SCRT Information Security

Examination of the Swiss Internet Voting System

Audit scope 2a: Development process

22.03.2022 / v1.0FINAL

Work performed for:

Swiss Federal Chancellery
Political Rights Section
Federal Palace West Wing
3003 Bern

Contact information

SCRT SA	T: +41 21 802 64 01
Rue du Sablon 4	E: info@scrt.ch
1110 Morges	
Switzerland	

Authors and contributors

Antonio Fontes	Lead examiner
Stéphane Adamiste	Reviewer
Sergio Alves Domingues	Reviewer

Management summary

Scope and objective of the examination

The objective of this examination was to assess whether Swiss Post's development process put in place for the development and maintenance of its e-voting system satisfies a subset of requirements (audit scope 2a) set forth by the Federal Chancellery's ordinance on e-voting.

Methodology

The examiners looked for evidence of effort to satisfy said requirements, whether the approach chosen by Swiss Post effectively aligns with what the examiners identified as good practice, and whether the proposition offers a level of assurance that users and customers could reasonably expect from the development process of the e-voting system.

The examination was performed during the second half of 2021 and data was obtained from written evidence (i.e., formal procedures, specifications, documentation, reports and other artefacts automatically generated by tools or programs) and oral statements (live presentations and interviews with Swiss Post e-voting teams). The total workload for this examination, including the edition of this report, is 120 hours.

Results

Six issues (findings) were identified and reported, and eleven recommendations for improvement were formulated.

The findings reflect that a large amount of effort has been deployed by Swiss Post to ensure that its development process satisfies the Federal Chancellery's requirements and responds to high quality standards. Still, additional effort remains necessary to fully integrate what the community currently recognises as the reference, or baseline, in terms of secure software engineering practices. In particular:

- Improving the integration of secure systems engineering methods (Finding F-01)
- Improving the definition and assignment of security responsibilities (Finding F-02)
- Improving protection from third-party component risk (Finding F-03)
- Improving and centralizing the security documentation (Finding F-04)
- Improving quality control on security attestation tools and methods (Finding F-05)
- Increasing security attestation during earlier stages of the development process (Finding F-06)

All findings were marked with the severity level 'moderate'. This is attributed to the fact that each finding reflected either an incomplete satisfaction of a requirement or the presence of mitigating factors, or compensating measures, which reduced the severity of the finding.

Final note

Assuming the recommendations (or equivalent measures) will be implemented within reasonable delay, the examiners consider the outcome of the examination to be generally positive.

The examiners conclude this summary by thanking Swiss Post, and more particularly all those who have been personally involved, for their cooperation and for the transparency demonstrated throughout the entire duration of the examination.

Table of content

1 Context 6
2 Methodology 7
2.1 General process 7
2.2 Collection of evidence 7
2.3 Findings 8
2.4 Severity of findings 8
2.5 Assumptions 8
3 Criteria 10
3.1 Federal Chancellery requirements 10
3.2 Additional criteria: secure systems development lifecycle 13
3.3 Additional criteria: inclusion of third-party components 15
4 Findings 16
4.1 F-01 Insufficient integration of security in the software development lifecycle 16
4.2 F-02 Conflicting/ambiguous attribution of security responsibilities 19
4.3 F-03 Insufficient protection measures against malicious third-party components 22
4.4 F-04 Insufficient security documentation 25
4.5 F-05 Insufficient quality control over security attestation operations 27
4.6 F-06 Insufficient security testing and attestation 28
4.7 Summary of findings 30
4.8 Summary of recommendations 31
5 References 32
6 Appendixes 36
6.1 Observations 36
6.2 Audit scope criteria 48
6.3 Interview sessions log 49
6.4 Development process 49
6.5 Reference evidence documents 51
6.6 Request for additional information on risk management 54
6.7 Interview plan - development process (additional criteria) + developers 55
6.8 Interview plan - testing and quality requirements 56
6.9 Interview plan - quality assurance requirements 57

1 Context

1. Electronic voting (hereafter referred to as: e-voting) was introduced in Switzerland through multiple pilot schemes from 2004 onwards. A total of 15 cantons made e-voting possible in over 300 trials, until early 2019. Two implementations were available: the system made by the canton of Geneva and the system operated by the Swiss Post (initially developed by Scytl). In June 2019, the canton of Geneva announced the withdrawal of its e-voting system with immediate effect. It was followed in July of the same year by the announcement by Swiss Post of the withdrawal of its e-voting system from operation to focus on improving the solution. Since then, e-voting is not possible in Switzerland.

2. In June 2019, the Swiss Federal Chancellery (hereafter: Federal Chancellery) was commissioned by the Federal Council to redesign a new trial phase, using “e-voting systems, which are fully verifiable” [1]. This redesign of the trial phase focuses on four objectives:

1. Further development of the e-voting systems
2. Effective controls and monitoring
3. Increased transparency and trust
4. Stronger connection with the scientific community

3. A taskforce was set up to make proposals for the future of internet voting. To that end, the Federal Chancellery invited experts from academia and industry to engage in a broad dialogue on internet voting in Switzerland. After this dialog, the Federal Chancellery and the cantons published a final report on the redesign and relaunch of internet voting trials, with a catalogue of measures [2].

