Berner Fachhochschule (BFH), CH-2501 Biel, Switzerland

Re-Examination of the Swiss Post Internet Voting System

Releases 1.2.3 (February 2023) and 1.3 (April 2023)
(with Addendum on Version 1.3.1)

Rolf Haenni, Reto E. Koenig, Philipp Locher, Eric Dubuis

June 30, 2023

On behalf of the Federal Chancellery

Contents

1. Introduction

This examination report discusses some specific technical topics of the Swiss Post e-voting system that were affected by the changes included in the February and April 2023 releases of both the documentation and the source code. As such, it provides a short addendum to our extensive evaluation report, which we recently delivered to the Federal Chancellery (FCh) on February 23, 2023. The full report was published on the FCh's web page on March 3, 2023, together with the reports from other experts.[1]

1.1. Purpose and Goals of Mission

We have been assigned with this supplementary task in January 2023 by the Federal Chancellery, and we received more detailed information about our mission on March 21 (for the February release) and on May 4 (for the April release), 2023. A first draft of this document, which included a discussion of the assigned topics for the February release, has been sent to the Chancellery on March 31, 2023. The final version of this document with a complete discussion of all assigned topics has been sent to the Chancellery on May 26, 2023.

With respect to the February release, the mission attributed to us consisted in re-examining the following four specific topics:

- Parameter generation p and q;
- Algorithm RecursiveHashToZq;
- Updated dependencies and third-party libraries (only if dependencies/libraries are replaced by Post code);
- Auditors' manual checks in verifier specification.

An additional topic to consider was added to this list in response to the publication of the April release:

- Specification and implementation of voter authentication (ending in the keystores being delivered and decrypted by the voters).

The examination has been conducted jointly by the listed authors from the Bern University of Sciences and independently of any other group of people.

1.2. Documents and Source Code

To conduct our examination of the new releases, we downloaded the following updated specification documents from the public repositories on gitlab.com:[2]

- [CryptPrim] *Cryptographic Primitives of the Swiss Post Voting System – Pseudo-code Specification*, Version 1.3.0, Swiss Post Ltd., April 13, 2023

[1] See https://www.bk.admin.ch/bk/de/home/politische-rechte/e-voting/ueberpruefung_systeme.html
[2] See https://gitlab.com/swisspost-evoting.

- [SysSpec] *Swiss Post Voting System – System Specification*, Version 1.3.0, Swiss Post Ltd., April 19, 2023
- [VerSpec] *Swiss Post Voting System – Verifier Specification*, Version 1.4.0, Swiss Post Ltd., April 19, 2023
- [ProtProofs] *Protocol of the Swiss Post Voting System – Computational Proof of Complete Verifiability and Privacy*, Version 1.2.0, Swiss Post Ltd., April 19, 2023
- [ArchDoc] *E-Voting Architecture Document*, Version 1.3.0, Swiss Post Ltd., April 14, 2023

As in our previous reports, our main focus of this assessment was on the first three documents of the above list.

The first new software release was published on February 22 and 23. We received an e-mail announcement from Swiss Post with pointers to corresponding repositories on March 1. The new release was announced as *Release 1.2.3*, which apparently referred to the current version of the main component e-voting (see GitLab histories in Figure 1). The components crypto-primitives and verifier were released as Versions 1.2.1 and 1.3.3, respectively. In this document, we will generally refer to it as the *February release.*

The second new software release was published between April 13 and April 19. Again, we received an e-Mail announcement from Swiss Post on April 20. Most components were released as Version 1.3.0, except for the verifier and a new component called e-voting-libraries (they were released as Version 1.4.0 and Version 1.3.1, respectively). Another new component called Data-integration-service was released as Version 2.6.0. In this document, we will generally refer to all components included in the latest release as the *April release.*

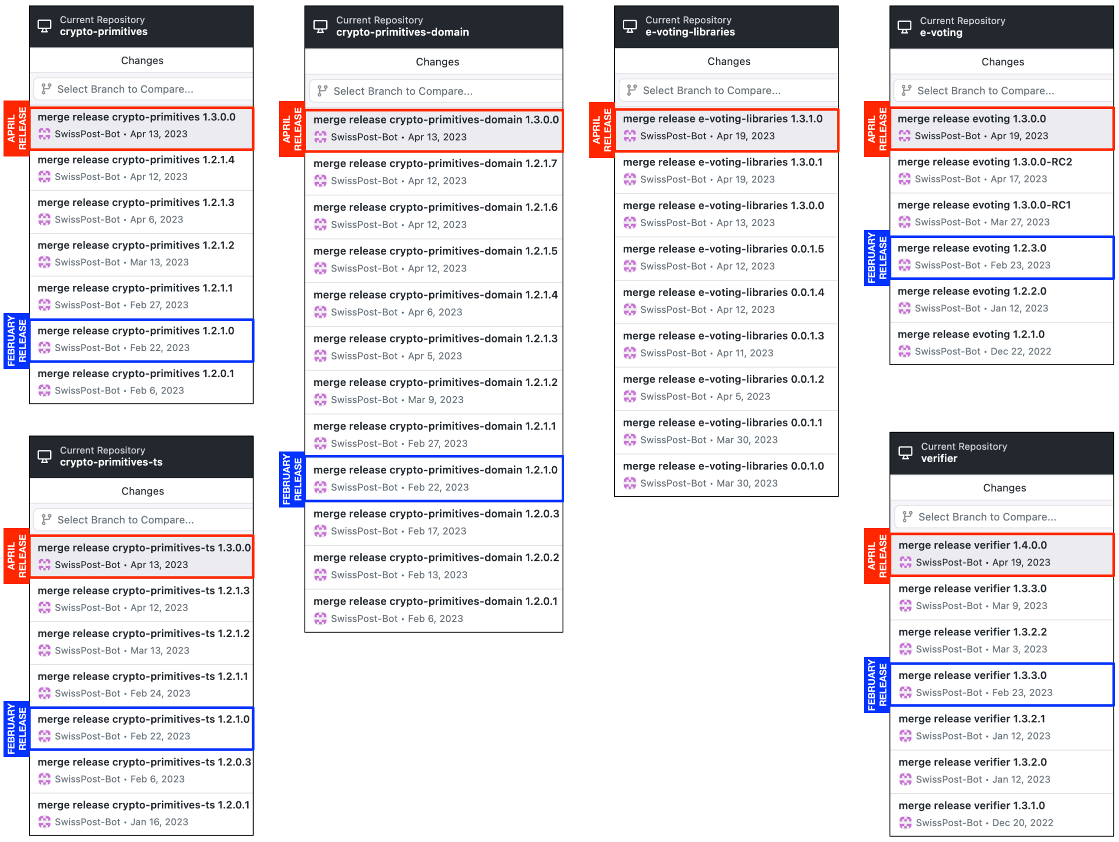
1.3. Changes Since December Release

1.3.1. February Release

The amount of changes made to the documentation and code were relatively moderate in the February release. According to the above-mentioned e-mail announcement, the changes have only a minimal impact on the cryptographic protocol:

> "*Release 1.2.3 focuses on the correction of functional bugs that were identified during the testing phase. These corrections primarily relate to the user interface displayed to the voter. Additionally, we updated third-party libraries to address known vulnerabilities. We would like to emphasize that the published releases have minimal impact on the implementation of the cryptographic protocol, except for a few specific cases.*"

As in earlier releases, we obtained special versions of two specification documents, in which the changes since the last version are made visible. All relevant changes were also accurately listed in corresponding CHANGELOG.md files. In the crypto-primitives and crypto-primitives-ts components, for example, we located only a few changes in five respectively four different files (see Figure 2), and the scope of these files corresponded exactly to the changes announced in the CHANGELOG.md file. In the E-voting and Verifier



Current Repository
crypto-primitives
Changes
Select Branch to Compare...

APRIL RELEASE
merge release crypto-primitives 1.3.0.0
SwissPost-Bot • Apr 13, 2023
merge release crypto-primitives 1.2.1.4
SwissPost-Bot • Apr 12, 2023
merge release crypto-primitives 1.2.1.3
SwissPost-Bot • Apr 6, 2023
merge release crypto-primitives 1.2.1.2
SwissPost-Bot • Mar 13, 2023
merge release crypto-primitives 1.2.1.1
SwissPost-Bot • Feb 27, 2023
FEBRUARY RELEASE
merge release crypto-primitives 1.2.1.0
SwissPost-Bot • Feb 22, 2023
merge release crypto-primitives 1.2.0.1
SwissPost-Bot • Feb 6, 2023

Current Repository
crypto-primitives-ts
Changes
Select Branch to Compare...

APRIL RELEASE
merge release crypto-primitives-ts 1.3.0.0
SwissPost-Bot • Apr 13, 2023
merge release crypto-primitives-ts 1.2.1.3
SwissPost-Bot • Apr 12, 2023
merge release crypto-primitives-ts 1.2.1.2
SwissPost-Bot • Mar 13, 2023
merge release crypto-primitives-ts 1.2.1.1
SwissPost-Bot • Feb 24, 2023
FEBRUARY RELEASE
merge release crypto-primitives-ts 1.2.1.0
SwissPost-Bot • Feb 22, 2023
merge release crypto-primitives-ts 1.2.0.3
SwissPost-Bot • Feb 6, 2023
merge release crypto-primitives-ts 1.2.0.1
SwissPost-Bot • Jan 16, 2023

Current Repository
crypto-primitives-domain
Changes
Select Branch to Compare...

