Rolling Re-Evaluation of the Swiss Post e-Voting System: Versions 1.4.0 and 1.4.3

Audit Scope 1: Cryptographic Protocol

Aleksander Essex

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

July 16, 2024

Submitted to the Swiss Federal Chancellery

Management Summary

In cooperation with the Chancellery, I re-evaluated the Swiss Post e-voting system, covering changes made between versions 1.3.0 and 1.4.0. Of 12 new issues identified in my 2023 report, 9 have been resolved, and 3 remain unresolved. Four new issues were identified as of version 1.4.0. Additionally, I reviewed the Cryptographic Primitives Specification changes from document versions 1.4.0 to 1.4.1 and found no issues introduced in this update.

As in previous reports, many of these findings pertain to minor issues in the specification. However, we make the following high-level recommendations:

1. **Redundant computation**. Eliminate redundant computations that do not contribute to the system's overall security guarantee (Issue EX-267).

2. **Update test vectors**. Review test vectors for correctness whenever changes are made to the specification (Issues EX-261, EX-262).

3. **Better support for performance optimizations**. Deviations from established cryptographic practices should be accompanied by a cost-benefit analysis, including (1) quantification of the performance improvement and (2) a substantive security analysis (Issues EX-268).

4. **Better support for security claims.** Security claims should be supported, to the greatest degree possible, with analysis against the threat model (Issues EX-265, EX-266).

5. **Preserving rationale for design decisions**. The reasoning for various design decisions should be documented for future readers to preserve institutional memory though time. See, e.g., Issue EX-266 and the (substantively resolved) issue about modulo bias (discussed below).

Version History

July 16, 2024	Minor clarifications in response to Chancellery feedback.
July 9, 2024	Management summary and minor typographical revisions.
July 7, 2024	Initial draft submitted to Chancellery.

Table of Contents

1 Documents Examined

Below is a list of the history of documents examined. For each document, the first column represents the version of my report. The second column lists the document version examined in my report. The third column lists the date the respective report was submitted to the Chancellery.

Primitives Specification

Description: Pseudocode specifications of cryptographic functions used by the Swiss Post system. Referred to throughout this document as the *primitives specification*.

Report	Version Examined	Date
2024 Rolling Re-Evaluation (this document)	1.4.1	2024-07-16 (submitted)
2024 Rolling Re-Evaluation (this document)	1.4.0	2024-07-16 (submitted)
2023 Rolling Re-Evaluation	1.3.0	2023-08-01
2023 Rolling Re-Evaluation	1.2.1	2023-08-01
2023 Addendum II	1.2.0	2022-12-09
2022 Re-Examination (Addendum I)	1.0.0	2022-06-24
2022 Re-Examination	1.0.0	2022-06-24
2021 Final Report	0.9.8	2021-10-15
2021 Preliminary Report	0.9.5	2021-06-22

Available: https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives/-/blob/master/Crypto-Primitives-Specification.pdf

System Specification

Description: Document describing the steps, phases and procedures of setting up, executing and verifying an election using the Swiss Post system. Referred to in this document as the *system specification*.

Report	Version Examined	Date
2024 Rolling Re-examination (this document)	1.4.0	2024-07-16 (submitted)
2023 Rolling Re-examination	1.3.0	2023-08-01
2022 Re-Examination	1.0.0	2022-06-24
2021 Final Report	0.9.7	2021-10-15
2021 Preliminary Report	0.9.6	2021-06-25

Available: https://gitlab.com/swisspost-evoting/documentation/-/blob/master/System/System_Specification.pdf

2 Examination of Changes in System Version 1.4.0

We discuss relevant findings in changes made to Swiss Post's e-Voting system in version 1.4.0. Specifically, we examine the changes made to the cryptographic primitives specification document as of version 1.4.0.[1] We additionally review changes made in the system specification document as of version 1.4.0.[2]

Findings. Of twelve new issues identified in my 2023 report,[3] 9 have been resolved and 3 remain unresolved. Since the 2023 report, the Chancellery instituted JIRA-based issue tracking. These three unresolved issues have now been mapped to the following JIRA issues:

- EX-265: Dates of birth as an authentication credential.
- EX-266: Explain how Argon2id memory profiles are applied.
- EX-268: Non-standard approach to verifiable parameter generation.