4. The Federal Council took note of the final report and commissioned the Federal Chancellery to amend the legal bases of the Confederation. In April 2021, the Federal Council opened a consultation procedure on the amendment to the legal bases, which was drafted by the Federal Chancellery. A consultation procedure for the redesign of the e-voting trials was initiated in April 2021 by the Federal Council. The redesign includes both a partial revision of the Ordinance on Political Rights (PoRo) [3] and a complete revision of the Federal Chancellery Ordinance on Electronic Voting (“VEleS”, or “OEV”) [4]. The OEV specifies, among others, the requirements for authorising electronic voting, including the technical and administrative controls for approving an e-voting system[1].

5. The Federal Chancellery issued an audit concept for the examination of Swiss internet voting systems [5] defining the foundations for assessing the compliance of electronic voting systems with the draft OEV and its annex [6], as per chapter 26 of the annex of the draft OEV, and for obtaining recommendations for improvements.

[1] The criteria for the examination reported in this document is built on a subset of these controls.

6. SCRT was mandated by the Federal Chancellery to assess the compliance of the Swiss Post's revamped e-voting system against some of the requirements of the draft OEV. The present report focusses on the examination of the perimeter defined as follows in the audit concept: *Scope 2a - Development process*.

2 Methodology

2.1 General process

7. The examination was based on SCRT's information systems audit methodology. The process specifies four-phases depicted in the figure below:

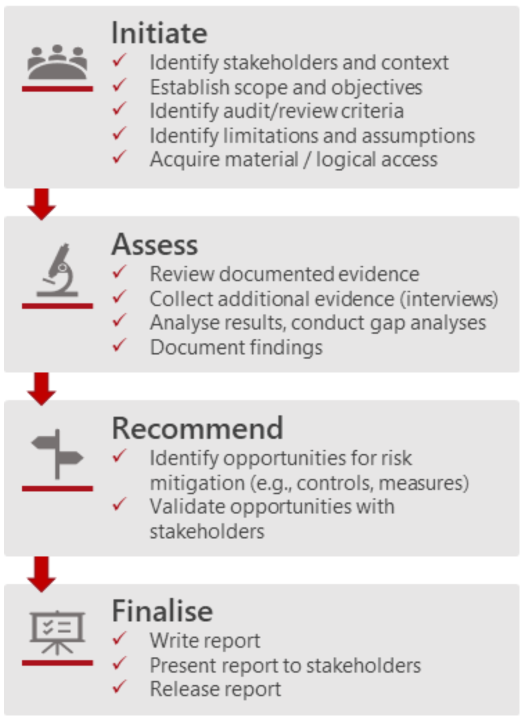

Initiate
✓ Identify stakeholders and context
✓ Establish scope and objectives
✓ Identify audit/review criteria
✓ Identify limitations and assumptions
✓ Acquire material / logical access
Assess
✓ Review documented evidence
✓ Collect additional evidence (interviews)
✓ Analyse results, conduct gap analyses
✓ Document findings
Recommend
✓ Identify opportunities for risk mitigation (e.g., controls, measures)
✓ Validate opportunities with stakeholders
Finalise
✓ Write report
✓ Present report to stakeholders
✓ Release report

Figure 1 - Process

2.2 Collection of evidence

8. As a general principle, the examiners aimed at acquiring two types of evidence for each requirement. Types of evidence included documents (e.g., documentation, test reports, written instructions, etc.), statements (e.g., obtained during plenary sessions or during interviews), and demonstrations (e.g., tools, scripts, configurations or process material shown during interviews).

9. Part of the examination included reviewing documents classified as confidential by Swiss Post and thus not released to the public. Motives for not disclosing these documents to the public included either or both the a) preservation of the confidentiality of business processes deployed at the organisation level and which may confer Swiss Post a competitive advantage

on other actors, and b) the preservation of confidentiality of operational data (e.g., risk control, infrastructure operations, etc.). Swiss Post confirmed to the examiners that these documents remain accessible to the Cantons.

10. Unless specified otherwise, written evidence collected and reviewed during the examination is referred in the bibliography of this report, with public links whenever possible. Some sources, which were not made physically available to the examiners but shown on-screen during live interviews with the examinees, are cited without reference.

11. Swiss Post provided the examiners an internal document mapping each Federal Chancellery requirement with one or more corresponding documented evidences [7].

2.3 Findings

12. The examiners raised a finding when evidence provided by the examinee did not provide satisfying assurance that the requirement is met (implicit miss) or when evidence provided explicitly indicates that the requirement is not or partially satisfied (explicit miss).

2.4 Severity of findings

13. The examiners specified three severity levels, as follows:

» *High severity* - The finding identifies a failure to produce evidence of satisfying a requirement.
» *Moderate severity* - The finding identifies a partial failure to produce evidence of satisfying a requirement.
» *Low severity* - The finding identifies a notable opportunity for improvement or optimisation.

14. Readers should note that the severity indicated in this report only reflects the opinion of the examiners and could be subject to re-evaluation by relevant parties.

2.5 Assumptions

2.5.1 Trustworthiness of statements

15. The examiners assume that the examinees were completely honest and transparent when providing answers to the examiners' assessment questions. Although several proofs of testing were shown, no observation of the actual implementation of the OEV's requirements within the e-voting system was carried out to verify the accuracy of the examinees' statements.

2.5.2 Trustworthiness of security measures

16. The examiners assume that the security measures described in the documents provided as evidence in the context of the present examination are implemented and are effective. No verification of the actual implementation effectiveness of the OEV's requirements within the

e-voting system (e.g., security testing, vulnerability assessment, penetration test, etc.) was carried out within the scope of this examination to verify the accuracy of the statements made in the security documents.