APRIL RELEASE
merge release crypto-primitives-domain 1.3.0.0
SwissPost-Bot • Apr 13, 2023
merge release crypto-primitives-domain 1.2.1.7
SwissPost-Bot • Apr 12, 2023
merge release crypto-primitives-domain 1.2.1.6
SwissPost-Bot • Apr 12, 2023
merge release crypto-primitives-domain 1.2.1.5
SwissPost-Bot • Apr 12, 2023
merge release crypto-primitives-domain 1.2.1.4
SwissPost-Bot • Apr 6, 2023
merge release crypto-primitives-domain 1.2.1.3
SwissPost-Bot • Apr 5, 2023
merge release crypto-primitives-domain 1.2.1.2
SwissPost-Bot • Mar 9, 2023
merge release crypto-primitives-domain 1.2.1.1
SwissPost-Bot • Feb 27, 2023
FEBRUARY RELEASE
merge release crypto-primitives-domain 1.2.1.0
SwissPost-Bot • Feb 22, 2023
merge release crypto-primitives-domain 1.2.0.3
SwissPost-Bot • Feb 17, 2023
merge release crypto-primitives-domain 1.2.0.2
SwissPost-Bot • Feb 13, 2023
merge release crypto-primitives-domain 1.2.0.1
SwissPost-Bot • Feb 6, 2023

Current Repository
e-voting-libraries
Changes
Select Branch to Compare...

APRIL RELEASE
merge release e-voting-libraries 1.3.1.0
SwissPost-Bot • Apr 19, 2023
merge release e-voting-libraries 1.3.0.1
SwissPost-Bot • Apr 19, 2023
merge release e-voting-libraries 1.3.0.0
SwissPost-Bot • Apr 13, 2023
merge release e-voting-libraries 0.0.1.5
SwissPost-Bot • Apr 12, 2023
merge release e-voting-libraries 0.0.1.4
SwissPost-Bot • Apr 12, 2023
merge release e-voting-libraries 0.0.1.3
SwissPost-Bot • Apr 11, 2023
merge release e-voting-libraries 0.0.1.2
SwissPost-Bot • Apr 5, 2023
merge release e-voting-libraries 0.0.1.1
SwissPost-Bot • Mar 30, 2023
merge release e-voting-libraries 0.0.1.0
SwissPost-Bot • Mar 30, 2023

Current Repository
e-voting
Changes
Select Branch to Compare...

APRIL RELEASE
merge release evoting 1.3.0.0
SwissPost-Bot • Apr 19, 2023
merge release evoting 1.3.0.0-RC2
SwissPost-Bot • Apr 17, 2023
merge release evoting 1.3.0.0-RC1
SwissPost-Bot • Mar 27, 2023
FEBRUARY RELEASE
merge release evoting 1.2.3.0
SwissPost-Bot • Feb 23, 2023
merge release evoting 1.2.2.0
SwissPost-Bot • Jan 12, 2023
merge release evoting 1.2.1.0
SwissPost-Bot • Dec 22, 2022

Current Repository
verifier
Changes
Select Branch to Compare...

APRIL RELEASE
merge release verifier 1.4.0.0
SwissPost-Bot • Apr 19, 2023
merge release verifier 1.3.3.0
SwissPost-Bot • Mar 9, 2023
merge release verifier 1.3.2.2
SwissPost-Bot • Mar 3, 2023
FEBRUARY RELEASE
merge release verifier 1.3.3.0
SwissPost-Bot • Feb 23, 2023
merge release verifier 1.3.2.1
SwissPost-Bot • Jan 12, 2023
merge release verifier 1.3.2.0
SwissPost-Bot • Jan 12, 2023
merge release verifier 1.3.1.0
SwissPost-Bot • Dec 20, 2022

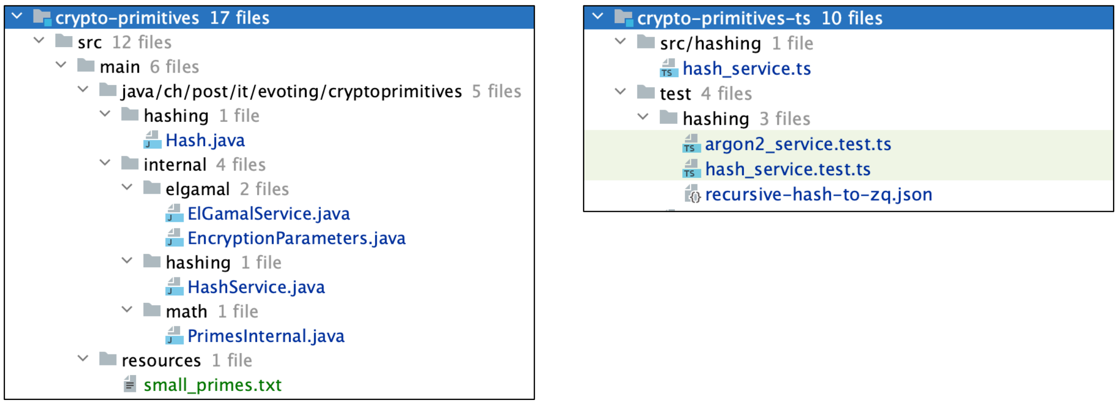
Figure 1: Commit histories of the GitLab projects since the December release.

components, we located a much larger amount of modified files files, but most of them were not relevant for the cryptographic protocol. The result of analyzing these changes are discussed in Section 2.

1.3.2. April Release

Some more fundamental changes were included in the April release. Again, we received special versions of four specification documents with all the changes highlighted. In the crypto-primitives and crypto-primitives-ts components, we observed only moderate modifications, which affect only a few files. These modifications correspond to what is listed in CHANGELOG.md files. In the crypto-primitives-domain component, we observed more fundamental changes (e.g. new classes such as `BallotBox`, `ElectoralBoard`, or `VotingCardSet`, and modified classes such as `Ballot`, `Question`, `ElectionOption`, or `PrimesMappingTableEntry`). While the names of these classes suggest that they are quite fundamental for the whole election system, they mainly reflect corresponding changes and extensions made to the e-voting component.

In the April release, we observed two substantial changes in the e-voting component. First, the voter authentication process has been redefined from scratch, both in the specification and the code. We see this as a response to our comments made in Appendix B.3.3 in our full evaluation report from February 2023. Second, by adding a third column called *semantic information*, an extension has been implemented to the primes mapping table



```
crypto-primitives  17 files
  src  12 files
    main  6 files
      java/ch/post/it/evoting/cryptoprimitives  5 files
        hashing  1 file
          Hash.java
        internal  4 files
          elgamal  2 files
            ElGamalService.java
            EncryptionParameters.java
          hashing  1 file
            HashService.java
          math  1 file
            PrimesInternal.java
      resources  1 file
        small_primes.txt
```

```
crypto-primitives-ts  10 files
  src/hashing  1 file
    hash_service.ts
  test  4 files
    hashing  3 files
      argon2_service.test.ts
      hash_service.test.ts
      recursive-hash-to-zq.json
```

Figure 2: Overview of February release changes in the crypto-primitives and crypto-primitives-ts components.

pTable to link its entries semantically to the actual list of voting options. This extension implied modifications at a various places of both the e-voting and verifier components, in the latter case for example by extending corresponding verification algorithms.

Other obvious changes in the April release are the two new Maven projects e-voting-libraries and data-integration-service. The goal of the former, according to its README.md file, is the following:

> "*The e-voting libraries serve as a central repository for implementing the data objects, methods, and algorithms utilized by various components of the Swiss Post Voting System. This centralization effectively minimizes code duplication while enhancing the code's quality, auditability, and maintainability.*"

The Maven project consist of three sub-modules domain, protocol-algorithms, and xml. The total amount of code in these modules is not very large (12, 18, and 19 classes, respectively), but especially the code containing protocol algorithms is very relevant for the whole system. Some of these algorithms (Factorize, QuadraticResidueToWriteIn, IntegerToWriteIn, isWriteInOption, DecodeWriteIns, GetMixnetInitialCiphertexts, VerifyMixDecOffline, VerifyVotingClientProofs) have been exported from the e-voting and verifier components with the goal of reducing code redundancy.

According to its README.md file, the purpose of the newly published Maven project data-integration-service, which consists of 100 additional Java classes, is the following:

> "*The Data Integration Service is a tool designed to process XML files describing the electoral roll and the configuration of election events. The tool produces a single, unified XML file that the Swiss Post Voting system can use.*"

To the best of our understanding, this system component is mainly used to process various files from the eCH-standard files, most notably eCH-0045 for the electoral role, eCH-0157 for an election, and eCH-0159 for a referendum. This component therefore provides an interface to the outer world of existing electronic election documents, which are managed by the cantons. According to [ArchDoc, Sect. 4], the information contained in these files is translated into three other XML files evoting-configuration.xml, evoting-print.xml, and evoting-decrypt.xml, which define the inputs and output for the voting system (see [SysSpec, Tab. 16] and [SysSpec, Tab. 18], respectively).

2. Summary of Findings: February Release

Given the short time period for this assessment, we restricted our analysis on the four specific topics mentioned in Section 1. This implies that we were unable to fully check if there are no changes in other parts of the code that are relevant for the cryptographic protocol. Each of the four topics is briefly discussed in one of the following subsections.

2.1. Parameter Generation

The performance of generating group parameters in algorithm GetEncryptionParameters has been improved in the February release. The current version can be seen as procedure of testing candidates for p and q on three nested layers:

- The inner while-loop (Line 14–21) implements a sieve relative to a given list of l small primes. The idea is to avoid more expensive primality tests for some candidates that are trivially composite.
- The outer do-while-loop (Lines 11–22) performs a fast probabilistic Baillie-PSW isProbablePrime test on both q and p.
- The final (more expensive) MillerRabin test on Line 30, again on both q and p, ensures that any failure of the faster isProbablePrime test will not lead to invalid group parameters.

