamal specification (Section 4 of the primitives specification) includes algorithms for generating domain parameters, key pairs, and verifiable encryption/decryptions. These approaches (e.g., parameter creation, multi-recipient encryption) all appear standard and correct. I did not observe anything in algorithms 4.1-4.11 that would lead to incorrect decryption or loss of privacy outside of the standard computational hardness assumptions. However, the algorithms specifications could be improved in the following ways:

– Section 4.1 says: "Since $p$ and $q$ are primes. . . ." However, as noted in the Notation paragraph above, the primality of these values was never explicitly stated.

– This section goes on to say: "the ElGamal encryption scheme specifies the default generator $g$ as the smallest element in $\mathbb{G}_q$." Did Tahir Elgamal actually specify this? Citation?

- **SP Response**: They are following precedent.[6]
- **My Response**: I have tried to trace the origins of this guidance. The earliest example I can find comes from the OpenSSL project, which discussed it in an email in 1995.[7] This email also, curiously, directs them to pick a generator of $\mathbb{Z}_p^*$, instead of $\mathbb{G}_q$. OpenSSL and numerous other projects follow this advice [8], leading to the least-significant bit of the exponent being revealed. Although the Swiss Post system *does not* do this, it does illustrate the risk of taking a TLS-centric use case and applying it to an advanced Decisional Diffie-Hellman (DDH) use case.

– Technically, there is no "smallest" element in a cyclic group in the same way there is no first point on a circle. Perhaps "the smallest integer x>1 such that x is a generator of $\mathbb{G}_q$."

– Table 5 uses the term "Strength," however I recommend using the term "security level" to be more in line with the cited NIST and ECRYPT standards (citations [2], [1], [21] in the primitives specification).

– The verifiable generation of safe prime $p$ in Algorithm 4.1 seems sufficient to avoid the trapdoored attacks suggested by Teague et al. [11].

– There are sufficiently many ways to approach primality testing that the algorithm IsProbablePrime is likely underspecified relative to OEV §2.13.2.

– In the Algorithm 4.1 context, security level $\lambda$ relates to the security level of $|p|$ and $|q|$. However, in line 8, it appears to be the number of rounds in the probable prime test, which is an independent notion.

– Algorithm 4.1 lines 10–14 pick the "smallest" generator of $\mathbb{G}_q$. What is the justification for picking the smallest generator instead of, say, the smallest *guaranteed* generator (i.e., smallest square)? In other words, if $g \in \{2, 3, 4\}$, why not just always output $g = 4$? Are lines 10–14 necessary? Suppose there was a security benefit to *not* always picking $g = 4$. If such a benefit existed, it could always be trivially circumvented. Because the generator space is so small, the election agency would only need to try one or two trivial variations of the election name to cause Algorithm 4.1 to output $g = 4$.

**Subgroup Structure of $q$.** There appear to be no requirements for subgroup order $q$ other than being prime and of size $|p| - 1$. This requirement is widely accepted and in line with the approach used by NIST to verifiably generate DSA group parameters. However, I would like to briefly pose the question of whether $q$ itself should be a safe

[6] See e.g., https://datatracker.ietf.org/doc/html/rfc3526

[7] https://github.com/openssl/openssl/blob/e0fc7961c4fbd27577fb519d9aea2dc788742715/crypto/dh/generate

prime by considering whether the subgroup structure of $q$ could ever be exploited for unintended purposes.

For example, consider primes $p, q, r$ in which $p = 2q + 1$ and $q = 2rs + 1$ and $r$ is small. There exists a subgroup $\mathbb{G}_q \subset \mathbb{Z}_p^*$, but there also exists a subgroup $\mathbb{G}_r$ of the exponent group $\mathbb{Z}_q^*$.

Suppose a player in the protocol selects an $x \in \mathbb{G}_r$ and computes $y = g^x$. Here $x$ can be recovered by finding an $\hat{x}$ such that $(y)^{\hat{x}} = g$, i.e., an $\hat{x} \in \mathbb{G}_r$ such that $x\hat{x} \equiv 1 \bmod q$. This can be found in $O(\sqrt{r})$ exponentiations using, e.g., a baby-step/giant-step approach.