2.5.3 Segregation of the development environment

17. The examiners assume that the examinee has successfully segregated the e-voting development environment from other environments, including other environments related to the e-voting system. In this context, segregation is defined as follows:

» Systems, service accounts and users with permissions to approve changes to the source code, or any of its embedded dependencies, cannot tamper nor access further e-voting environments (e.g., release, production)[2,3].
» Systems, service accounts and users with any permission into the development environment are only granted permissions (e.g., read, create, modify, delete, etc.) strictly required to accomplish their duties and on resources over which their duties are expected to be carried out.
» Systems, service accounts and users that have not explicitly been identified and authorised as contributors to the development environment do not have access to the development environment, including in particular, its code and configuration repository (configuration management system), and its build systems.

2.5.4 Boundaries to the development process

18. The examiners considered, as parts of the "development process", all steps necessary to produce the artifacts required to satisfy a change request to the e-voting system, including tools, people, and methods.

The following were considered as part of the development process:

» Translating a change request into functional and technical requirements,
» Establishing a design or architecture change proposal to satisfy the change request,
» Implementing the proposal (e.g., identifying, selecting and importing/including third party components, writing new source code or modifying existing source code, etc.),
» Converting or packaging the resulting work into their final state (e.g., builds, binaries, components, documents) and making them available to subsequent operations (e.g., release to third parties, release to production, etc.).

[2] Of course, this excludes situations in which authorised members of the development environment access the e-voting production systems strictly as end users to exercise their voter rights.

[3] The examiners noted that developers can access the development environment remotely and from untrusted locations using a VPN connection. The assumption of segregation typically operates on the hypothesis that this segregation has been enforced at the infrastructure level (i.e., only authorised users and devices can approve change to the source code).

The following were not considered as part of the development process:

- Any operation, individual or system involved prior to a change request being entered into the master backlog of the e-voting system.
- Any operation, individual or system involved after the requested artefacts have been released and assembled in a final state (e.g., documents, binaries, bytecode, packages, extensions, modules, etc.) and made available to further operations (e.g., deployment for testing, analysis or acceptance outside the development environment, deployment into production, etc.).

3 Criteria

This section summarizes the requirements on which the examination was performed.

3.1 Federal Chancellery requirements

19. The following tables[4] enumerate the 17 requirements set forth in the criteria for scope 2A ("assess the development process") of the concept for examining Swiss internet voting systems [5][5].

3.1.1 Development process and lifecycle requirements

Key	Requirement
24.1.1	A life cycle model is defined. The life cycle model: » is used for the development and maintenance of the software (a); » provides for the necessary controls during the development and maintenance of the software (b); » is documented (c).
24.1.2	A list must be made of the development tools used and configuration options chosen for the use of each development tool.
24.1.3	The documentation for the development tools includes: » a definition of the development tool (a); » a description of all conventions and directives used in the implementation of the development tool (b); » a clear description of the significance of all configuration options for using the development tool (c).
24.1.4	The implementation standards to be applied must be specified.

Table 1 - E-voting requirements: lifecycle

[4] In this section, the requirements are regrouped per topic. The grouping reflects the examiners' interpretation of the requirements and may not reflect that of the Swiss Federal Chancellery.

[5] The requirements were grouped by topic by the examiners. The groups not necessarily reflect the Federal Chancellery's vision.

3.1.2 Software security documentation requirements

Key	Requirement
24.1.20	Software development security documentation includes: » a description of the physical, procedural, personnel, and other security measures necessary to protect and ensure the integrity of the design and implementation of the software in its development environment (a); » evidence that the security measures provide the necessary level of protection to preserve the integrity of the software (b).

Table 2 - E-voting requirements: software security documentation

3.1.3 Quality assurance requirements

Key	Requirement
24.5	Regular and objective checks are carried out to ensure that the processes carried out and the associated work products comply with the description of the processes, standards and procedures to be implemented (a). Deviations are followed up until they are corrected (b).

Table 3 - E-voting requirements: quality assurance

3.1.4 Configuration management system requirements

Key	Requirement
24.1.14	The software is provided with a unique identification.
24.1.15	The configuration management documentation includes: » a description of how configuration items are identified (a); » a configuration management plan describing how the configuration management system will be used in the development of the software and the procedures that will be followed for the adoption of changes or new elements (b); » evidence that the procedures for adoption provide for adequate review of changes for all configuration items (c).
24.1.16	The configuration management system: » uniquely identifies all configuration items (a); » provides automated measures to ensure that only authorised changes are made to configuration items (b); » supports the development of the software through automated procedures (c); » ensures that the person responsible for accepting the configuration item is not the same person who developed it (d); » identifies the configuration items that make up the security functions (e); » supports verification of all changes to the software using automated procedures, including logging of the author and the date and time of the change (f); » provides an automated method for identifying any configuration items that are affected by a change to a particular configuration item (g); » can identify the version of the source code on the basis of which the software is generated (h).

24.1.17	All configuration items are inventoried in the configuration management system.
24.1.18	The configuration management system is used in accordance with the configuration management plan.
24.1.19	A configuration list is created that contains the following items: » the software, » evidence of the checks required to ensure security compliance, » the parts that make up the software, » the source code, » reports on security flaws and on the status of their correction (a). For each element relevant to security functions, the developer is named (b). Each element is uniquely identified (c).