First, and most importantly, the described algorithm will generate with overwhelming probability suitable group parameters in a verifiable manner. The main requirements are therefore met.

The fact that the added MillerRabin test returns $\perp$ in case of a failure indicates that it has only been installed as an additional safety net. However, since isProbablePrime is called with the security parameter λ as an additional parameter, the question that remains unanswered is whether this safety net is really necessary. Assuming that isProbablePrime guarantees a negligible failure probability of at most $\frac{1}{2^\lambda}$ (this information is missing in Section 7.1), then performing an additional MillerRabin test is clearly not necessary. On the other hand, if isProbablePrime returns composite values with non-negligible probability, then we would recommend installing an additional while-loop around the MillerRabin test. The fact that the current algorithm potentially returns $\perp$ instead of proper group parameter is unsatisfactory.

In the implementation of this algorithm, we observed that for isProbablePrime the standard Java method `BigInteger.isProbablePrime` is invoked. For the Miller-Rabin test, we found an additional method `millerRabin(BigInteger n, int rounds)` in the class `EncryptionParameters`. Unfortunately, the algorithm behind this method is not specified. It would therefore be good to include it also in pseudo-code, or at least give a proper reference, for example to the Handbook of Applied Cryptography [MvV96, Section 4.2.3]. It would also be good to give some information about the respective performances of the two algorithms.

Another observed change is the selection of the initial candidate in $[2^{|q|-1}, 1.5 \cdot 2^{|q|-1})$ instead of $[2^{|q|-1}, 2^{|q|})$ in Lines 2-3. The purpose of this restriction is unclear and we do not see an obvious advantage of it.

Further potential for improvements:

- We recommend defining $|p|$ as an input parameter (together with *seed*). $|q| = |p|-1$ can then be derived in Step 1 of the algorithm.
- On the other hand, we recommend removing the list of small primes from the input parameters. It can be hardcoded into the algorithm.
- Passing λ as a second argument to `isProbablePrime` without specifying its exact purpose is problematical. Also, according to Table 2, it is not clear what λ is, $\lambda \in \{\textit{testing-only}, \textit{legacy}, \textit{extended}\}$ or $\lambda \in \{-, 112, 128\}$?
- The number of rounds in the Miller-Rabin test is hardcoded to 64 (security level *extended*). This is inconsistent given the remark above about `isProbablePrime`. We recommend handling both cases equally.
- The variable $jump = 6$ should be either hardcoded or used everywhere (also in Line 4).
- Line 6: replace "$i \in [0, l\rangle$" by "$i \in [0, l)$".

2.2. Algorithm RecursiveHashToZq

The purpose of the algorithm RecursiveHashToZq in [CryptPrim, Section 4.2] is to hash one or multiple arbitrarily complex objects into elements of $\mathbb{Z}_q = \{0, \ldots, q-1\}$. The algorithm depends strongly on algorithm RecursiveHashOfLength, which computes a hash of an arbitrary bit length $\ell > 1$ using essentially the same recursive procedure as RecursiveHash for a fixed bit length $\ell = 256$.

The challenge of implementing RecursiveHashToZq is to avoid a bias in the resulting hash values, because implementations in which some values appear more frequently than others are prone to a variety of attacks. An example of an invalid implementation results from computing a single hash value of length $\|q\|$ using RecursiveHashOfLength and then taking the result modulo q. Depending on q, certain values in such an implementation will appear twice as often as others.

In the previous version of algorithm RecursiveHashToZq, both in the specification and the implemented code, such a bias was avoided by computing a sequence of candidate values $h' \in \{0, \ldots, 2^{\|q\|} - 1\}$ using RecursiveHashOfLength, until a proper value satisfying $h' \in \mathbb{Z}_q$ was found. In this solution, because each candidate has a success probability of $P(h' \in \mathbb{Z}_q) \geqslant 0.5$, the number of necessary rounds is usually very limited. Most importantly, a bias in the generated hash values is completely eliminated.

Another way of avoiding a statistically relevant bias is to generate hash values of length $\|q\| + k$ bits using RecursiveHashOfLength, and then take the result modulo q. In this solution, the bias vanishes with an increasing parameter k. It will never be eliminated entirely, but the statistical difference will be negligible already for relatively small values

k. The advantage of this approach over the first approach described above is its performance, because the hash value is always found in a single computational step, i.e., in constant time.

In the February release, RecursiveHashToZq has been changed from the first to the second approach discussed above, for a fixed value $k = 256$. We have found corresponding changes in both the specification and the code (Java class `HashService` in crypto-primitives and TypeScript class `HashService` in crypto-primitives-ts). In both cases, the code corresponds to the pseudocode algorithm.

2.3. Updated Dependencies and Third-Party Libraries

In the February release, a large number of dependencies have been updated. However, all of the updates are only minor version updates affecting about thirty Java dependencies and forty JavaScript dependencies. Two JavaScript dependencies for downloading Electron release artifacts and for making promise-based HTTPS requests have been removed from the SDM front end. And a new dependency for accessing spreadsheets has been added to the voter-portal.

We could not find any third-party library which has been replaced by source code written by Swiss Post. The two removed dependencies appear to be old and already unused. There are also no updates which directly affect the cryptographic algorithms. In general, we appreciate Swiss Post's efforts to keep dependencies up to date and consider it an important measure for the security of the overall system.

2.4. Manual Checks in Verifier Specification

The Verifier Specification in the February release has reached Version 1.3.1. Please note, in the GitLab repository it is referred to as Version 1.3.2 (see Figure 3). Within this new version, the main focus has been set to the improvements of the description for the manual checks for the human auditors.

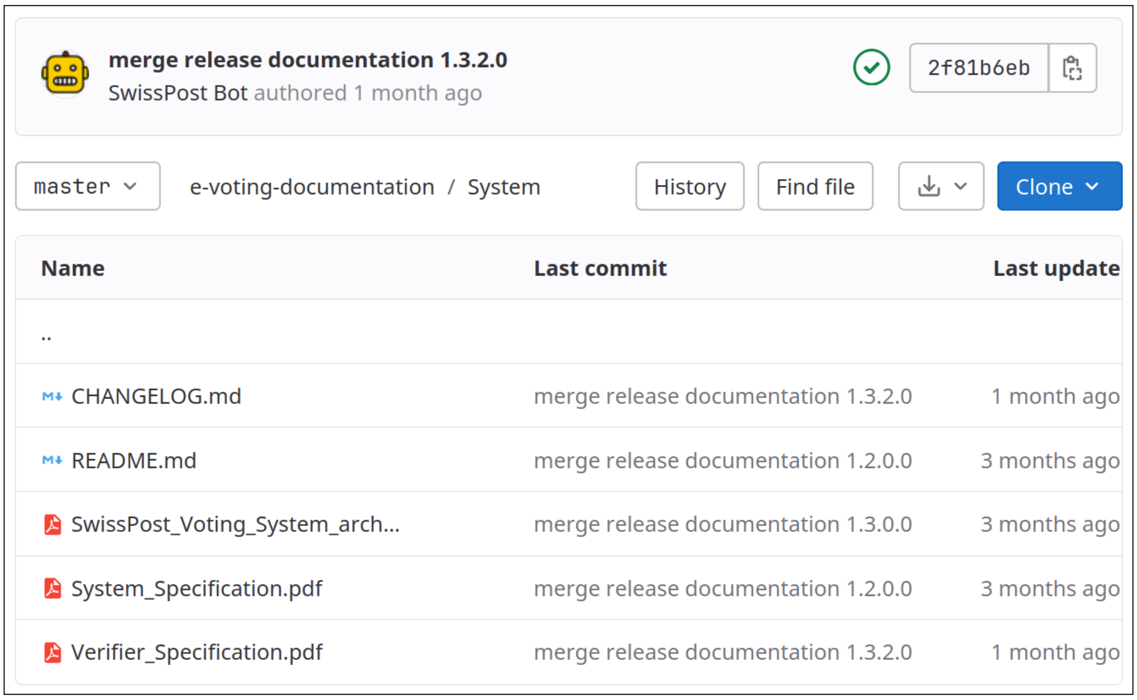
The beginning of the section provides an interesting assumption:

> "*We assume that the auditors have access to the necessary information regarding the expected, correct configuration of the election event. This includes knowledge of the following parameters: ... The auditors can learn this information from publicly available sources, compare it against data from previous election events, or confirm it against the actual electoral roll.* "

This way we conclude that there is no official document possibly signed by the canton, that an auditor can rely on. This, however, implies that there is no ground truth and thus there is an open parameter on which disagreement can occur. We highly recommend to fix that open parameter by signing the 'necessary information'.

In 0.01 ManualChecksByAuditors an explanation on abstract level has been introduced. This provides a level of formalism one can start building a concrete procedure upon. But then, a subtle question arises with the expression

> "*Check that all expected verifications executed successfully*"


merge release documentation 1.3.2.0
SwissPost Bot authored 1 month ago

2f81b6eb

master · e-voting-documentation / System

Name	Last commit	Last update
..		
CHANGELOG.md	merge release documentation 1.3.2.0	1 month ago
README.md	merge release documentation 1.2.0.0	3 months ago
SwissPost_Voting_System_arch...	merge release documentation 1.3.0.0	3 months ago
System_Specification.pdf	merge release documentation 1.2.0.0	3 months ago
Verifier_Specification.pdf	merge release documentation 1.3.2.0	1 month ago

Figure 3: Confusing version annotation for the verifier specification.