Four new issues were identified as of version 1.4.0:

- EX-261: Invalid group generator in several test vectors.
- EX-262: JSON parse error in test vector.
- EX-264: Undefined input for some algorithms in the testing-only security level.
- EX-267: Cryptographically redundant computation in Algorithm 8.1.

[1] Swiss Post Cryptographic Primitives specification version 1.4.0. Available: https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives/-/raw/crypto-primitives-1.4.0/Crypto-Primitives-Specification.pdf

[2] Swiss Post Cryptographic System Specification version 1.4.0. Available: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/raw/documentation-1.6.0.0/System/System_Specification.pdf

[3] Aleksander Essex. Rolling Re-evaluation of the Swiss Post e-Voting System. Version 1.2.3 and 1.3.0. Audit Scope 1: Cryptographic Protocol. August 1st, 2023. Available: https://www.bk.admin.ch/dam/bk/en/dokumente/pore/E_Voting/Examination_reports_August2023/Scope%201%20Final%20Report%20Aleksander%20Essex%2001.08.2023.pdf.download.pdf/

3 Review of Progress on Issues Identified in the 2023 Re-evaluation

This section reviews progress on issues previously identified in my August 2023 report. Of the twelve issues identified in the 2023 report, 9 have since been resolved. The three unresolved issues have been updated with a JIRA issue identifier (EX-xxx) for future reference.

3.1 Primitives Specification

This section summarizes the progress made on existing issues in the primitives specification as of version 1.4.0.

Issue: Double-testing of primes (Resolved)
Description: `GetEncryptionParameters` in the primitives specification applies Baillie-PSW and Miller-Rabin. Essentially, a prime candidate is tested twice using two different methods.
Recommendation: Discontinue the use of the BPSW primality test.
Action Taken: `GetEncryptionParameters` now only invokes `MillerRabin` for primality testing.

Issue: Unnecessary complexity in primality testing (Resolved)
Description: The primality testing required *two* two approaches to primality testing: (a) Swiss Post's custom variant of the Baillie-PSW test, and (b) a conventional application of Miller-Rabin.
Recommendation: Switch to deterministic generation of safe primes.
Action Taken: Deterministic safe prime generation has clear performance benefits as detailed in my 2023 report. However, the modifications made in `GetEncryptionParameters` in version 1.3.0 are acceptable.

Issue: Non-uniform prime generation (Resolved)
Description: Primes are no longer selected uniformly in the set of primes $\mathbb{P}$. Rather, an integer is selected uniformly, and the next highest prime is selected. This approach introduces a statistical bias due to varying gaps between primes.
Recommendation: Provide an analysis of this approach, both in terms of the security *cost* (to what degree could this impact GNFS polynomial selection), and the performance *benefit* (quantified relative to the standard approach).
Action Taken: This issue is superseded by EX-268 (see below).

Issue: No analysis or definitions of modulo bias mitigation (Resolved)
Description: `RecursiveHashToZq` internally hashed to $(\|q\|+256)$ bits before being reduced modulo q to mitigate the effect of a modulo bias. Swiss Post has claimed the modulo bias was sufficiently "small." However, no analysis was provided. Additionally, they claimed the modulo bias was "negligible," despite being constant in the security parameter.
Recommendation: The modulo bias should be quantified and shown to be negligible in the security parameter.
Action Taken: The issue was substantively (although not fully) addressed. `RecursiveHashToZq` (Algorithm 5.6, Line 1) now sizes the output h' as a function of security parameter λ, and the current parameter sizing is reasonable to mitigate the modulo bias however no quantification of the bias was provided.