Table 4 - E-voting requirements: configuration management system

3.1.5 Testing requirements

Key	Requirement
17.1	The functions relevant to the security of the system (security functions) are tested and the tests are documented with test plans and expected and actual test results. (a) The test plan (b): » specifies the tests to be performed; » describes the scenarios for each test, including any dependencies on the results of other tests. The expected results must show the results that are expected if the test is successfully executed. (c) The actual results must be consistent with expected results. (d)
17.2	An analysis must be made of the test coverage. This includes evidence that: » the tests defined in the test documentation match the functional specifications of the interfaces (a); » all interfaces have been fully tested (b).
17.3	An analysis must be made of the depth of testing. This includes evidence that: » the tests defined in the test documentation match the subsystems related to security functions and modules that play a role in ensuring security (a); » all subsystems related to the security functions mentioned in the specifications have been tested (b); » all modules that play a role in ensuring security have been tested (c).
25.13.3	The integration tests cover all modules.
25.13.4	The software tests cover all modules.

Table 5 - E-voting requirements: testing

3.1.6 Transparency requirement

Key	Requirement
8.12	Known flaws and the need for action associated with them are communicated transparently.

Table 6 - E-voting requirements: transparency

3.1.7 Systematic correction of flaws requirements

Key	Requirement
24.4.1	Processes are defined for the correction of flaws. The processes include: » documentation of specific aspects, in particular with regard to the traceability of flaws for all versions of the software, and of the methods used to ensure that system users have information on flaws, corrections and possible corrective actions (a); » the obligation to describe the nature and impact of all security flaws, information on the status of work to find a solution and the corrective measures adopted (b); » a description of how system users can make reports and enquiries about suspected flaws in the software known to the software developers (c); » a procedure requiring a timely response and automatic dispatch of security flaw reports and appropriate corrective actions to registered system users who may be affected by the flaw (d).
24.4.2	A process is defined for handling reported flaws (a). This process ensures that all reported and confirmed flaws are corrected and that the procedures for correction are communicated to system users (b). It provides for arrangements to ensure that the correction of security flaws does not give rise to new security flaws (c).
24.4.3	Policies must be defined for the reporting and correction of flaws. These include: » instructions on how system users can report suspected security flaws to the developer (a); » instructions on how system users can register with the developer to receive reports of security flaws and the corrections (b); » details of specific contact points for all reports and inquiries on security issues concerning the software (c).

Table 7 - E-voting requirements: systematic correction of flaws

3.2 Additional criteria: secure systems development lifecycle

20. A central part of the work consisted in evaluating the development process put in place by Swiss Post for its e-voting system. To limit potentially ambiguous interpretations of what could qualify as the OEV requirement "life cycle model, which provides for the necessary controls during maintenance and development of the software" [6, p. 33], the examiners derived an interpretation of "necessary controls" based on the following references:

- Security software lifecycle requirements and assessment procedures (v1.0), PCI [8],
- Application software security controls, Critical security controls (v.8), CIS [9],
- Fundamental practices for secure software development (third edition), SAFECode [10].
- SAMM - Software assurance maturity model assessment tool (v1.5), OWASP [11].

21. The characterisation of "necessary controls" is summarised as follows:

Organisational measures:

- The organisation has appointed a security champion within each development team, which, among others, acts as a security liaison and leader between the development team and the organisation's security structures,
- The organisation established interfaces between the development team and the organisation's information security structures and with external security advisor(s),
- The organisation established interfaces with its incident response structure,
- The organisation ensures that personnel involved in the design, construction or attestation of the system attended role-based security awareness and training.

Operational enablement measures:

- A catalogue of threats, with their respective controls or countermeasures, and status (e.g., mitigated, not mitigated, etc.), is documented and maintained,
- Security requirements are documented,
- Secure design principles or secure architecture baseline requirements are documented and integrated in the development process,
- Changes to the system are subject to a threat assessment (e.g., threat modelling, abuse cases, attacker stories, etc.) aimed among others at identifying potential and relevant threats and identifying appropriate countermeasures,
- Coding guidelines, or equivalent, are documented. In particular, they propose standardised responses to well-known causes of risk and error (e.g., input canonicalization and validation, output encoding, command interpreter query parameterisation, filesystem access, database access, protected storage of sensitive data or secrets, etc.),
- High-risk code is identified as such and subject to extended review (e.g., manual review or testing),
- Vulnerability management is integrated and performed throughout the entire development process with the support of adequate tools and processes,
- Source code, including all relevant adjacent artifacts, released to customers is centralised, versioned, and protected from unauthorised access.

Attestation measures:

- Change requests are subject to standardised or routine security verification (e.g., security checklist in definition of ready).
- New code is tested for well-known vulnerabilities and errors (aka, source code review, static analysis, etc.) as well as existing code (to mitigate regressions) prior to release.
- Third-party components are vetted against well-known threats prior to being integrated into the system.

- Third-party components are inventoried and monitored for known issues or vulnerabilities.
- The runtime is tested for well-known vulnerabilities and errors (e.g., dynamic/runtime application security testing) prior to release.
- The security of the final system, both in its entirety and its individual high-risk components, is regularly tested by independent actors through adequate methods (e.g., external penetration testing, bug bounty, 3rd party expert review, etc.).
- Releases and all associated artifacts are certified, and their authenticity can be independently verified (e.g., code or binary signing, etc.),

3.3 Additional criteria: inclusion of third-party components

22. Due to the extensive use of this-party components in the e-voting system, the examiners also derived an interpretation of "necessary controls" for the use and inclusion of third-party components as part of the software engineering process. The following reference was used to derive requirements:

- Managing security risks inherent in the use of third-party component, SAFECode [12].

23. The characterisation of "necessary controls", in the context of third-party components, is summarised as follows:

Organisational measures:

- The organisation conducts a standardised risk assessment prior to integrating a third-party component into the system.