What is considered a successful verification? Is a verification successful if it finds a mistake within the protocol run? We propose a more precise wording here, e.g. "*Check if all the expected verifications accept the protocol run under inspection*".

The notion of *'spot checking'* is used but has not been introduced within the document. From our perspective, we cannot deduce what exactly has to be done for spot checking within this context.

Within this section, some of the tasks are described as optional at multiple steps within the chain of verification.

> "*To further increase transparency and accountability, the auditors should record some of the election results, which can be compared against the official published results at a later time. This record-keeping can help to identify any potential discrepancies and ensure that the election results are reliable and trustworthy.* "

It remains unclear if the auditor have to do that or not. We propose to make these 'optional' procedures compulsory and to provide a high level description on how to pick 'some' election results and how to compare them against the official published results.

The following 'optional' verification step in the beginning of the section, however, requires the auditors to question the content of a signed document.

> "*The configuration of the election event—signed by the canton (see section 3.2)—contains additional information about the election event, such as the precise wording of the questions, the names of the candidates and lists, and an identifier for each voting option. While the auditors can assume that this*

> *additional information is correct, it is still recommended that they manually verify the accuracy of this information by cross-checking it against other available sources.*"

Here it remains unclear what exactly the auditor has to verify. Is this still part of the universal verifiability of the e-voting channel? If so, why is it optional? If not, why is it proposed?

3. Summary of Findings: April Release

Given the main task assigned to us by the Federal Chancellery (see Subsection 1.1), we focused our evaluation of the April release on the redefined voter authentication process. The results and conclusions of our analysis are discussed in Subsection 3.1. Comments about observations made while looking at other changes made to the system are listed in Subsection 3.2, and a list of minor problems is given in Subsection 3.3.

3.1. Voter Authentication

The underspecified authentication process was a missing element in the specification documents of earlier versions of the system. In the December release, additional information and explanations about the authentication process have been added to Section 5.1 of the system specification. In Appendix B.3.3 of our final report, we criticized the solution as unnecessarily complicated for no obvious benefit, and we pointed to the fact that the actual implementation was very different from the specification. Since the implementation looked as a leftover from the earlier Scytl system, we recommended to eliminate it completely.

As a response to our criticism, the authentication process has been re-designed and re-implemented from scratch. The current version now implements an approach that is based on *time-based one-time passwords* (TOTP) as defined in RFC6238, which are derived from the *election event identifier* $\mathtt{ee}$, the *start voting key* $\mathtt{SVK}_{\mathtt{id}}$, and an *extended authentication factor* $\mathtt{EA}_{\mathtt{id}}$ (the voter's birth date given as a decimal string of lenght $\mathtt{l}_{\mathtt{EA}}$).[3] By fixing the TOTP time interval to the default value $T_X = 30$ seconds, the generated one-time passwords change twice every minute based on the current time. Note that TOTP requires the system times of the server and the client to be synchronized, which may not always be the case.[4] In such exceptional cases, voter authentication will obviously not work as expected.

According to [SysSpec, Fig. 8] in the updated Section 5.1 about voter authentication, the current TOTP value is computed by the voting client using the new algorithm GetAuthenticationChallenge. The algorithm first computes a hash $\mathtt{hAuth}_{\mathtt{id}}$ from the given inputs $\mathtt{ee}$, $\mathtt{SVK}_{\mathtt{id}}$, and $\mathtt{EA}_{\mathtt{id}}$, and then uses Argon2id to derive the resulting TOTP value $\mathtt{hhAuth}_{\mathtt{id}}$ from $\mathtt{hAuth}_{\mathtt{id}}$, the current time counter $\mathtt{T} = \lfloor \frac{\mathtt{TS}}{30} \rfloor$, and a voter-specific salt $\mathtt{salt}_{\mathtt{id}}$. This is roughly in accordance with the TOTP standard as defined in RFC6238, except for the usage of Argon2id.

On the server side, the (untrusted) voting server uses algorithm VerifyAuthenticationChallenge to repeat essentially the same computations based on given values $\mathtt{hAuth}_{\mathtt{id}}$ for all voters. If the resulting TOTP value matches with the one submitted by the voting client, and if the number of authentication attempts has not yet exceeded the threshold

[3]Unfortunately, we were unable to locate the specification of the format and length of the $\mathtt{EA}_{\mathtt{id}}$ string. However, we conclude from existing test data that the strings are of length $\mathtt{l}_{\mathtt{EA}} = 8$ in the format `"ddmmyyyy"`, for example `"01061944"`.

[4]Due to a bug in the camera software, users of Microsoft Surface Pro X computers are currently asked to modify their system time back to May 23, which apparently is a workaround for resetting the blocked cameras until appropriate software updates are available, see article on https://www.itmagazine.ch from May 26, 2023.

of 5 attempts, authentication is considered successful. In the success case, the voter's keystore is delivered to the voting client, who then calls algorithm GetKey to open the keystore using the start voting key $\mathtt{SVK}_{\mathtt{id}}$. In the failure case, the process can be repeated until the maximal number of attempts has been reached.

In the updated specification document, the above initial process is called *AuthenticateVoter Sub-Protocol*, which precedes the existing vote casting process. However, we learned that essentially the same authentication is repeated twice, first at the beginning of the SendVote sub-protocol and second at the beginning of the ConfirmVote sub-protocol (see [SysSpec, Figs. 9 and 10]), but with the same values $\mathtt{ee}$, $\mathtt{SVK}_{\mathtt{id}}$, and $\mathtt{EA}_{\mathtt{id}}$ already known by the voting client (only the current time counter $\mathtt{T}$ changes at each of the three invocations).

Based on our understanding of the above simple process, we have some general remarks about the appropriateness of the chosen approach:

- According to [OEV, Paragraph 2.8], it is sufficient for the protocol to provide *effective authentication*, which means that "*it must be ensured that no attacker can cast a vote in conformity with the system without having control over the voters concerned*". In the given protocol, this requirement is already met without the newly introduced voter authentication sub-protocol, because possessing the right start voting key $\mathtt{SVK}_{\mathtt{id}}$ is a necessary precondition for the vote casting process. Therefore, the benefit of the added authentication steps remains unclear, especially in the presence of an untrusted voting server, who will possibly not truthfully follow the authentication steps added to the three sub-protocols AuthenticateVoter, SendVote, and ConfirmVote.

- The previous remark can also be expressed from an adversarial perspective. Clearly, a scenario in which there is a strategy for an adversary to circumvent effective authentication, for example by controlling the voting server, could only be explained by flaws in protocol itself. Since this is not the case according the existing formal proofs, such an adversary can be excluded independently of the added authentication sub-protocol. Note that the discussion in [ProtProofs, Sect. 11.2] on *Voter Authentication* comes to exactly the same conclusion. The actual attack scenario, which the proposed extension is supposed to prevent, is therefore unclear.

- In our own attempt to formulate a convincing attack scenario, we imagined a situation in which an adversary obtains the start voting key from the voter, for example by collecting unopened election envelopes from abstaining voters at the paper recycling station. In such a case, the attacker would only need to obtain the voter's birth date to prevent being stopped by the maximum number of authentication attempts. Since the voter's name and address is printed on the envelope's address tag, we believe that voter's birth date can be found quite easily in most cases, for example on personal social media pages. From a security perspective, using the voter's birth date as an additional authentication factor seems not to provide much. [updated in June release: Some justifications for using the voter's birth date are given in the updated Section 4.1.4]

- Another explanation for including the voter's birth date as an additional authentication factor, even if the added value for the security is very limited, is the continuation of existing user workflow from previous versions of the system. Assuming that there are such good reasons to hold on to the voter's birth date, we believe that the current TOTP-based approach is possibly not an ideal choice. Generally, TOTP is a technique for generating password that expire quickly, which prevents attackers to use stolen password over a long period. We do not see such a scenario in the given application context. From our perspective, it seems that a technology is used without having a good reason to do so.

- With the current approach, there may be at least two usability problems. The first comes from the above-mentioned synchronization requirement between server and client, which in exceptional cases may prevent a voter from casting the vote. The second problem comes from the fixed TOTP expiry period of 30 seconds, which implies that the three invocations of the authentication process cannot be conducted in less than one minute. For voters that are familiar to the user interface and the vote casting process, this may lead to an unnecessarily prolonged voting process, for example in simple cases of a single referendum. [updated in June release: see Appendix A.3.1]

To conclude our analysis of the new voter authentication process, we refer to our remarks and recommendations in Appendix B.3.2 of our final report from February 2023. We proposed a much simpler solution, where the hash value $h_{\mathtt{id}} = \mathsf{hash}(\mathtt{SVK}_{\mathtt{id}})$ of the start voting key is submitted to the voting server as a key for selecting the right key store $\mathtt{VCks}_{\mathtt{id}}$ from a dictionary. If—for whatever reasons—the voter's birth date is indispensable as an additional authentication factor $\mathtt{EA}_{\mathtt{id}}$, this approach could be extended easily by adding $\mathtt{EA}_{\mathtt{id}}$ as a secondary key to the dictionary. Furthermore, if the maximum number of attempts must be limited to 5, then a counter could be assigned to every value $h_{\mathtt{id}}$. In this way, the two new algorithms GetAuthenticationChallenge and VerifyAuthenticationChallenge could be greatly simplified and the two above-mentioned usability problems could be avoided.

