# Rolling Re-Evaluation of the Swiss Post e-Voting System: Version 1.4.5

## Audit Scope 1: Cryptographic Protocol

Aleksander Essex

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

May 30, 2025

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 1.4.4 and 1.4.5. The changes were minor, and I make one low-priority recommendation for notation improvement in Section 2.1.

Additionally, I examined the subsequent "hotfix" (version 1.4.5.1) in response to a finding made by Thomas Haines regarding encryption nonce length. Although the issue was mitigated by the hotfix, Swiss Post's response characterized the finding as "security-relevant" but "low risk." I address this remark in Section 3 and provide the following cryptographic design recommendation:

1. **Recommendation: General-case Parameterizations**. Cryptographic components should be parameterized according to best practices related to their use in the general case, i.e., *independently* of context-dependent assumptions, such as the number of times a function is invoked.

## Version History

| | |
|---|---|
| June 22, 2025 | Final draft submitted to Chancellery. |
| June 1, 2025 | 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 |
|---|---|---|
| 2025 Rolling Re-Evaluation (This document) | 1.4.2 | 2025-05-30 |
| 2024 Rolling Re-Evaluation | 1.4.1 | 2024-07-16 |
| 2024 Rolling Re-Evaluation | 1.4.0 | 2024-07-16 |
| 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 |
|---|---|---|
| 2025 Rolling Re-examination (This Document) | 1.4.2 | 2025-05-30 |
| 2024 Rolling Re-examination | 1.4.0 | 2024-07-16 |
| 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 1.4.5

We discuss relevant findings regarding changes made to Swiss Post's e-Voting system in version 1.4.5. Specifically, we examine the changes made to the cryptographic primitives specification document as of version 1.4.2.[1] Additionally, we review changes made in the system specification document as of version 1.4.2.[2]

### 2.1 Summary of Changes to Primitives Specification

The primary change to the primitives specification is the introduction of Algorithm 3.10.

**Algorithm 3.10.** Algorithm 3.9 accepts an integer $x \in \mathbb{N}$ and returns $B \in \mathcal{B}^n$, encoding an integer $x$ as an array of $n$ bytes. Algorithm 3.10 was introduced in this latest version to extend the functionality of Algorithm 3.9, allowing the returned array to have a user-specified output byte length that is equal to or greater than $n$. If the user-specified length exceeds the inherent byte length of $x$, Algorithm 3.10 pads zero bytes (`0x00`) to the low indices of $B$, consistent with the little-endian encoding of Algorithm 3.9. However, there are two minor issues with the notation of Algorithm 3.10:

1. Algorithm 3.10 redefines the variable $n$ from the byte length of input $x$ in Algorithm 3.9 to mean the output length of $B$, and the byte length of $x$ is renamed $m$.
2. Lines 4–7 of Algorithm 3.10 duplicate the functionality of Lines 2–4 of Algorithm 3.9.

For notational clarity and consistency, we recommend: (1) replacing lines 4–7 of Algorithm 3.10 with a function call to Algorithm 3.9, followed by a for-loop to copy the returned result into the output vector $B$; and (2) retaining the notation of $n$ as the byte length of input $x$, while using $m \geq n$ to denote the byte length of Algorithm 3.10's output. This recommendation aims to communicate more immediately to the reader that (1) the output space of Algorithm 3.10 is a superset of Algorithm 3.9, and that the lengths of the outputs of the two functions may not be the same for equivalent inputs $x$.

### 2.2 Summary of Changes to Protocol Specification

The main change to the protocol specification is Algorithm in 4.9, which was updated to call the new Algorithm 3.10 from the primitives specification. No issues were observed.

## 3 Examination of Changes in "Hotfix" System Version 1.4.5.1

In an email to the expert examiners dated May 25, 2025, Swiss Post responded to a finding by Thomas Haines regarding the use of 48-bit nonces for password-based symmetric-key authenticated encryption (AES-GCM). This length is significantly below the generally recommended minimum of 96-bits,[3] which is necessary to prevent nonce reuse with cryptographically high probability. Swiss Post acknowledged it as a "security-relevant issue."