Operational enablement measures:

- Threats derived from the use and inclusion of third-party components in the software application are identified and documented, and adequate countermeasures or controls implemented wherever relevant and/or necessary.

Attestation measures:

- Third-party components are tested for known vulnerabilities or malicious activity, both prior to their inclusion in the system and after (monitoring).

4 Findings

4.1 F-01 Insufficient integration of security in the software development lifecycle

Federal Chancellery requirement

Key	Requirement
24.1.1	A life cycle model is defined. The life cycle model: » is used for the development and maintenance of the software (a); » **provides for the necessary controls during the development and maintenance of the software (b);** » is documented (c).

Severity

24. MODERATE.

Rationale

25. Third-party material used to evaluate the development process of the Swiss Post e-voting program reflects a common paradigm: security should be addressed at all stages of the development process [8, p. 24], [9, p. 53], [10, p. 6]. Our examination indicates that Swiss Post started to implement this paradigm in the e-voting system development process (see appendix: Observations) but controls appear to be yet missing, in particular with artefacts that do not directly associate to the voting protocol. While the protocol itself appears to have been subjected to accrued scrutiny during its design phase (threat modelling, security analysis, formal proof, including all associated documentation), the examiners could not observe the same effort deployed on other parts or components of the voting system.

26. The examiners noted that quantifying the relationship, or impact, of an incomplete integration of security in the software development process is a difficult endeavour paved with uncertainties. A good illustration of this problem is the threat modelling activity: it is very difficult to estimate accurately the likelihood of significant flaws or vulnerabilities running undetected in the e-voting system after a development team fails to perform threat modelling regularly.

One can easily claim, however, that failing to address security at all opportunities given throughout the development process induces two consequences:

» *Increased burden on remaining controls*: as some controls are skipped, the dependency towards the remaining controls (e.g., automated code scanning. bug bounty, penetration testing, software composition analysis) increases and these controls end up carrying a heavier burden. Although this could theoretically be compensated by increasing the extent and scope of the remaining controls (see finding F-06), all controls are not created equal towards each category of vulnerability (e.g., a penetration test may be less adequate to reveal cryptographic flaws than a formal protocol review).

» *Late discovery of flaws and increased costs*: When the security strategy emphasises controls that occur at later stages of the development lifecycle (e.g., security testing, penetration testing, bug bounty), the discovery of certain types of flaws, such as design flaws or poorly designed processes, typically occurs post-implementation, or worse, post-release. The late discovery of flaws that were introduced at early stages of the development process may induce both increased and unnecessary remediation costs[6], and unnecessarily complexify the prioritization of effort (i.e., deciding what should be done 'next' in a context of limited resources and bandwidth).

27. The examiners noted that the Federal Chancellery provided Swiss Post with a reference threat model. The catalogue includes 37 threats or attack scenarios, which establish a baseline threat model for all components of the e-voting system [6, p. 24]. Swiss Post demonstrated the integration of this threat catalogue into a formal validation instrument[7] used to track responsible parties and their mitigation efforts deployed to address each threat scenario.

Still, the examiners identified two potential weaknesses with the approach:

» *Strong delegation of security testing to third parties*: Swiss Post heavily relies on public scrutiny through publicly accessible source code, quality reviews of the documentation and the bug bounty program to gather insight on the security of the voting system.

» *Generalised response to threats*: the threat catalogue is not evaluated for each component or critical feature but rather towards the e-voting system, as a whole. The examiners claim that this approach may prevent the identification of threats that only affect specific categories of components.

28. The examiners noted that although the architects and developers have not been formally trained for secure software engineering, they are encouraged to regularly attend internal meetings that discuss systems security and to attend public cybersecurity conferences.

29. The examiners noted that submissions to the configuration management system (pull requests) implement a four-eyes principle by requiring the approval of a second individual. Consequently: a developer cannot modify the source code of the e-voting system without the change operation being authorised by another developer or architect. Assuming the scenario of a compromised developer account (threat entries 13.7, 13.10, 13.31, 13.37) [6, p. 24], this

[6] The reader should note that the authors emphasized the distinction between flaws (inadequate design) and vulnerabilities (implementation failures): while the late discovery of vulnerabilities tends to induce lower remediation costs (e.g., topical modification of the source code or a configuration setting), failing to identify flaws before release can induce design changes, which would require a complete re-validation of the voting system or one of its components.

[7] The document (*Bruttorisiken_***.docx*) is not publicly referenced as it was still classified as highly confidential at time of the examination. Still, the examiners were given access to the document during the examination.

control constitutes an essential line of defence against attempts to insert malicious code in the system.

30. The examiners noted the deployment of an extended set of measures to facilitate public scrutiny, which may significantly enhance the overall potential to detect flaws or vulnerabilities. These measures include:

- The disclosure of security documentation to the public,
- The disclosure of the source code of essential components to the public,
- A public penetration testing program,
- A bug bounty program.

Recommendation

31. [R.01] The examiners encourage Swiss Post to formally integrate publicly available guidance, references or standards on secure development processes or lifecycles into its existing e-voting software development process. This integration should allow Swiss Post to:

- Identity and select processes and activities typically required to establish a secure development lifecycle,
- Share information and communicate on its adhesion to such guidance and its level of integration,
- Benchmark its own activities and processes internally on a regular basis.

This effort will (should) primarily allow Swiss Post to better document its own adoption of secure development practices to the public (see finding: F-04). Additionally, this effort will (should) allow Swiss Post to demonstrate to the public that it has already implemented a significant set of measures and controls commonly recommended in secure development lifecycle guidance.