3.2. Other Changes

In our attempt to obtain a comprehensive overview of all relevant changes, we managed to look at all three system components. The results of our analysis are discussed in the following subsections.

3.2.1. Crypto-Primitives

The following list gives an overview of the major changes made to the crypto-primitives component. In most cases, we were able to locate them consistently in the specification and the code. In some cases, we have recommendations for improvements:

- In [CryptPrim, Sect. 4.5], three profiles for Argon2 algorithm parameters are defined according to RFC9106. The first two entries in the given table correspond to the two RFC9106 options "FIRST RECOMMENDED" and "SECOND RECOMMENDED", while the third entry is for testing purposes. A new Java class `Argon2Profile` has been added to the crypto-primitives code base, which provides the algorithm parameters as defined in the profiles (similar for TypeScript). What is currently very confusing is the fact that there are now three different Java classes for almost the same purpose. In the code, a given initial enum object `Argon2Profile` is transformed into a `Argon2Context` object, which is finally transformed into an `Argon2Config` object, while the reasons or benefit of this procedure remains unclear. This part of the code should clearly be refactored.

- Algorithm KDFToZq in [CryptPrim] and the corresponding method `KDFToZq` in the Java class `KDFService` have been changed in a similar way as in algorithm RecursiveHashToZq, i.e., from the first to the second approach discussed in Subsection 2.2. The same reasoning can be applied to justify the changes. The Java code and the pseudocode are aligned.
- Small performance improvements have been implemented in the following Java classes by replacing computation $y^{-e} \bmod p$ by $(y^{-1})^e \bmod p$ for small exponents e (similar for TypeScript):
  - `SchnorrProofService`,
  - `DecryptionProofService`,
  - `ExponentiationProofService`,
  - `PlaintextEqualityProofService`.

  These changes have not been made explicit in the respective pseudocode algorithms. Even if the alignment is quite obvious, we recommend adjusting them accordingly.
- A list of X.509 properties has been added to [CryptPrim, Sect. 6.1]. Since the certificates are self-signed, this is not a critical topic.
- The Maven plugin dependency-check-maven has been added to the project's pom.xml. This plugin automatically checks project dependencies for known published vulnerabilities, which is a useful extension for the project's build process.

3.2.2. E-Voting

Besides introducing a new voter authentication mechanism (see discussion in Subsection 3.1), numerous modifications have been made to a large amount of files in the code base of the e-voting component. Within the limited evaluation period, we were unable to conduct a comprehensive analysis of everything that has changes since the December release. Therefore, we have to restrict the following discussion to two specific topics, which we think are most relevant from a security perspective:

- A major general change, that affects many parts of the system (including the verifier), is the inclusion of "semantic information" into the primes mapping table pTable. This extension is described in [SysSpec, Sect. 3.4.2]. Each entry of pTable is now a triple $(v_i, \tilde{p}_i, \sigma_i)$, where $\sigma_i \in \mathbb{A}_{UCS}^*$ denotes the added semantic information (a string of arbitrary length). While the content and format of these strings is not specified in [SysSpec], we observed in the exemplary datasets of the verifier component, that they are combined textual descriptions of respective voting questions and options in German, for example `"2a. Wollen Sie den Vorschlag` `annehmen?-YES"`.[5] We assume that this information is extracted either from the configuration file configuration-anonymized.xml or directly from the official eCH-0157 or eCH-0169 documents obtained from the cantons.

To ensure that all participating parties have the same view with respect to the given voting options, the data from the voting option entries of pTable, including the newly added semantic information, is included in the voting client's zero-knowledge proofs and in the associated data of the verification key stores. Corresponding algorithms have been adjusted consistently in both the specification and the code. In the current implementation, inconsistent views would now necessarily imply situations in which some proofs could no longer be verified or some key stores could not no longer be opened successfully, i.e., adversarial attempts to infiltrate inconsistent views would be detected by the involved parties. Note that this was already the case before introducing this extension.

To the best of our understanding, the actual benefit from the implemented extension happens during the verification, which now includes an extended consistency check VerifyPrimesMappingTableConsistency [VerSpec, Verification 3.08] that compares the content of pTable with the content of the configuration XML document. This check helps to avoid some of the problems and attack scenarios that we described in Section 3.1.3 and Appendix A.3.2 of our full report from February 2023. Therefore, we have no general objections against this extension, even if the amount of redundant data in the election model is further increased. Note that in the February 2023 report, we recommended (1) to remove pTable from the specification and the code, and (2) to introduce of a more comprehensive formal election model. Since our recommendations have not been considered, the potential for simplifications is still present in the current system.

With respect to the implemented extension, we have the following two comments:

– Restricting the semantic information to a particular language looks like an arbitrary choice, even if from our understanding this does not imply an obvious problem. However, in a more consequential implementation, we would expect that different user languages are treated with "equal rights", for example by concatenating the textual descriptions of all available languages.

– To the best of our knowledge, the voting client does not check if the semantic information included in pTable corresponds to what is actually displayed to the voter. This is what we would have expected, even if cast-as-recorded verifiability seems to prevent corresponding attack scenarios.

- By removing the legacy cryptolib and cryptolib-js libraries, the number of third-party JavaScript libraries has been significantly reduced. This is directly reflected in the size of the ov-api, which contains all the cryptographic functions and operations on the client side. The size of the minified version of the ov-api has been reduced by about 44% from 1.587 MB (build 1.2.0) to 891 KB (build 1.3.0). This cleanup of legacy code is highly appreciated. To further improve transparency and scrutiny of the live system, we strongly recommend distributing the unminified version of the ov-api instead of the minified version. The unminified version is 2.222 MB in

[5] In [VerSpec, Tab. 4], more explanations are given about this topic, but the specified format, for example `"question - answer"` for standard questions in a referendum, does not exactly correspond to the given data sets (no spaces around the dash symbol).

size, which is only 2.5 times larger than the minified version, and we think the size can easily be reduced further. For example, by removing the utility library lodash, which is becoming more and more obsolete thanks to new JavaScript built-in features, the size of the unminified version could be reduced by about 24% to 1.678 MB. Thanks to the intrinsic modularity of lodash it is also possible to include only very specific utility functions instead of the entire library. We think, given the importance and sensitivity of the cryptographic operations on the client side, any attempt to further cleanse the ov-api would be worth the effort.

3.2.3. Verifier

In the April release of the verifier component, the amount of observed modifications is relatively moderate. Some changes are related to the extended pTable (see discussion in Subsection 3.2.2), and some changes are related to the missing consistency check relative to the file eCH-0222.xml in earlier versions. Here is a short summary of all observed changes:

- Verification 1.01 – VerifySetupCompleteness: A new verification added to Setion 3.1 for checking the completeness of the setup data (given as a textual description rather than pseudocode).
- Verification 3.08 – VerifyPrimesMappingTableConsistency: An extended version of an existing verification to check that the setup component's primes mapping table corresponds to the configuration XML. It refers to a new Table 4 in Subsection 3.3, which defines roughly the format of the semantic information for different voting option types.
- Verification 6.01 – VerifyTallyCompleteness: A new verification added to Setion 4.1 for checking the completeness of the tally input data (given as a textual description rather than pseudocode, similar to Verification 1.01).
- Algorithm 4.2 – VerifyTallyControlComponentBallotBox: Modified algorithm with adjustments to the extended pTable.
- Algorithm 4.3 – VerifyProcessPlaintexts: Modified algorithm with adjustments to the extended pTable, including calls to sub-algorithms GetEncodedVotingOptions and GetActualVotingOptions for selecting corresponding rows of pTable.
- Algorithm 4.4 – VerifyTallyFiles: An extended version of an existing algorithm to check the consistency of the provided eCH-0222.xml file. In earlier version, this check was missing.

3.3. Further Recommendations

While conducting our assessment on the February and April releases, we found some additional, mostly minor problems in both the specification documents and the code. We list them in the following two tables and provide some recommendations for improvements.

Document	Page	Comments
[CryptPrim, Sect. 6.1]	34	In the added statements about the X.509 certificate, it is unclear what is meant with "*should have the following properties*". How mandatory are the listed properties?
[SysSpec, Fig. 2]	10	The illustration of the voting process in Fig. 2 has not been adjusted to the inclusion of the authentication process. For example, entering the birth date is missing.
[SysSpec, Sect. 3.4.2]	21	Change "The auditors include [...] the voting client's zero-knowledge in VerifyTally" into "The auditors include [...] the voting client's zero-knowledge proofs in VerifyTally".
[SysSpec, Sect. 3.6]	29	The value given for e is incorrect. From $\lvert\mathbb{A}_{Base32}\backslash\{\texttt{'='}\}\rvert = 31$ and $\mathtt{l}_{\mathsf{SVK}} = 24$ we conclude $e = \lceil 24\log_2(31)\rceil = 119$, not $e = 120$. In the sequel, an estimation of how many security bits are effectively added by Argon2 is missing. Argon2 should be configured to compensate for the current security gap of 9 bits in the 128 bits security level. [Remark: Our statement is incorrect in the current context, where $\mathbb{A}_{Base32}$ contains the padding character $\texttt{'='}$, i.e., $\mathbb{A}_{Base32}$ contains 33 not 32 characters, which ultimately leads to $e = 120$.]
[SysSpec, Fig. 6]	39	The data required in SetupVoting (Fig. 6) is strongly dependent on information created during SetupTally (Fig. 7). So the current order of the two figures is very confusing. We recommend switching them (this may require the enclosing sections to be restructured).
[SysSpec, Fig. 6]	39	Calling the new algorithm GetVoterAuthenticationData in the upper part of SetupVoting, where it is not yes needed, is a bit confusing. We recommend moving it further down.
[SysSpec, Alg. 4.4]	43	The remark that a 256-bit hash value corresponds to 44 characters in Base64 is incorrect, it should be $\lceil 256/\log_2(64)\rceil = 43$ characters. [Remark: Our statement is only partially true. While representing 256 bits with an alphabet of size 64 (6 bits for each character) requires only $\lceil 256/6\rceil = 43$ characters, the Bases64 encoder adds a padding characters to guarantee that the encoding length is a multiple of 4 (for reasons that are unimportant here). Since the Base64 decoder discards these extra padding characters, the padding character could be skipped for making the encoding one character shorter.]
[SysSpec, Sect. 4.1.4]	43	The text following the pseudocode of Algorithm 4.4 is at the wrong place.