[1] Swiss Post Cryptographic Primitives specification version 1.4.2. 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 1.4.2. Available: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/raw/documentation-1.6.0.0/System/System_Specification.pdf

[3] See e.g., NIST Special Publication 800-38D. https://csrc.nist.gov/pubs/sp/800/38/d/final

### 3.1 Nature of the Change

Before this finding, nonces in the `StreamedEncryptionDecryptionService` class were generated as 12 random bytes drawn from the base 16 alphabet (i.e., `0-9A-F`), resulting in 4 bits of entropy per character:[4]

```
private static final int NONCE_BYTE_LENGTH = 12;
...
final byte[] nonce = random.genRandomString(NONCE_BYTE_LENGTH,
  Base16Alphabet.getInstance()).getBytes(StandardCharsets.UTF_8)};
```

This results in a nonce with only $4 \cdot 12 = 48$ bits of entropy. To implement the fix, "a small adjustment was necessary to make the `RandomBytes` method accessible." The updated code:

```
private static final int NONCE_BYTE_LENGTH = 12;
...
final byte[] nonce = random.randomBytes(NONCE_BYTE_LENGTH)};
```

correctly returns a nonce with the recommended 96-bits of entropy.

### 3.2 Cryptographic Implication of Nonce Collisions

We assume the cryptographic role of nonces in AES-GCM is well understood by Swiss Post; however, to briefly restate the implications: when the same nonce $n$ is used to encrypt two messages $m_1 \neq m_2$ with the same key $k$, it directly affects both the confidentiality and message integrity security guarantees.

Because AES-GCM operates in CTR mode, the attacker can make linear changes to the ciphertext, inducing linear changes in the resulting ciphertext. This, by itself, is insufficient to produce valid ciphertext. However, since AES-GCM's `GHASH` algorithm is linear, given the authenticated encryptions of $m_1, m_2$ encrypted with the same nonce $n$ and key $k$, an attacker can efficiently solve the constituent linear system to recover the `GHASH` key. The attacker can now create a valid MAC tag on the modified ciphertext using this key, enabling the production of valid ciphertexts with linear modifications to the underlying ciphertext. Moreover, with the authenticated encryptions of $m_1, m_2$, an attacker can recover $m_1 \oplus m_2$ by computing $c_1 \oplus c_2$, thus breaking semantic security.

### 3.3 Remark on Security Design

While the nonce length issue was addressed by the change noted above, Swiss Post made the following additional remark, to which we respond in this section:

> While the risk of a nonce collision is low, since the streamable encryption is applied to a very limited number of messages, we consider this a security-relevant issue and want to address it as soon as possible.

[4] https://gitlab.com/swisspost-evoting/e-voting/e-voting-libraries/-/blob/master/e-voting-libraries-domain/src/main/java/ch/post/it/evoting/evotinglibraries/domain/encryption/StreamedEncryptionDecryptionService.java

The risk of nonce collision depends on both the nonce's entropy and the number of messages encrypted with a common key. Our aim here is to highlight the implications of connecting a cryptographic security guarantee to a prescribed use case scenario, i.e., a "limited number of messages" encrypted under the same key.

For the risk to be *low*, the number of encrypted messages under a shared key must also be low. For example, if a single message is encrypted with a given key $k$, the probability of nonce reuse is zero. Swiss Post has identified at least three applications of the `StreamedEncryptionDecryptionService` class: SDM instances, verifier datasets, and encrypting voter print card PDFs. Based on a search of Swiss Post's GitLab repository, it was unclear exactly how the print card PDFs were being encrypted. However, if we assume that (1) a unique print card PDF is generated for each eligible voter and (2) each PDF is encrypted with a single password-derived key, we can proceed with a brief analysis of the probability of nonce reuse based on these assumptions.

**Analysis of Likelihood.** According to the birthday paradox, the probability of a collision between two randomly generated 48-bit nonces exceeds 50% after approximately 20 million encryptions. Assuming that, at some future date, all eligible voters in Switzerland are assigned such a print card, and using the number of eligible voters in 2025 (i.e., 5,620,065)[5] as a concrete example, the probability of a collision is approximately 5.5%. Assuming this probability remains constant and independent between elections (i.e., follows a binomial distribution), even if the password-based key is changed between elections, the likelihood of observing a nonce collision is greater than 50% after only 13 election events. This analysis highlights that the risk of nonce collisions depends on the number of messages encrypted under a common key and that such an outcome may not necessarily be "low" under real-world conditions at the 48-bit level.

**Recommendation.** Parameterization based on application-dependent assumptions creates a fragile security guarantee: Neither the primitives, system specifications, nor the Java code include any notices or controls restricting the number of instantiations of the `StreamedEncryptionDecryptionService` class. As the e-voting project evolves, this poses a risk that the class could be used in new contexts where unstated assumptions about its use are no longer upheld.

- **Recommendation: General-case Parameterizations.** Cryptographic components should be parameterized for security in the general case, i.e., *independently* of assumptions regarding their use, such as the number of times a specific function will be invoked.

[5] IFES Election Guide for the 2025 Swiss Referendum. https://www.electionguide.org/elections/id/4497/