Successive exponentiations would still allow the retrieval of their product from the exponent group in $O(\sqrt{r})$ operations. Let $x_1, x_2, \ldots, x_k \in \mathbb{G}_r$ and $y = ((((g)^{x_1})^{x_2}) \ldots)^{x_k}$. The product $x_1 \cdot x_2 \ldots \cdot x_k \bmod q$ would still have small order $r$ and be recoverable from $y$ in $O(\sqrt{r})$ exponentiations.

Since many of the zero-knowledge proofs combine exponentiation operations with multiplication operations in $\mathbb{Z}_p^*$ (resp. multiplications with additions in the exponent group $\mathbb{Z}_q^*$), it seems unlikely that this small exponent subgroup structure could be preserved or recovered toward realizing most attack goals (e.g., to break soundness). However, it may be worth considering the possibility of small subgroups of the exponent group being used by colluding players to implement a subliminal/covert channel in the election data set.

**SP Response.** Swiss Post reiterates that their computational assumptions reduce to the DDH assumption, and therefore they do not see how small subgroups of $q$ could be exploited, including by number-field sieves (NFS).

**My Response.** DDH assumes exponents are chosen independently and uniformly from $\mathbb{Z}_q$. I am ultimately asking about what happens if a voter (or voters) intentionally violate DDH assumptions by picking exponents *not independently* and/or *not uniformly*, for example, by selecting from some small $\mathbb{G}_r \subset \mathbb{Z}_q^*$. At first glance, this may seem trivial—a voter could always pick a trivial exponent (e.g., 1 or 2) and expose their plaintext to brute-force search. However, smaller subgroups of the exponent have a modestly more interesting property: The product of arbitrarily many successive exponentiations of exponents drawn from $\mathbb{G}_r$ can be recovered in $O(\sqrt{r}) \ll O(\sqrt{q})$, which is *not* the case for arbitrarily many exponentiations of small exponents drawn from $\mathbb{Z}_q$.

I do not see an immediate way to exploit this in the Swiss Post system beyond using it as a subliminal channel (which has not been addressed in the Swiss Post response). Ultimately, Swiss Post's appeal to DDH is necessary—privacy cannot be assured with out it. The main goal of raising this question is simply to point to the existence of non-trivial and evidently unnoticed algebraic structures and ask what could be the consequences if DDH assumptions are maliciously deviated from.

**ElGamal Parameter Selection.** The arbitrary *testing-only* level should be eliminated as it admits trivially insecure parameters (e.g. $p = 5, q = 2$, |p|=3, |q|=1, which is the 1-bit security level). Purposefully weak parameters seemingly exist to offer efficient testing, but it comes at the arguably unacceptable risk of accidental use in a production capacity.

As a general principle, cryptographic software should be robust in the presence of weak parameters, i.e., not designed purposeful designed to admit weak parameters unless the user remembers to use the correct security level.

Table 1 shows the anticipated approximate security levels of safe primes of various lengths. Swiss Post's *default* security level is 112 bits and the *extended* security level is 128 bits (see Table 5, primitives specification).

| | | Allowed | | |
|---|---|---|---|---|
| $\|p\|$ | Security level | Swiss Post | ECRYPT | NIST |
| 3 | 1 bit | *testing-only* | No | No |
| 829 | Broken (2020) | *testing-only* | No | No |
| 1024 | 80 bits | *testing-only* | No | No |
| 2048 | 112 bits | *default* | No | No after 2029 |
| 3076 | 128 bits | *extended* | Yes | Yes |

**Table 1.** Security level of the discrete logarithm problem (DLP) modulo safe primes $p$ relative to recommendations at various bit lengths $|p|$.

The characterization by Swiss Post of the 2048-bit prime $p$ as the "default" security level is somewhat misleading. ECRYPT recommends "new systems should use a *minimum* $p$ of 3072-bits" [14]. Similarly, NIST does not recommend using a 2048-bit $p$ for any data with a security lifetime past 2030, If the security life of Swiss Post election data extends past 2030, then, according to NIST, "protection at a security strength of 112 bits will not be sufficient" [1].