[SysSpec, Fig. 8]	57	The information flow between the voting server and the voting client as shown in Fig. 8 is incomplete. For example, calling algorithm CreateVote in Fig. 9 by the voting client requires much more input values than the transmitted ones shown in Fig. 8.
[SysSpec, Tab. 14]	59	The entry for `hhAuthid` should not be listed in this table, because it is invoked *during* the algorithm, not before.
[SysSpec, Table 16]	99	In Fig. 9 and 10, $\texttt{vc}_{\texttt{id}}$ has been added to the messages exchanged between the voting client, the voting server, and the CCRs. However, corresponding entries have not been update in Table 16.

Java Class	Comments
`PrimesMappingTable`	The JavaDoc has not been adjusted to the extended entries of the primes mapping table (a statement about `semanticInformation` is missing).
`ConversionsInternal`	A new try-catch statement has been introduced to the `StringToByteArray` method to test that the input is a valid UTF-8 string, but nothing has been changed in the pseudocode of the StringToByteArray algorithm in [CryptPrim, Sect. 3.3], thus creating a misalignment.
`Conversions`	Defining an interface with no abstract method, but a total of 7 static method, which do nothing else than calling corresponding methods from the class `ConversionsInternal` (which does not implement `Conversions`) makes absolutely no sense from software-engineering point of view.

A. Addendum-1: June Release

Updates of both the specification and the code were announced on June 16 for the documents [CryptPrim], [SysSpec], [VerSpec], and for all software components. Most software components are now in Version 1.3.1 or 1.3.2 (except for the verifier and data-integration-service, which follow their own version numbering schemes). In the sequel, we will refer to these new versions as the *June release* (see release history in Figure 4).

We agreed with the Federal Chancellery to inspect the implemented changes within a 2-weeks period. The purpose of this addendum section is to give an overview of the changes made in the June release, to discuss these changes from the perspective of the cryptographic protocol, and to verify if any of the changes affects the findings listed in the previous sections of this report. As in Subsection 3.2, we structure our analysis according to the changes made in each of the three main system components.

A.1. Overview of Changes

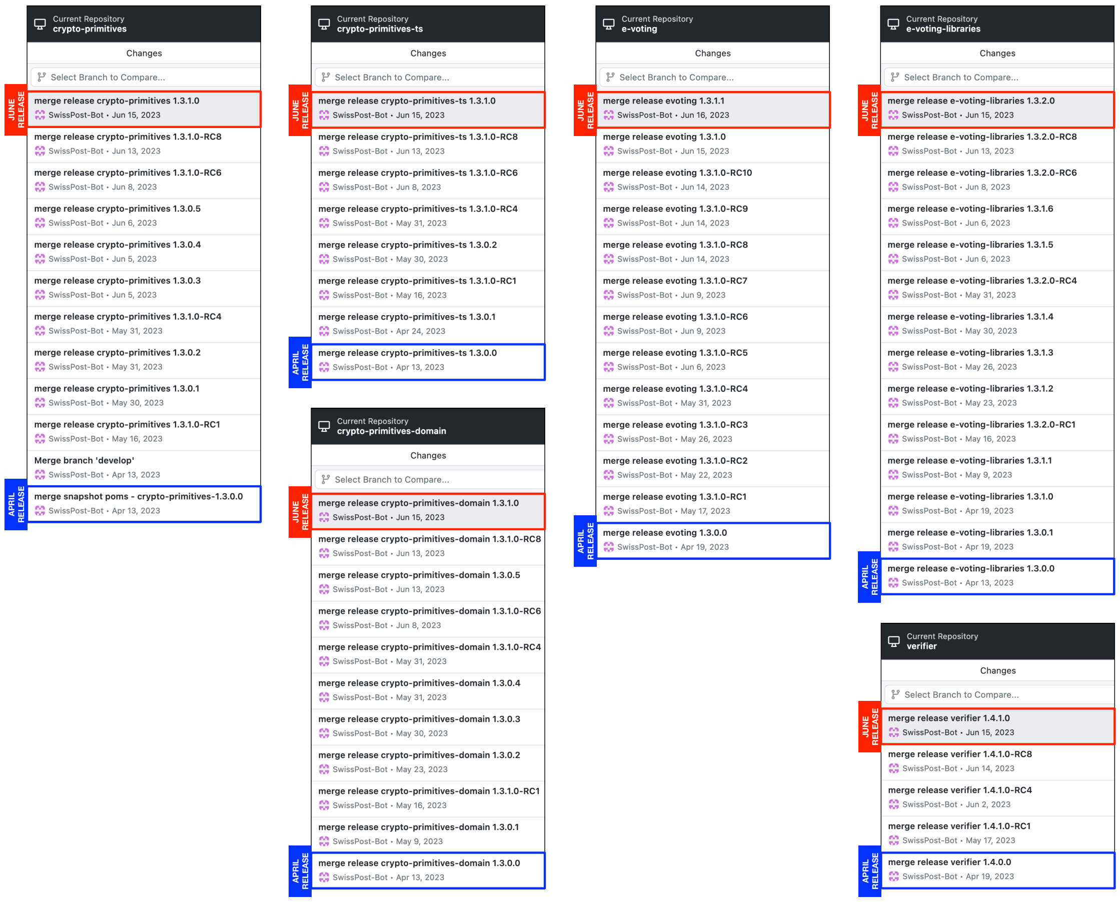
Here is the list of the current documents that we considered in our analysis of the June release. Note that [ProtSpec] and [ArchDoc] have not been updated:

- [CryptPrim] *Cryptographic Primitives of the Swiss Post Voting System – Pseudo-code Specification*, Version 1.3.1, Swiss Post Ltd., June 15, 2023
- [SysSpec] *Swiss Post Voting System – System Specification*, Version 1.3.1, Swiss Post Ltd., June 15, 2023
- [VerSpec] *Swiss Post Voting System – Verifier Specification*, Version 1.4.1, Swiss Post Ltd., June 16, 2023

As in earlier releases, special versions of these specification documents were given to the examination experts with all the changes highlighted, and summaries of the changes are given in corresponding revision charts and CHANGELOG.md files. Generally, the number of relevant changes in the specification documents is very moderate.

In the software code, the largest amount of changes deals with performance optimizations of different types. While these changes affect the core of all cryptographic computations in the whole system, their correctness can be checked quite easily in the code, because they all follow a small set of repeating implementation patterns.

In Figure 4, we give an overview of the GitLab commit histories of corresponding repositories. It shows that all components have been updated in the new release. It also shows that internal sub-versions are created regularly, approximately once every week. To conduct our analysis, we only looked at the latest versions released on June 15 and June 16. In the remaining of this addendum section, we summarize the findings of our analysis.


Current Repository
crypto-primitives

Changes

Select Branch to Compare...

JUNE RELEASE
merge release crypto-primitives 1.3.1.0
SwissPost-Bot • Jun 15, 2023

merge release crypto-primitives 1.3.1.0-RC8
SwissPost-Bot • Jun 13, 2023

merge release crypto-primitives 1.3.1.0-RC6
SwissPost-Bot • Jun 8, 2023

merge release crypto-primitives 1.3.0.5
SwissPost-Bot • Jun 6, 2023

merge release crypto-primitives 1.3.0.4
SwissPost-Bot • Jun 5, 2023

merge release crypto-primitives 1.3.0.3
SwissPost-Bot • Jun 5, 2023

merge release crypto-primitives 1.3.1.0-RC4
SwissPost-Bot • May 31, 2023

merge release crypto-primitives 1.3.0.2
SwissPost-Bot • May 31, 2023

merge release crypto-primitives 1.3.0.1
SwissPost-Bot • May 30, 2023

merge release crypto-primitives 1.3.1.0-RC1
SwissPost-Bot • May 16, 2023

Merge branch 'develop'
SwissPost-Bot • Apr 13, 2023

APRIL RELEASE
merge snapshot poms - crypto-primitives-1.3.0.0
SwissPost-Bot • Apr 13, 2023

Current Repository
crypto-primitives-ts

Changes

Select Branch to Compare...

JUNE RELEASE
merge release crypto-primitives-ts 1.3.1.0
SwissPost-Bot • Jun 15, 2023

merge release crypto-primitives-ts 1.3.1.0-RC8
SwissPost-Bot • Jun 13, 2023

merge release crypto-primitives-ts 1.3.1.0-RC6
SwissPost-Bot • Jun 8, 2023

merge release crypto-primitives-ts 1.3.1.0-RC4
SwissPost-Bot • May 31, 2023

merge release crypto-primitives-ts 1.3.0.2
SwissPost-Bot • May 30, 2023

merge release crypto-primitives-ts 1.3.1.0-RC1
SwissPost-Bot • May 16, 2023

merge release crypto-primitives-ts 1.3.0.1
SwissPost-Bot • Apr 24, 2023

APRIL RELEASE
merge release crypto-primitives-ts 1.3.0.0
SwissPost-Bot • Apr 13, 2023

Current Repository
crypto-primitives-domain

Changes

Select Branch to Compare...