Issue: Signature scheme should be fully specified (Resolved)
Description: Swiss Post specifies the `GenSignature` algorithm as outputting a signature of 384 bytes (3072 bits). Where does 384 come from? This seems to necessarily imply `Sign()` is an RSA-based signature.
Recommendation: If RSA signatures are intended, it should be specified along with the padding scheme used (e.g., PKCS1.5, PSS, etc.)
Action Taken: Signature scheme is clearly specified: RSASSA-PSS with $\|n\| = 3072$ bits (i.e., 384 bytes) in Table 3 of the primitives specification (as of version 1.4.0)

Issue: Possible typo (Resolved)
Description: In Line 4 of Algorithm 6.2, the output of the `Sign()` operation says to "See Algorithm 3.11", i.e., `StringToByteArray()`
Recommendation: Confirm if this is a typo or intended.
Action Taken: Reference to `StringToByteArray` removed.

Issue: Typo in Argon2id parameterization profiles (Resolved)
Description: Argon2id parameterization profiles (`STANDARD`, `LESS_MEMORY`, `TEST`) labeled "Memory (in GiB)," implying that the three profiles use 21, 16, and 14 GiB of memory, respectively.
Recommendation: The column should be changed to: "Memory (in KiB)" and the 21, 16 and 14 should be raised to the power of 2.
Action Taken: Corrected.

Issue: Define term (Resolved)
Description: Regarding Argon2id profiles, Swiss Post says: "The first profile STANDARD, ... is considered uniformly safe by RFC9106." It is unclear what "uniformly safe" means in this context, and RFC9106 does not appear to define it either.
Recommendation: Define "uniformly safe."
Action Taken: Term removed.

Issue (EX-268): Non-standard approach to verifiable parameter generation (Unresolved)

Description: Regarding selecting candidates for prime q, Swiss Post acknowledges the specific ways in which their *increment-and-test* approach deviates from the FIPS 186-5 *sample-and-test* approach. Swiss Post claims that its approach is "significantly faster" but provides no performance or security analysis.

In terms of the statistical bias of prime selection, no number-theoretic analysis is provided. Swiss Post claims: "with the bit lengths we are operating with ... a prime fixed in advance still has a negligible chance of being chosen."

There are two issues with this statement.

1. The bit lengths they are operating at include, by their own definition, a *testing-only* security level for which a prime *could* be "fixed in advance" given a sufficiently small security parameter λ. This has no impact on operational parameterizations. So they should clarify the claim applies at the *standard* security level only.
2. The statement misconstrues the nature of the original observation, which was about the non-uniform selection of primes from $\mathbb{P}$—not about the ability to target a specific $q \in \mathbb{P}$.

Recall that the original concern with verifiable parameter generation pertains to the possibility that an adversary could pick a p with a polynomial form more favourable to the general number field sieve (GNFS). What is the number-theoretic argument that this prime-gap selection bias will not provide the adversary with additional degrees of freedom towards optimizing p for GNFS?

In terms of efficiency, Swiss Post states the approach of picking a random starting point for prime candidate q and incrementing upwards by a fixed step size "significantly speeds up the algorithm's performance" relative to picking a random integer and computing the greatest common divisor with a pre-selected primorial (as in FIPS 186-5). At a minimum, Swiss Post should quantify this claim in terms of the order of magnitude of the claimed speed-up.

Recommendation: Provide a number-theoretic argument (if possible) that this prime-gap selection bias will not provide the adversary meaningful degrees of freedom towards optimizing the representation of p. Quantify the speedup of this optimization relative to the approach of FIPS 186-5.

3.2 System Specification

This section summarizes the progress made on existing issues in the system specification as of version 1.4.0.