Furthermore, these standards only represent the *minimal* security levels, which may not provide a sufficient security margin for an application as prominent as an online public election.

Discrete logarithms in 795-bit safe prime groups have been accomplished in practice,[8] and with the recent factorization of RSA-250,[9] discrete logarithms in a 829-bit safe prime group are now feasible.

Finally, a higher level be selected for the extended security level. Ideally, the "extended level" would be pegged to the 256-bit level, implying $|p| = 15350$. This would incur significant performance reduction. However, the feasibility of this recommendation should be assessed in the setting of an election's one-time cost and not in the conventional network security setting in which a server performs these computations on a continual basis.

**Recommendations:**

- The arbitrary *testing-only* security level be disallowed.
- The *default* 112-bit security level be disallowed.

[8] https://caramba.loria.fr/dlp240-rsa240.txt
[9] https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2020-February/001166.html

– The *extended* 128-bit level be made the default level.

– The 256-bit level be made the extended level.

– Articulate a minimum security horizon (in years) for election data to be non-decryptable.

**SP Response A.** They are working on code optimizations to speed up performance for $|p| = 3072$-bits.
**My Response.** If performance is a higher priority than meeting the security standards they themselves cite, why work in finite fields instead of elliptic curves?
**SP Response B.** The *testing-only* level was revised to $|p| = 8n$ for postive integers $n > 0$. The *default* and *extended* levels are unchanged.
**My Response.** Trivially weak parameters as small as $p = 167$, $q = 83$ are still valid in the *testing-only* level and it seems the risk of accidental (or intentional) deployment outweighs SP's unexplained need for optimally-fast (and optimally insecure) testing parameters.

**ElGamal Operations.** This section pertains to Sections 4.2-4.11 of the primitives specification.

– Throughout the primitives specification, ElGamal secret keys and random factors are specified as $r, sk \in \mathbb{Z}_q$, whereas the corresponding public key (or group element) is specified as $pk \in \mathbb{G}_q$. Although $0 \in \mathbb{Z}_q$, and $y = g^0 \equiv 1 \bmod p$, which is an element of $\mathbb{G}_q$, it is not a *generator* of $\mathbb{G}_q$.

– This mismatch carries over to the primitive specification GenRandomInteger() (see Algorithm 3.1), which includes 0 in the codomain. If GenRandomInteger() outputs 0, or if a malicious player simply selected 0 during key generation (or encryption), Algorithm 4.2 would output an invalid public key $g^0 \equiv 1 \bmod p$, which is not a generator of $\mathbb{G}_q$.

- **SP Response**: "Whether we shall exclude 0 as a secret key and, conversely, 1 as a public key was discussed with multiple experts in the past." Junod argues[10] that 0 is a valid exponent of a generator of $\mathbb{G}_q$ but goes on to make an inductive argument along the lines of "if you remove 0, then why not 1, then why 2, then why not...?"

- **My Response**: I accept the base case when a single exponentiation of $g$ is performed, but I fail to see the inductive step of this argument:

  1. $(((g^{x_1})^{x_2})...)^{x_k}$ results in 1 if any $x_i$ is 0.

  2. If the protocol ever applied successive exponentiations of a generator of $\mathbb{G}_q$, and if any exponent $x_i = 0$, it would erase the contribution of the other exponents $x_{j \neq i}$.

[10] https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives/-/issues/7

3. This is not true of exponents $x > 0$.

Algorithm 5.6, for example, implicitly acknowledges this fact by ensuring elements of Pedersen commitment keys are generators of $\mathbb{G}_q$, i.e., in $\mathbb{G}_q \backslash \{1\}$.

– Algorithm 4.6 is explained as an algorithm that "exponentiates the ciphertext." A more descriptive title would be "exponentiates each ciphertext element by an exponent $a$."

– Explain/define the $\bar{*}$ symbology in Algorithm 4.11.

**Mixnet.** Due to time, I was unable to closely examine the Bayer-Groth specification in Section 5 of the primitives specification. I do, however, have a few brief comments:

– What is the design rationale for using mixnets as opposed to, say, homomorphic tallying? My sense is that the voting systems of the respective cantons can not be implemented using the latter approach, however, this appears not to have been addressed in the documentation. Because the cryptography of the Swiss Post system is so involved, a more explicit case should be made for its necessity.