JUNE RELEASE
merge release crypto-primitives-domain 1.3.1.0
SwissPost-Bot • Jun 15, 2023

merge release crypto-primitives-domain 1.3.1.0-RC8
SwissPost-Bot • Jun 13, 2023

merge release crypto-primitives-domain 1.3.0.5
SwissPost-Bot • Jun 13, 2023

merge release crypto-primitives-domain 1.3.1.0-RC6
SwissPost-Bot • Jun 8, 2023

merge release crypto-primitives-domain 1.3.1.0-RC4
SwissPost-Bot • May 31, 2023

merge release crypto-primitives-domain 1.3.0.4
SwissPost-Bot • May 31, 2023

merge release crypto-primitives-domain 1.3.0.3
SwissPost-Bot • May 30, 2023

merge release crypto-primitives-domain 1.3.0.2
SwissPost-Bot • May 23, 2023

merge release crypto-primitives-domain 1.3.1.0-RC1
SwissPost-Bot • May 16, 2023

merge release crypto-primitives-domain 1.3.0.1
SwissPost-Bot • May 9, 2023

APRIL RELEASE
merge release crypto-primitives-domain 1.3.0.0
SwissPost-Bot • Apr 13, 2023

Current Repository
e-voting

Changes

Select Branch to Compare...

JUNE RELEASE
merge release evoting 1.3.1.1
SwissPost-Bot • Jun 16, 2023

merge release evoting 1.3.1.0
SwissPost-Bot • Jun 15, 2023

merge release evoting 1.3.1.0-RC10
SwissPost-Bot • Jun 14, 2023

merge release evoting 1.3.1.0-RC9
SwissPost-Bot • Jun 14, 2023

merge release evoting 1.3.1.0-RC8
SwissPost-Bot • Jun 14, 2023

merge release evoting 1.3.1.0-RC7
SwissPost-Bot • Jun 9, 2023

merge release evoting 1.3.1.0-RC6
SwissPost-Bot • Jun 9, 2023

merge release evoting 1.3.1.0-RC5
SwissPost-Bot • Jun 6, 2023

merge release evoting 1.3.1.0-RC4
SwissPost-Bot • May 31, 2023

merge release evoting 1.3.1.0-RC3
SwissPost-Bot • May 26, 2023

merge release evoting 1.3.1.0-RC2
SwissPost-Bot • May 22, 2023

merge release evoting 1.3.1.0-RC1
SwissPost-Bot • May 17, 2023

APRIL RELEASE
merge release evoting 1.3.0.0
SwissPost-Bot • Apr 19, 2023

Current Repository
e-voting-libraries

Changes

Select Branch to Compare...

JUNE RELEASE
merge release e-voting-libraries 1.3.2.0
SwissPost-Bot • Jun 15, 2023

merge release e-voting-libraries 1.3.2.0-RC8
SwissPost-Bot • Jun 13, 2023

merge release e-voting-libraries 1.3.2.0-RC6
SwissPost-Bot • Jun 8, 2023

merge release e-voting-libraries 1.3.1.6
SwissPost-Bot • Jun 6, 2023

merge release e-voting-libraries 1.3.1.5
SwissPost-Bot • Jun 6, 2023

merge release e-voting-libraries 1.3.2.0-RC4
SwissPost-Bot • May 31, 2023

merge release e-voting-libraries 1.3.1.4
SwissPost-Bot • May 30, 2023

merge release e-voting-libraries 1.3.1.3
SwissPost-Bot • May 26, 2023

merge release e-voting-libraries 1.3.1.2
SwissPost-Bot • May 23, 2023

merge release e-voting-libraries 1.3.2.0-RC1
SwissPost-Bot • May 16, 2023

merge release e-voting-libraries 1.3.1.1
SwissPost-Bot • May 9, 2023

merge release e-voting-libraries 1.3.1.0
SwissPost-Bot • Apr 19, 2023

merge release e-voting-libraries 1.3.0.1
SwissPost-Bot • Apr 19, 2023

APRIL RELEASE
merge release e-voting-libraries 1.3.0.0
SwissPost-Bot • Apr 13, 2023

Current Repository
verifier

Changes

Select Branch to Compare...

JUNE RELEASE
merge release verifier 1.4.1.0
SwissPost-Bot • Jun 15, 2023

merge release verifier 1.4.1.0-RC8
SwissPost-Bot • Jun 14, 2023

merge release verifier 1.4.1.0-RC4
SwissPost-Bot • Jun 2, 2023

merge release verifier 1.4.1.0-RC1
SwissPost-Bot • May 17, 2023

APRIL RELEASE
merge release verifier 1.4.0.0
SwissPost-Bot • Apr 19, 2023

Figure 4: Commit histories of the GitLab projects since the April release.

A.2. Cryptographic Primitives

Only two types of changes were made to the specification document [CryptPrim]. The first type of changes affects the computation of recursive hash values in Algorithm 4.9 RecursiveHashToZq. To extend domain-separation, the group order q and the domain separation string `"RecursiveHash"` are now used as a prefix to the values included in the input vector $\mathbf{v}$. The code of the methods `recursiveHashToZq` in the files HashService.java and HashService.ts have been changed accordingly. Similarly, both calls of RecursiveHashToZq now include an additional domain-separation string (`"HashAndSquare"` for Algorithm 4.11 HashAndSquare and `"commitmentKey"` for Algorithms 8.6 GetVerifiableCommitmentKey). Other than making these calls consistent, we do not see the reason for these changes. Some explanations would have been helpful (according to the file CHANGELOG.md, other experts suggested this change). At least, we were able to verify that the implemented Java and TypeScript code correspond to the updated specification.

The purpose of the second type of changes is to improve the alignment between pairs of proof generation and proof verification algorithms, specifically between Algorithm 9.5 GenDecryptionProof and Algorithm 9.6 VerifyDecryption, and similarly between Algorithm 9.8 GenExponentiationProof and Algorithm 9.9 VerifyExponentiation. In each of the two cases, two lines of the verification algorithms have been merged into one line. We generally appreciate any sort of adjustments that improves the alignment between different algorithms. Such adjustments contribute to the overall readability of the specification documents and the code.

A.2.1. Optimizations

Two types of optimizations have been implemented in the new version. The first optimization comes from using Douglas Wikström's VMGJ library to allow fixed-base and product exponentiations. As documented in [HLG19, HL20], the potential performance optimization is approximately one order of magnitude, which is clearly something that should be exploited. Fixed-base exponentiation is initialized by calling the method `BigIntegersOptimizations::prepareFixedBaseOptimizations` for a given base and modulus. This creates a cache of precomputed values, which accelerates modular exponentiations for the specified base. We observed that such precomputations are conducted for the group generator g (corresponding computations for the election ElGamal public keys and for the prime number encodings of the voting options are invoked from the secure-data-manager component). We also observed that for the size of the precomputation table, the default value 2^{16} from Wikström's base class `FpowmTab` is taken unchanged. Given the large amount of available computational resources at the Swiss Post infrastructure, we believe that this value could be considerably larger. Unfortunately, no information is given to justify this choice. Also for product exponentiations, Wikström's library is used as such. After 30 days, the precomputation cache is cleared automatically (using the mechanism implemented in Google's guava library).

The second implemented optimization now allows parallel computations also for the exponentiation proof generation and verification. The usage of parallelization can be configured by the property `"enable.parallel.streams"` in the system's configuration file. Interestingly, at other places parallelization is always used, for example in the classes `HadamardArgumentService`, `ShuffleArgumentService`, or `ZeroArgumentService`. While we generally approve such optimizations, we would expect to always see the exact same implementation pattern. We can not judge whether configuring parallelization has a benefit (possibly for testing purposes?), but if the above propery exists in the system's configuration file, we would expect to consider it at each invocation.

A.2.2. Other Observations

We observed that the dependency to the third-party library bouncycastle has been updated from Version 1.72 to Version 1.73. While this seems like a normal update, we observed that the import statement for the Jacobi symbol computation has bee changed from `org.bouncycastle.pqc.math.linearalgebra.IntegerFunctions.jacobi` to

`org.bouncycastle.pqc.legacy.math.linearalgebra.IntegerFunctions.jacobi`,

i.e., with the term `"legacy"` in the package name. We do not know the reason for this change, but we generally recommend not to include any legacy code from third parties.

A.3. E-Voting

In the protocol specification document [SysSpec], we observed a few textual improvements and clarifications in Sections 3.5 and 4.2.2. These changes have no impact on the cryptographic protocol. All other changes in the document are related to the description of the voter authentication method introduced in the April release.

A.3.1. Voter Authentication

We have already given our comments about the redesigned authentication method in Subsection 3.1. One of our concerns was the usability problem caused from fixing the TOTP time interval to the default value of $T_X = 30$ seconds, which implied that the three invocations of the authentication procedure cannot be conducted in less than one minute. The updated description of this procedure and the new implementation address this problem.