32. [R.02] The examiners encourage Swiss Post to organise role-based security training to all personnel directly involved in the development or maintenance of the e-voting system. Participants should be made aware of well-known threats and attacks to which the development and maintenance operation, the deliverables and the overall infrastructure may likely be exposed, in addition to threats provided by the Federal Chancellery [6, p. 24].

Additionally:

- Architects should be trained on methods to identify, anticipate, and address well-known software threats prior to starting their implementation (e.g., secure software architecture design, secure software engineering principles, threat modelling),
- Developers should be trained on methods to avoid well-known coding vulnerabilities and errors (e.g., secure coding, defensive programming, security code review, etc.), and to identify code areas that should be subject to formal security review,
- Project managers and product owners should be trained on more general topics such as information security and risk management, understanding security and privacy

requirements, and how security is typically integrated in agile development processes[8] (e.g., one-time operations, every-sprint operations, regular operations, etc.).

33. [R.03] The examiners encourage Swiss Post to infuse threat modelling in its e-voting system development and maintenance process. In particular, each development team, including the product owner, should have access to an up-to-date list of threats to their respective component or project and be given adequate resources to conduct regular assessments on these threats. Alternatively, or additionally, the examiners recommend integrating abuse cases in the e-voting system's development process, typically through the use of attacker stories[9].

34. [R.04] The examiners encourage Swiss Post to identify and document a minimal or baseline set of security principles or standard responses to known and probable security issues at various levels (e.g., design/architecture, coding, testing, etc.). These could typically take the form of configuration/policy sets, banned functions/APIs, design principles, coding standards, reference implementations, test plans, etc.

4.2 F-02 Conflicting/ambiguous attribution of security responsibilities

Federal Chancellery requirement

Key	Requirement
24.1.1	A life cycle model is defined. The life cycle model: » is used for the development and maintenance of the software (a); » **provides for the necessary controls during the development and maintenance of the software (b);** » is documented (c).

Severity

35. MODERATE.

Rationale

36. Based on their observations (see Appendix: Observations), the examiners claim that assigning potentially conflicting priorities to the architect role increases her/his exposure to conflicts of interest. In software engineering operations, this could translate into architecture or design decisions being taken against fundamental systems security engineering principles.

[8] Recommended read: "Agile, yes, but secure?" (Falk, Andreas, 2015), last available at: https://owasp.org/www-pdf-archive/Owasp_stuttgart_agile_secure_20150803.pdf

[9] Attacker stories: negative use cases triggered by an attacker or other source of threat, whose successful resolution in the issue tracking system relies on the fact that the use case is or has become unfeasible or impractical. Additional information at: https://cheatsheetseries.owasp.org/cheatsheets/Abuse_Case_Cheat_Sheet.html

Examples could include postponing or precipitating the refactoring of code involved in critical operations, integrating a third-party library or API without a thorough vetting process, failing to address a need to standardise or formalise a solution to a frequent problem, etc.

37. In addition to conflicting priorities, the examiners notice their dilution across multiple individuals (multiple architects), which may unnecessarily make the measure of performance more difficult and, in the examiners' opinion, reduce individual responsibility.

38. Multiple research and guidance materials [13], [14] suggest that the security champion should be typically "conversant in the software development tools and methodologies used in the team, in addition to specific secure software development and deployment skills" and is proficient in "threat modelling and incident response". She or he also provides "the glue between the organisation's security structures and development". The security champion should be both someone who "wants to upgrade security [of the product]" and with "insight into the project's internal kitchen" [15]. Later guidance [16] recommends software development teams to identify a "security advisor" and to identify who is "responsible for tracking and managing security for the product".

39. In addition to the risks identified earlier, the examiners claim that the absence of security champions in each development team, and more precisely, the absence of the more technical skills specifically associated with secure systems engineering[10] could also evolve towards an over-reliance or over-dependence on the efficiency of reactive vulnerability detection mechanisms put in place at later stages of the development process.

Mitigating factor

40. Swiss Post's e-voting system development team reported having unrestricted access to an external application security advisor who supports the team with technical advice and reviews on secure software engineering and cryptography.

Recommendation

41. [R-05] The examiners encourage Swiss Post to formally appoint a security responsible, or *security champion*, for each team in the e-voting system.

Duties of the security responsible could include, among others, the following:

- Stay up to date with known and emergent cybersecurity threats that could affect the team, its processes, its tools, and its deliverables,
- Establish, document, and maintain the baseline threat model for the team or project,
- Support the team in identifying and proposing reasonable countermeasures to threats that require mitigation,

[10] E.g., identifying and deploying secure design and defensive coding principles, conducting architecture and design threat modelling and assessments, reviewing critical source code for security vulnerabilities, identifying abuse cases and positive/negative security testing scenarios, etc.

- Support the team in establishing standards for designing secure features and writing secure code, whether through its own internal effort or through the evangelisation of the organisation's security standards,
- Help members of the team become autonomous in their individual ability to assess whether a change request or any other work item should be further investigated,
- Contribute to the team's adoption of a baseline feature readiness and completion standard that improves security (i.e., definition of ready, definition of done),
- Support the product owner's decision process with relevant cost-benefit analyses regarding cybersecurity risk,
- Evangelise application security amongst the team,
- Contribute to establishing trusted interfaces between the team and third parties, both internal (e.g., organisation's information security and incident response structures) and external (e.g., user groups, communities, security experts, etc.).