- **SP Response**: "Switzerland has complex electoral models. Every canton has different election laws, and an election event can comprise elections and referendums on the Federal, Cantonal, and municipal levels - with different voters having different eligibility rights. Consequently, the counting process is often non-trivial ... Therefore, verifiable mixnets are a much more appropriate design decision than homomorphic tallying in our specific use case. We will add this clarification in the next version of the crypto-primitives."

– Why was the Bayer-Groth mixnet chosen in particular? I saw an efficiency argument made in the documentation. However, this approach adds a lot of complexity, making the primitives specification much more intricate (see, e.g., Algorithms 5.18-5.26).

– Section 5 claims "verifiable mixnets underpin most modern e-voting schemes." Is there a citation to a systematic literature review that supports this?

– Algorithms 5.1 says, "The public key is passed as context to all sub-arguments." Should this not be sub-functionalities?

- **SP Response**: The term "sub-arguments" is the correct. The Bayer-Groth proof has a hierarchical structure of arguments.

- **My Response**: The source of my confusion appears to come from the overloading of the term "argument." Here, "argument" is being used in the context of a *proof*, not as a *parameter*. I reexamined the document and confirm they are using it consistently.

– Algorithm 5.4 line 5: The subscript of $\pi_{i+offset}$ should be taken modulo $N$.

- **SP Response**: Since $i \in [0, N)$ and $offset \in [0, N - i)$, then $i + offset < N$ and a reduction is unnecessary.

– The verifiable generation of domain parameters (i.e., generators $g, h$) of the Pedersen commitments in Algorithms 5.6 appears to be well structured and prevents finding relationships between $g, h$.

**Zero-Knowledge Proof Systems.** The zero-knowledge proof systems in Section 6 follow the approach of Maurer, which is a slight adaptation and generalization of more well-known proof systems (e.g., Schnoor, Chaum-Pedersen). In particular, the multi-recipient nature of the Swiss Post ciphertext creates an additional conceptual barrier when trying to draw a mental line between a single-recipient proof system as used in other e-voting solutions (e.g., Chaum-Pederson as used in Helios/ElectionGuard).

I examined Algorithms 6.2-6.9 and checked for the following:

– Insufficient context passed into the Fiat-Shamir challenge, which could lead to attacks such as those proposed by Bernhard et al. [2] which exploit degrees of freedom as a result of insufficient dependence on pre-challenge values. However, I did not observe any instances where the recursive hash had insufficient context.

– Working with invalid inputs could lead to overriding a proof's soundness in instances where the verifier function accepted a spurious proof, such as was done in Helios [6]. However, I was not able to find any instances where soundness could be broken, assuming all inputs are checked for the anticipated structure (e.g., group membership). However, Line 1 of Algorithms 6.2, 6.5, and 6.8 invoke GenRandomInteger and GenRandomVector functions, which are capable of returning 0, which may lead to invalid inputs to subsequent functionalties.

If input values are not thoroughly checked, it may be possible for the proof verifying algorithms to accept trivial inputs. For example, suppose the decryption proof in Algorithm 6.2 were to allow the key pair $pk = 1$, $sk = 0$ as input. This is partially allowable since the algorithm explicitly states $sk \in \mathbb{Z}_q$, which includes the $sk = 0$ case, and $1 \in \mathbb{G}_q$, even though 1 is not a generator thereof.

Without loss of generality, assume the single-recipient ciphertext case. In the first line, the prover could select $b = 0$, and compute response $z \leftarrow b + e \cdot sk$. This would yield $z \equiv 0 \bmod q$ independently of all Fiat-Shamir challenges (i.e., recursive hash values) $e$ rendering the contribution of the Fiat-Shamir challenge effectively useless in the response value $z$.

Nevertheless, the point remains that the group identity element (i.e., 1 in this setting) should not be used as a base in an exponentiation operation.