Issue: Domain separation (Resolved)
Description: In Algorithm 3.9 `DeriveBaseAuthenticationChallenge()` Line 2: The extended authentication factor and start voting keys are being concatenated before being input to the Argon2id hash. This does not enforce an explicit domain separation.
Recommendation: Enforce domain separation of types.
Action Taken: Domain separation enforced by `StringToByteArray("Auth")` separator between `EA_id` and `SVK_id`.

Issue (EX-266): Explain how Argon2id memory profiles are applied (Unresolved)
Description: Several algorithms in the Systems Specification require the use of the "less memory" Argon2id profile: • Algorithm 3.14 (`DeriveBaseAuthenticationChallenge`) • Algorithm 3.13 (`DeriveCredentialId`) • Algorithm 4.9 (`GenCredDat`) • Algorithm 5.1 (`GetAuthenticationChallenge`) • Algorithm 5.2 (`VerifyAuthenticationChallenge`) • Algorithm 5.3 (`GetKey`) The Cryptographic Primitives Specification states that the `LESS_MEMORY` profile is "designed for environments with limited memory" but does not explain why it is specifically required in the algorithms listed above. These algorithms are associated with the voting client; however, the need for a low-memory profile is unclear. Is the intended benefit for the voting client at generation or for the voting server at verification?
Recommendation: Add a brief explanation describing how/when an Argon2id memory profile is chosen, e.g., `LESS_MEMORY` is used in the voting client.

Issue: (EX-265) Dates of birth as an authentication credential (Unresolved)

Description: Dates of birth are not a strong authentication credential

Recommendation: Ideally use another approach to voter authentication. At a minimum: specify concrete security requirements and provide a cyber-risk assessment of dates of birth as an extended authentication factor. Swiss Post should openly acknowledge these limitations if it intends to pursue this approach.

Action Outstanding:
Swiss Post briefly acknowledged some of the limitations: "this factor only offers limited security value due to its low entropy and potential exposure through sources like social media." Regarding dates of birth as an authentication credential, Post makes two unsupported security claims in Section 4.1.3 (`GetVoterAuthenticationData`):

1. "The extended authentication factor serves as a minor psychological hurdle for individuals attempting voter impersonation."
2. "The voter and cantons are the primary holders of this information. As such, the operators of the voting server, control or printing components must exert considerable effort to learn the date or year of birth from a large voter base."

Regarding 1: This framing should be removed or supported further. Are there any studies supporting this claim? How minor is *minor*? Does someone determined to impersonate a voter face any barriers? Could the characterization of being a "minor hurdle" provide a false sense of security?

It remains an important exercise to work through the adversarial capabilities and assumptions. For example, Swiss Post conceptualizes the threat actors regarding the voting server, control, and printing components, which under-generalizes the real-world threat surface to the mailed credential.

Regarding 2: Swiss PostThey state that an adversary "must exert considerable effort to learn the date or year of birth from a large voter base." As is the claim that "the voter and cantons" are the primary holders of date of birth information. These claims are contestable and should be withdrawn. There are many large data sets warehoused across government and industry linking identities with dates of birth. Because dates of birth are static and permanently tied to a voter, access to any of these datasets through a data breach (or even potentially through a legal, commercial purchase) would directly undermine the claim of the necessity of "considerable effort."

Recommend Swiss Post provide additional clarification and evidence supporting these security claims, or remove them.

4 New issues in 1.4.0

This section summarizes new issues found in the primitives specification as of version 1.4.0.

4.1 Parameter Generation

This section concerns changes to Algorithm 8.1 `GetEncryptionParameters` concerning primality testing of group parameters.

Let $(q+\delta)$, $2(q+\delta)+1$, be positive integers. One of the primary goals of Algorithm 8.1 (`GetEncryptionParameters`) is to transparently generate $(q+\delta)$, $2(q+\delta)+1$ such that both values are prime. The notation $\texttt{MillerRabin}(c,k)$ denotes $k \geq 1$ rounds of the Miller Rabin primality test applied to a prime candidate $c \geq 1$.