Hesitations could typically arise on whether the security responsible should be appointed at full-time (vs. part-time along with other responsibilities), and whether he or she should be an active contributor to the project (vs. an external individual). It is the examiners' opinion that the allocated time should reflect no more, and no less, than the time necessary to carry out the assigned duties, which may typically vary depending on the team's size and work velocity.

Regarding the appointment of a third-party, it is the examiners' opinion that the security responsible should be appointed from within existing members of the team, or from within existing members of teams that work closely with the team he or she would be assigned to (e.g., someone working on component A team could typically be appointed as security responsible for both component A and component B teams).

Regarding conflicts of interest, the examiners advise against diluting the position amongst multiple individuals in any given team, and against appointing a member of the team that solely carries out a critical responsibility towards the team. More particularly, the security responsible position should not be given to an architect that acts as the sole architect in a team, or to a developer that is solely in charge of implementing security-critical features of the product, when possible.

4.3 F-03 Insufficient protection measures against malicious third-party components

Federal Chancellery requirement(s)

Key	Requirement
24.1.1 24.1.15	A life cycle model is defined. The life cycle model: » is used for the development and maintenance of the software (a); » **provides for the necessary controls during the development and maintenance of the software (b);** » **is documented (c).**

Severity

42. MODERATE.

Rationale

43. Based on their observations (see Appendix: Observations), the examiners identified an increased focus on preventing the risk of vulnerable third-party components (TPCs) using automated software composition analysis[11]. From a security point of view, the examiners recommend Swiss Post to continuously consider two categories of TPCs, which could be intentionally or accidentally embedded into the e-voting system:

» *Malicious TPCs*: components intentionally deployed by malicious actors. These typically host hard-to-detect vulnerabilities or backdoors.
» *Vulnerable TPCs*: components unintentionally published with vulnerabilities, typically because of poor development practices (e.g., insecure design, insecure coding, insufficient testing).

Additionally, the examiners recommend assessing two threats in the context of e-voting:

» *Backdoored components*, where an unauthorised party has remote control over the component's behaviour,
» *Logic bombs*, where a component is intentionally (attacks) or accidentally (errors) built to execute unexpected or undesired logic once certain conditions are met (e.g., a time delay, a network signature, a user input, etc.).

The actual vectors through which these threats could arise are numerous and would typically include, at least:

» Malicious TPCs unknowingly chosen and imported by developers into the e-voting system,
» Internally developed components being maliciously replaced by external components (e.g., dependency injection attacks),

[11] An extended introduction on software composition analysis is included in the appendix (Observations - Development process).

- TPCs being imported from trusted editors or sources, which have then unknowingly published compromised software (e.g., following a supply chain attack, compromised editor's infrastructure or developer accounts, etc.),
- TPCs that are granted permission to execute arbitrary commands upon their installation or after instantiation at runtime,
- TPCs that collect and store sensitive information unbeknownst to the developers (e.g., unidentified or undocumented log trails, debug builds, etc.) or relay information to third parties (e.g., relaying of pseudo-anonymised or pseudo-non-confidential data to external monitoring or advertising systems, etc.)

44. The examiners recognise that monitoring TPCs for publicly known vulnerabilities or malicious intent using automated checking of software bill of materials (SBOMs) constitutes an essential control for mitigating TPC risk. Still, this approach covers the "publicly known" spectrum of the threat, and fails to cover its "unknown / undisclosed" counterpart, as illustrated below[12]:

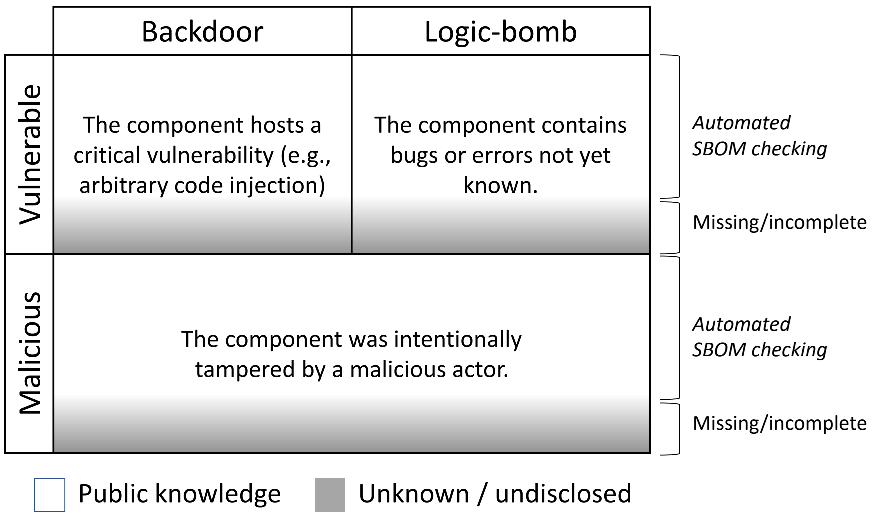

Backdoor
Logic-bomb
Vulnerable
The component hosts a critical vulnerability (e.g., arbitrary code injection)
The component contains bugs or errors not yet known.
Automated SBOM checking
Missing/incomplete
Malicious
The component was intentionally tampered by a malicious actor.
Automated SBOM checking
Missing/incomplete
Public knowledge
Unknown / undisclosed

Figure 2 - Security threat landscape - third-party components (TPCs)

45. Considering the above, the examiners claim that failing to implement adequate controls to mitigate TPC risk increases the exposure of the e-voting system to uncontrolled behaviour or unauthorised access.