In the modified voter authentication protocol, the voting client generates a nonce to allow multiple authentications within a given 30-seconds window. The adapted procedure is described in the updated Section 5.1, in the updated Algorithms 5.1 and 5.2 (GetAuthenticationChallenge and VerifyAuthenticationChallenge, respectively), and in the update of Table 14. According to Algorithm 5.1, the 256-bits value $\texttt{nonce}$ is selected at random by the voting client and transferred to the (untrusted) voting server, together with $\texttt{credentialID}_{\texttt{id}}$ (as before) and $\texttt{hhAuth}_{\texttt{id}}$ (as before, but now dependent on $\texttt{nonce}$). Based on this input and the current time $\texttt{TS}$, the voting server can then compute two candidate values $\texttt{hhAuth}'_{\texttt{id},1}$ (for the current time window) and $\texttt{hhAuth}'_{\texttt{id},0}$ (for the previous time window), and check if one of them corresponds to the transmitted authentication value $\texttt{hhAuth}_{\texttt{id}}$.

While this procedure seems to work as intended, it should no longer be called a protocol "*heavily inspired by*" the Time-based One-Time Password (TOTP) protocol, "*a widely-used authentication method with robust security proofs*", which "*deviates from RFC6238 in some details*", because adding a nonce as proposed may possibly create another kind of authentication method with completely different security properties. Generally, this kind of cryptographic engineering—taking an existing cryptographic standard and modify it to current needs—should be avoided.

In the light of our remarks in Subsection 3.1 about the introduced voter authentication procedure, we still do not see the exact purpose of the proposed solution. The attack scenarios that this method should prevent are still very unclear, especially since the voting server is an untrusted system component. Given the additional information about using the voter's birth date as a secondary authentication factor in the updated Section 4.1.4, we can accept this particular decision as sufficiently reasonable, but this does not imply the necessity of the proposed solution. Note that the problem of voting clients with out-of-sync system times remains in the current version of the June release.

A.3.2. Optimizations

In many parts of the e-voting component, we observed the introduction of parallelization by invoking the method `Stream::parallel` in corresponding Java stream pipelines, especially in the control-component and secure-data-manager (backend) sub-components. What is currently missing is a clear description of the underlying implementation strategy, because in the current version, parallel streams are used *sometimes*, i.e., without following any obvious pattern. Therefore, it would be good to define such a pattern based on some truly expensive operations, such as modular exponentiation or recursive hashing, and to apply it in a strict and comprehensive manner throughout the code. An extreme example of an obviously unnecessary invocation of parallel streams can be found in the class `GetVoterAuthenticationDataAlgorithm` (Lines 72-75), where `hAuth_id` invokes nothing but a getter method of the `VoterAuthenticationData` record:

```
List<String> hAuth = voterAuthenticationData.stream()
    .parallel()
    .map(VoterAuthenticationData::hAuth_id)
    .toList();
```

As already mentioned in Appendix A.2.1, we also observed that the secure-data-manager (backend) sub-component invokes precomputations for fixed-based exponentiations (in the configuration phase, as part of the `GenVerDatService::GenVerDat` method), both for the election public keys and the prime-number encodings of the voting options. Note that no other such invocations exist in any of the other sub-components. Without having further analyzed the potential for performance optimizations in other sub-components, we could imagine that for example the control-component sub-component could also benefit from such pre-computations, for example as part of the mix-net.

A.3.3. Other Observations

Some other modifications are listed on the CHANGELOG.md file of the e-voting component. Most of the the topics seem relatively uncritical, but due to the very short 2-weeks evaluation period, we were unable to further look into them as we would have wished. For reasons of completeness, we list them below:

- Reduced the amount of information transferred when a voter logs in but the vote was already confirmed.
- Improved performance by storing individual entries of the CMTable and pCC allow list in the database instead of the table as a whole.
- Added the chunk-wise processing of large data items to avoid time-outs and out-of-memory issues.
- Introduced idempotent processing of requests in the voting server to increase robustness.
- Reduced the amount of transferred data during the export and import by filtering irrelevant data items.
- Improved the Secure Data Manager's reliability by disabling certain actions when preconditions are not met.

- Increased the minimum length of electoral board passwords to 24 characters.
- Improved method transactions in the voting server.
- Fixed a minor issue related to the zip-slip vulnerability whereby empty directories could be created by a malicious ZIP file.

A.4. Verifier

Relative to the updated verifier specification document [VerSpec], the CHANGELOG.md file indicates only some minor changes. The most notable modification seems to result from merging four different verifications, namely

- VerifySignatureSetupComponentVerificationData,
  ⇒ now called Algorithm 3.1 (formerly Verification 2.05)
- VerifySignatureControlComponentCodeShares,
  ⇒ now called Algorithm 3.2 (formerly Verification 2.06)
- VerifyEncryptedPCCExponentiationProofs,
  ⇒ now included as part of Algorithm 3.3 (formerly Verification 5.21)
- VerifyEncryptedCKExponentiationProofs,
  ⇒ now included as part of Algorithm 3.4 (formerly Verification 5.22)

into one verification named VerifySignatureVerificationDataAndCodeProofs (called Verification 5.21). However, on closer inspection, i.e., by comparing [VerSpec, Version 1.4.0] and [VerSpec, Version 1.4.1] manually, we learned that the changes run deeper than initially reported, i.e., the nature of the changes have turned out to be more complex than *just* merging some verification checks.

In the updated specification document, the renaming of Verifications 2.05 and 2.06 into Algorithm 3.1 and 3.2, respectively, was made with the intention of optimizing performance during large election events, particularly by reducing the need for deserializing the same data multiple times (see comment given on top of [VerSpec, Page 18]). These algorithms are now called by the newly introduced Verification 5.21.

However, there were two verifications formerly called Verification 5.21 and 5.22 (see overview given above), which have now disolved into the two Algorithms 3.3 (formerly called Algorithm 3.1) and Algorithm 3.4 (formerly called Algorithm 3.2) from Subsection 3.6. Since the merged algorithms work differently than the former ones, they need to be checked carefully from scratch. Therefore, what is announced as a simple merge of some verifications has turned out to be the result of an obscure and confusing renaming and refactoring procedure that involves multiple algorithms.

These adjustments introduce a question about the essence of the specification: is it merely reflecting implementation-specific tuning? Does the specification risk becoming an implementation document? Would this be a requirement for any independent implementation of a verifier for the Swiss Post system? It is crucial to separate specifications from concrete implementations, and this blurring of boundaries starts to be concerning.

Regrettably, the provided diff-document for the verifier specification falls short when it comes to effectively distinguishing between different versions of the specification. As evidenced in Figure 5, the user can be misled into thinking that the algorithm was previously incorrect and never functional.

Verification 2.05 VerifySignatureSetupComponentTallyData

Context:

The trust store containing the system's certificates

Input:

The message ~~SetupComponentVerificationData~~ SetupComponentTallyData from table 3

The signature $s \in \mathcal{B}^*$

Operation:

1: ~~VerifySignature(“sdm_config”, $(\{\mathtt{vc}_{\mathtt{id}}, \mathtt{K}_{\mathtt{id}}, \mathtt{c}_{\mathtt{pCC,id}}, \mathtt{c}_{\mathtt{ck,id}}\}_{id=0}^{\mathtt{N_E}-1}, \mathtt{L}_{\mathtt{pCC}})$, (“verification data”, ee, vcs), s)~~ VerifySignature(“sdm_config”, (vc, $\mathbf{K}$), (“tally data”, ee

▷ See crypto primitives specification

Output:

$\top$ if the verification succeeds, $\bot$ otherwise.

Figure 5: Misleading diffs for verifications.

A similar concern arises with the newly introduced algorithms (as shown in Figure 6). Swiss Post is advised to refrain from including such confusing or misleading content.

Algorithm 3.1 VerifySignatureSetupComponentVerificationData

Context:

The trust store containing the system's certificates

Input:

The message ~~SetupComponentTallyData~~ SetupComponentVerificationData from table 3

The signature ~~$s \in \mathcal{B}^*$~~ $\mathtt{s}_{\mathtt{SCVD}} \in \mathcal{B}^*$

Operation:

1: ~~VerifySignature(“sdm_config”, (vc, $\mathbf{K}$), (“tally data”, ee, vcs), s)~~
VerifySignature(“sdm_config”,
$(\{\mathtt{vc}_{\mathtt{id}}, \mathtt{K}_{\mathtt{id}}, \mathtt{c}_{\mathtt{pCC,id}}, \mathtt{c}_{\mathtt{ck,id}}\}_{id=0}^{\mathtt{N_E}-1}, \mathtt{L}_{\mathtt{pCC}})$,
(“verification data”, ee, vcs), $\mathtt{s}_{\mathtt{SCVD}}$) ▷ See crypto primitives specification

Output:

$\top$ if the verification succeeds, $\bot$ otherwise.

Figure 6: Misleading diffs for algorithms.

These issues underline the increasing complexity of the verifier specification at hand, indicating that it is still under active development. Consequently, it is not really feasible to examine the diffs alone nor by any pure analytical means. The specification, in its entirety, must be re-examined to determine whether an independent verifier implementation is capable of correctly supporting an election outcome of the specified e-voting system.

References

[HL20] R. Haenni and P. Locher. Performance of shuffling: Taking it to the limits. In M. Bernhard, L. J. Camp A. Bracciali and, S. Matsuo, A. Maurushat, P. B. Rønne, and M. Sala, editors, *Voting'20, FC 2020 International Workshops*, LNCS 12063, pages 369–385, Kota Kinabalu, Malaysia, 2020.

[HLG19] R. Haenni, P. Locher, and N. Gailly. Improving the performance of cryptographic voting protocols. In A. Bracciali, J. Clark, F. Pintore, P. B. Rønne, and M. Sala, editors, *Voting'19, FC 2019 International Workshops*, LNCS 11599, pages 272–288, Frigate Bay, St. Kitts and Nevis, 2019.

[MvV96] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, USA, 1996.