Implementations of the Miller Rabin test return $\perp$ immediately upon a candidate c failing any of the k test iterations. Composite integers c exist that could pass all k rounds of the test, but the probability of such an outcome diminishes exponentially in k if the bases are chosen randomly after c is fixed. However, for the overwhelming majority of composite integers c, $\texttt{MillerRabin}(c,k)$ will return $\perp$ in the first iteration.

As of Primitives Specification 1.4.0, primality testing in Algorithm 8.1 is done in two parts. This approach appears to be designed as an efficiency speedup: $(q+\delta)$ and $2(q+\delta)+1$ are each subjected to one round of the Miller Rabin primality testing algorithm, ostensibly as an initial filter to eliminate most non-prime candidates quickly. The second round acts as a full, cryptographically rigorous test, where the number of rounds is parameterized in the system security parameter λ.

The current test sequence in `GetEncryptionParameters` is:

1. $\texttt{MillerRabin}(q+\delta, 1)$
2. $\texttt{MillerRabin}(2(q+\delta)+1, 1)$
3. $\texttt{MillerRabin}(q+\delta, \lambda/2)$
4. $\texttt{MillerRabin}(2(q+\delta)+1, \lambda/2)$

Items 1 and 2 occur on Line 22 and Items 3 and 4 occur on Line 23 as summarized below in Table 1.

10: **do**
11: **do**
<increment δ to avoid smooth factors>
22: **while** $\neg(\text{MillerRabin}(q+\delta, 1))$ **or** $\neg(\text{MillerRabin}(2(q+\delta)+1, 1))$
23: **while** $\neg(\text{MillerRabin}(q+\delta, \lambda/2))$ **or** $\neg(\text{MillerRabin}(2(q+\delta)+1, \lambda/2))$

Table 1. Primality testing in `GetEncryptionParameters`.

As I wrote in my August 2023 report, the witnesses to the primality of $(q + \delta)$ and $2(q + \delta) + 1$ are not independent. If $(q + \delta)$ is prime, then the primality of $2(q + \delta) + 1$ is established by a single Fermat test, or, by extension, a single iteration of the `MillerRabin` function.

Theorem 4.1. *If* $(q + \delta)$ *is prime, and* $\mathsf{MillerRabin}(2(q + \delta) + 1, 1)$ *outputs* $\top$*, then* $2(q + \delta) + 1$ *is prime.*

Proof. The proof follows directly from the proof of Theorem 9.4 in my 2023 report. □

Theorem 4.2. *If* $\mathsf{MillerRabin}(2(q + \delta) + 1, 1)$ *outputs* $\top$ *and* $\mathsf{MillerRabin}((q + \delta), \lambda/2)$ *outputs* $\top$*, then* $2(q + \delta) + 1$ *and* $q + \delta$ *are both prime with probability greater than* $1 - (\frac{1}{2})^{\lambda}$

Proof. The proof follows directly from the proof of Theorem 9.5 in my August 2023 report. □

Corollary 4.1. *Performing* $\lambda/2$ *rounds of* MillerRabin *on prime candidate* $2(q + \delta) + 1$ *on* `Line 23` *of* `GetEncryptionParameters` *is mathematically redundant.*

The consequence is that the successful completion of Steps 1–3 establishes p's primality to the same bound as q—without running Step 4. We recommend updating `GetEncryptionParameters` to run the following operations in the following order:

1. $\texttt{MillerRabin}(q + \delta, 1)$
2. $\texttt{MillerRabin}(2(q + \delta) + 1, 1)$
3. $\texttt{MillerRabin}(q + \delta, (\lambda/2) - 1)$

This proposed sequence of operations retains Swiss Post's approach of initially filtering out any obviously composite values (Steps 1 and 2) while eliminating the redundant computation of Step 4. We also suggest decrementing the number of rounds in Step 3 by one, since one round was already performed in Step 1. See Table 2 for the specific proposed code modification.

```
10: do
11:     do

            <increment δ to avoid smooth factors>

22:     while ¬(MillerRabin(q + δ, 1)) or ¬(MillerRabin(2(q + δ) + 1, 1))
23: while ¬(MillerRabin(q + δ, (λ/2) − 1))
```

Table 2. Primality testing in `GetEncryptionParameters`.