Fortunately due to this proof's particular design, this does not create an immediate or obvious risk to the proof's soundess; the verifier tests for equality between the prover's claimed hash $e$ and the verifier's re-generated hash $e'$. Therefore cancelling out

the contribution of the hash $e$ from response $z$ is not, on its own, sufficient to break soundness.

Alternatively, if the algorithm fails to guarantee that an input $\beta$ is a generator of $\mathbb{G}_q$, a malicious prover could attempt to use a generator of $\mathbb{Z}_p^*$ instead. Assuming the Fiat-Shamir challenge $c$ was even, then $\beta^c \in \mathbb{G}_q$, and the proof would accept the invalid input. One application might be to create an accepting proof of an invalid an ElGamal ciphertext $\langle \alpha \in \mathbb{G}_q, \beta \notin \mathbb{G}_q \rangle$ as was done, for example, to Helios [6].

- **SP Response**: Swiss Post expressed some confusion over this section (issues #133, #134, #135).
- **My Response**: My point was simply that a group's identity element is *not* a generator and should not be used as a base in an sequence of exponentiation operations. However, I do not see any evidence this is happening in the proofs, so the point can be disregarded.

**Additional comments:**

- Algorithms in Section 6 variously use $\phi$ as a function, and a value.
- Algorithm 6.7 Line 7 uses the notation $\mathbf{z} \leftarrow \mathbf{b} + e \cdot (r, r')$, which seems to mean $z = (b_1 + e \cdot r, b_2 + e \cdot r')$, but should be clarified.
- Algorithm 6.9 Line 6: The variable $e$ should be $e'$. $e$ is the asserted hash of the prover, whereas 6.9 is computing $e'$, which is the has as computed by the verifier.

**Recommendations:**
The documentation should be modified to make it explicitly clear that ensuring an Algorithm's input has the correct form is required. This could be done, for example, by using the "**Ensure**" keyword, such as is found in Algorithm 5.4, or the "assert" keyword as is found in many programming languages, as long is it is made clear to the reader that the algorithm explicitly fails and returns $\perp$ if any assertion fails. Specifically:

- All instances of $y \in \mathbb{Z}_q$ in the documentation should be reexamined to see if an element of $\mathbb{Z}_q^*$ is required instead (i.e., if 0 should be excluded).
- All instances of $x \in \mathbb{G}_q$ in the documentation should be reexamined to see whether a *generator* of $\mathbb{G}_q$ is required instead (i.e., if 1 should be excluded).
- For all inputs requiring $x$ to be a generator of $\mathbb{G}_q$, assert:
  - $2 \leq x < p$
  - $x^q \equiv 1 \bmod p$
- For all inputs requiring $y$ to be in $\mathbb{Z}_q^*$, assert:

- $1 \leq y < q$

– All invocations of GenRandomInteger(q) requiring the output to be in $\mathbb{Z}_q^*$ should disallow 0 as output.

– **SP Response A**: We want to avoid repeating group membership tests in every algorithm.

– **My Response**: This is perfectly understandable for efficiency reasons, as long it is unambiguously and prominently communicated to the reader how critical it is to *only ever* work with verified parameters. If an Algorithm does not explicitly check, it must make a trust assumption about the input. This trust assumption should be clearly communicated so future developers cannot unknowingly admit invalid cryptographic parameters. This may seem like a trivial point, but my experience suggests otherwise. The industry has systematically overlooked parameter verification in discrete logarithm implementations including in OpenSSL, GPG, Helios, TLS 1.2 (cf. [8]) and even in the previous Scytl/Swiss Post implementation [11].

– **SP Response B**: Calculating the Jacobi symbol is usually more efficient than checking that $x^q = 1 \bmod p$.

– **My response**: Computing a Legendre, Jacobi, or Kronecker symbol would be more efficient but is only correct in the *specific case* of safe primes.

**A Brief Word about Quantum Computing.** Recognizing that I only have access to public information on the subject, my view is that an implementation of Shor's algorithm reaching quantum supremacy, i.e., capable of, or comparable to computing discrete logarithms at $|p| > 829$-bits is highly unlikely in the next 10 years, and, in my estimation of the balance of probabilities, is unlikely to be seen in our lifetimes.

Furthermore, I believe evidence of major advances in this area will become apparent in the public sphere with enough horizon to cease use of the system before it could be deployed against an active election. Of course, the threat to historical election data would remain.

Nevertheless, the threat of quantum computing is only addressed at a glancing level in the system specification: "quantum computer could break some of the mathematical assumptions underpinning the Swiss Post Voting System."

**Recommendations:**

– Provide a detailed description of which mathematical assumptions are broken by the existence of a sufficiently large quantum computer.

– Describe which specific exploits could be made against the election if (a) the quantum computer could execute the attack against an active election and (b) could be executed against historical election data some years after.

**SP Response:** They accept the recommendation and will consider adding discussion to future versions of the documentation.

### 9.3 Verifier Specification

This section pertains to the verifier specification document. Although this document directly builds on the other documentation, I had difficulty forming a complete picture of the verification. Much of the information (and intuition) is hidden inside recursive vectors and algorithms making it difficult to reason about the sufficiency of the set of checks.

Overall, the specification is well annotated but not very self-contained. The verification algorithms (e.g., Verification 1.01) and supporting algorithms (e.g., Algorithm 3.1) are ultimately top-level checks that unpack the input variables, calls one or more subroutines, and outputs the conjunction of returned truth values resembling, in essence, polynomial-time reductions to a series of verification oracles.

- §1.2 of the verifier specification discusses the *integrity verification* category, which includes ensuring data elements "correspond to the specification" and are "all within the specified ranges." I interpret this to more broadly mean that data elements have the correct *form*, i.e., not just the correct range, but correct type, anticipated membership, etc.). This section does not really address the question of integrity verification and the checks are not explicit or exhaustive (e.g., ensuring all elements have the correct group membership).

- Throughout the verifier specification (and other documents) there is an improper use of quotation marks. All instance of a quotations of the form "quote" should be corrected to "quote" (see e.g., Algorithm 3.1, line 5). This appears to be a LaTeX representation issue. Instead of `''quote''`, use ``` ``quote'' ```.

- §4.3 introduces the notion of secure logs. The explanation relies almost entirely on a citation to Scytl's prior work. What are the security goals of this approach? What is the argument/proof?

- What is the purpose of a *checkpoint* log entry?

- What is the purpose of the encrypted key `ESK` and how is decryption verified?

- The HMAC and digital signature algorithms are fully specified in §4.3, but the encryption algorithm is not.

- The indefinite article of "HMAC" should be *an* HMAC, not *a* HMAC.

- §4.3 contains the following text: "`HMAC`: a HMAC of the log entry." Throughout this section, HMAC is being used as both a verb and an overloaded noun: *to HMAC* (as in *to invoke* an HMAC function on an input), *HMAC* (as in the *function* itself), and *HMAC* (as in the *output* of the HMAC function).

– HMAC is to a MAC as AES is to symmetric-key encryption. Recommend putting the HMAC calls inside of explicit MAC_sign and MAC_verify functions and referring to the output of MAC_sign as a MAC *tag*. In this way, one can talk about *verifying* the *tag* produced by MAC_sign, instead of HMACing the HMAC created by the HMAC.

– What key is used to create the checkpoint MAC tags?

– Since multiple keyed primitives are used in §4.3, recommend making every instance of "key" explicit: e.g., symmetric encryption key (i.e., the one used to create the `ESK`, the MAC key, the public signature verification key, the private signing key, etc.).

– Algorithms 4.5–4.7 list the MAC key as being a $k \in \mathcal{B}^{32}$ implying a 256-bit security level. This is mismatched with the specified *default* and *extended* security levels.

– There should be a mapping from the variables on Page 31 (`LSK, ESK, PHMAX`, etc.) to the inputs to Algorithms 4.5–4.7.

– Algorithm 4.7 suggests the modulus of a public RSA signature verification key is a value $m \in \mathbb{P}$. Rather, the modulus is an integer $n \in \mathbb{P} \times \mathbb{P}$. This fact, however, cannot be checked by the verifier.

## 10 Compliance with protocol requirements

Section 2.14 of the OEV addresses requirements for mathematical proofs establishing the protocol's compliance with the OEV's other requirements.