Performance Comparison. We implemented and tested `GetEncryptionParameters` in Python (as per Table 1) as well as the proposed modification (as per Table 2).

The Miller-Rabin test was implemented directly using GMPY2's `is_strong_prp(c,b)` function, which accepts a prime candidate `c` and tests whether it is a strong probable prime to base `b`. Normally, bases `b` are randomized. However, for the purposes of direct performance comparison, both implementations were tested using a common, static list of bases in the respective `is_strong_prp` function calls. Additionally, both implementations were given the same randomly generated initial starting candidates q.

The proposed method showed a very marginal performance improvement (approx. 1%). This is to be expected; most of the computation is done in Line 22, which remains unchanged. Therefore, the impetus for the proposed change is more on simplifying the specification and analysis by eliminating unnecessary function calls.

Issue (EX-267): Cryptographically redundant computation in Algorithm 8.1
Description: Algorithm 8.1 (`GetEncryptionParameters`) prescribes cryptographically redundant primality tests on candidate p.
Recommendation: Remove redundant computation as per Table 2. See explanation above.

4.2 Other issues

Three other issues were found in the primitives specification. These issues were present as of document Version 1.4.0.

Issue (EX-261): Invalid group generator in several test vectors
Description: Several test vectors in the primitives specification use the group generator: `g: "0x4"`. However, $g = 4$ is no longer a valid output of Algorithm 8.4 (`GetEncryptionParameters`). Currently, only $g = 2$ or $g = 3$ is a possible output. This issue affects: • `get-ciphertext.json` associated with Algorithm 8.5 (`GetCiphertext`) • `get-ciphertext-product.json` associated with Algorithm 8.8 (`GetCiphertextProduct`) • `get-verifiable-commitment-key.json` associated with Algorithm 9.6 (`GetVerifiableCommitmentKey`) • `verify-exponentiation.json` associated with Algorithm 10.9 (`VerifyExponentiation`)
Recommendation: Search test vectors for this issue and generate new test vectors using valid group parameters generated by the current incarnation of `GetEncryptionParameters`.

Issue (EX-262): JSON parse error in test vector
Description: Parse error on line 13 of the `get-encryption-parameters.json` test vector in the primitives specification associated with Algorithm 8.1 (`GetEncryptionParameters`). Whereas Algorithm 8.1 should output three values: p, q and g, the test vector "output" field contains two outputs, i.e., two distinct sets of (p, q, g) domain parameters for a single seed value. One of these sets is an invalid output for the given seed.
Recommendation: Fix the JSON error. Remove the invalid (p, g, q) or provide the correctly associated seed value.

Issue (EX-264): Undefined input for some algorithms in the testing-only security level
Description: In a recent change, the "Security Strength" parameter λ is consumed and directly used by the following algorithms: • Algorithm 5.6 (`RecursiveHashToZq`) • Algorithm 5.10 (`KDFToZq`) • Algorithm 8.1 (`GetEncryptionParameters`) At the standard security level, $\lambda = 128$. However, in the testing-only security level, λ is unspecified. Therefore, the above algorithms would run on an undefined input when testing. It is unclear how the computation would proceed.
Recommendation: For the testing-only security level in Table 2, specify either (a) a concrete value for λ, or (b) specify a range of valid possible values for a user-chosen λ and provide a concrete algorithm for deriving $\|p\|$, and $\|q\|$ from it.

5 Examination of Changes in System Version 1.4.3

I reviewed the changes in the primitives specification from Version 1.4.0 to 1.4.1. I did not find any issues introduced by this update. At the time of writing I have not had the opportunity to examine the other documents for new issues.