> *2.14.1 One symbolic and one cryptographic proof must demonstrate that the cryptographic protocol meets the requirements in Numbers 2.1-2.12.*

Due to time, I did not examine the proofs of compliance made in the protocol specification.

> *2.14.2: The proofs must directly refer to the protocol description that forms the basis for system development.*

Due to time, I did not examine the proofs of compliance made in the protocol specification.

> *2.14.3: The proofs relating to basic cryptographic components may be provided according to generally accepted security assumptions (e.g. 'random oracle model', 'decisional Diffie-Hellman assumption', 'Fiat-Shamir heuristic')*

The cryptographic proofs in the primitives specification document (especially the Zero-Knowledge proofs in §6) and the cryptographic building blocks in the protocol specification (§3-9) are consistent with generally accepted security assumptions and should generally be comprehensible and recognizable to an individual with a background in cryptography.

## References


1. E. Barker and A. Roginsky. Transitioning the use of cryptographic algorithms and key lengths. *NIST Special Publication 800-131A Rev. 2*, 2019.
2. D. Bernhard, O. Pereira, and B. Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 626–643. Springer, 2012.
3. R. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P. S. Herrnson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, et al. Scantegrity ii municipal election at takoma park: The first e2e binding governmental election with ballot privacy. 2010.
4. A. Cardillo, N. Akinyokun, and A. Essex. Online voting in ontario municipal elections: A conflict of legal principles and technology? In *International Joint Conference on Electronic Voting*, pages 67–82. Springer, 2019.
5. A. Cardillo and A. Essex. The threat of ssl/tls stripping to online voting. In *International Joint Conference on Electronic Voting*, pages 35–50. Springer, 2018.
6. N. Chang-Fong and A. Essex. The cloudier side of cryptographic end-to-end verifiable voting: a security analysis of helios. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 324–335, 2016.
7. C. Culnane, M. Eldridge, A. Essex, and V. Teague. Trust implications of ddos protection in online elections. In *International Joint Conference on Electronic Voting*, pages 127–145. Springer, 2017.
8. K. Dorey, N. Chang-Fong, and A. Essex. Indiscreet logs: Diffie-hellman backdoors in tls. In *Network and Distributed System Security Symposium (NDSS)*, 2017.
9. M. Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. *NIST Special Publication 800-38-D*, 2007.
10. R. Haenni, R. Koenig, P. Locher, and E. Dubuis. Chvote protocol specification, version 3.2. In *IACR e-print archive*, 2020.
11. 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 (SP)*, pages 644–660, 2020.
12. J. A. Halderman and V. Teague. The new south wales ivote system: Security failures and verification flaws in a live online election. In *International conference on e-voting and identity*, pages 35–53. Springer, 2015.
13. E. Moher, J. Clark, and A. Essex. Diffusion of voter responsibility: Potential failings in e2e voter receipt checking. *USENIX Journal of Election Technology and Systems JETS*, 1:1–17, 2014.
14. N. Smart et al. Algorithms, key size and protocols report (d5.4). *ECRYPT-CSA H2020-ICT-2014 – Project 645421*, 2018.

## A Restrictions and Limitations

Except for the issues raised by Teague et al. [11] in 2019, I had no prior familiarity with the Swiss Post system. The totality of documentation falling under Scope 1 (Cryptography) was extensive. Due to limited time available to conduct the analysis, therefore, I could not fully examine all aspects of the system (as noted throughout).

## B Author Bio

**Aleksander Essex** is an associate professor of software engineering at Western University in Canada and associate chair (graduate) of the Department of Electrical and Computer Engineering. He holds a Ph.D. in Computer Science from the University of Waterloo (2012).

His research specializes in cybersecurity and applied cryptography with a focus on election technology. He has numerous publications on e-voting security and cryptographic end-to-end election verification (E2E-V). He was one of the principal members of the Scantegrity project, which ran the first fully E2E-V election in Takoma Park, Maryland, in 2009-11. His cybersecurity study of e-voting in the 2018 Ontario Municipal Election won the best paper award at E-Vote-ID in 2019.