Indeed, the insertion of a malicious TPC in the e-voting system could, in the examiners' opinion, compromise both vote secrecy (i.e., the component could leak data) or the availability of the voting system itself (i.e., the component could trigger anomalous behaviour and trigger the interruption of the ballot).

Mitigating factor

46. Assuming the cryptographic protocol [17] implementation satisfies the requirements set forth in the Federal Chancellery's ordinance on electronic voting [6], the examiners estimate

[12] The ratio between the "publicly known" and "unknown/undisclosed" areas has been set arbitrarily.

the likelihood of successfully compromising individual or universal verifiability properties of the Swiss Post e-voting system, as the result of embedding a malicious or vulnerable TPC in the system, as very unlikely, thanks to the multiple proofs and intermediate control opportunities offered by the protocol along the chain of events.

Recommendation

47. [R-06] The examiners encourage Swiss Post to extend and formalise its position on third-party component risk. In particular:

- » Establish or update a reference threat model for the use of third-party components[13]. This threat model would at least identify opportunities to reduce the exploitation of publicly unreported backdoors or vulnerabilities in third-party components embedded in the e-voting system, and entries to mitigate the successful ignition of unidentified or undesired behaviour (e.g., faults, errors, bugs) through those components.
- » Evaluate these threats and their associated risk under the regular risk assessment methodology.
- » Identify adequate controls and/or countermeasures to mitigate the threats for which risk is above threshold.
- » Track and report on the correct implementation of these controls and/or countermeasures.
- » Document the above (e.g., "usage of third-party components").

48. [R-07] The examiners encourage Swiss Post to formalise and document a third-party component vetting procedure aimed at assessing its security risk. The procedure could, among other things, assess[14]:

- » The editor's likelihood and ability to maintain the component in the near future (e.g., number of employees / developers, longevity of the component, release cycle, etc.),
- » The security documentation of the component (e.g., information on the development process and security testing performed, etc.),
- » Whether the component was likely reviewed or assessed by third parties,
- » Alternatives in the case of a security failure in the component (e.g., ability to take over the component's source code and maintain it, identifying alternate components, etc.)

[13] On this topic, the examiners strongly recommend evaluating SLSA, a framework designed to help software application owners and acquirers increase trust in the supply chain and reduce risk from supply chain attacks [18], [19].

[14] Examples of third-party component evaluation criteria can be found in referenced work (see: Criteria, additional criteria: inclusion of third-party components).

4.4 F-04 Insufficient security documentation

Federal Chancellery requirement(s)

Key	Requirement
24.1.20	Software development security documentation includes: » **a description of the physical, procedural, personnel, and other security measures necessary to protect and ensure the integrity of the design and implementation of the software in its development environment (a);** » **evidence that the security measures provide the necessary level of protection to preserve the integrity of the software (b).**

Severity

49. MODERATE.

Rationale

50. The examiners were given access to an extended amount of security documentation (see Appendix: Observations - software security documentation) about the voting protocol, the development process, types of testing and testing tools, and the architecture of the operational environment, among others.

However, security assurance information is distributed across many documents and essential information is still missing from the proposed evidence, such as:

- Threats to all components of the e-voting system, including their mitigation measures and the status of these mitigations[15],
- Standard security architecture and design principles,
- Coding standard, guidelines, or rules with instructions on how to address well-known or common security threats,
- Security testing procedures (i.e., what is precisely subject to security testing, and how) and code attestation procedures (i.e., how production confirms that the code is legitimate).
- Protection measures set forth in the development environment to prevent unauthorised access to the deliverables (e.g., code protection measures, developer access protection measures, build environment protection measures, pipeline environment protection measures, artifacts protection measures, protection acquisition of third-party components, etc.)[16].

[15] As indicated in the observations (see Appendix: Observations - Security documentation), evidence of risk management processes and documents produced at the organization level were shown to the examiners. However, these documents failed to demonstrate an actual threat analysis specifically focused on the e-voting system, including both its operational and non-operational processes and environments, in addition to the default analysis provided by the Federal Chancellery [6, p. 24]

[16] Equivalent documentation and evidence were produced for the production environment, this item aims at extending these efforts to include the development environment, too.

In the examiners' opinion, centralizing the information and completing it with how security assurance is obtained prior to code release would help Swiss Post better demonstrate its efforts and increase public trust over the overall process.

Recommendation

51. [R-08] The examiners encourage Swiss Post to centralize security relevant information on its voting system software. For example, Swiss Post could produce a whitepaper on the security of its voting system.

The examiners would recommend including the following information, among other things:

- The inventory of security requirements, which the system must satisfy,
- The list of assumptions, which were made on elements outside the control of Swiss Post,
- A list of threats to the voting system and its components,
- Information on how these threats were or are mitigated by Swiss Post,
- Information on how security is attested throughout the entire system's lifecycle, including not only during its development but also while in operation.

4.5 F-05 Insufficient quality control over security attestation operations

Federal Chancellery requirement(s)

Key	Requirement
24.5	**Regular and objective checks are carried out to ensure that the processes carried out and the associated work products comply with the description of the processes, standards and procedures to be implemented (a).** Deviations are followed up until they are corrected (b).

Severity

52. MODERATE.

Rationale

53. Through their observations (see Appendix: Observations - quality assurance), the examiners noted a lack of processes and measures to assess the quality and performance of security assurance methods and tools put in place in the development process.

In particular, the examiners noted that, although Swiss Post uses multiple internal (e.g., software composition analysis, automated code scanning, etc.) and external (e.g., code and documentation publicly accessible, penetration tests, bug bounty) security assurance mechanisms, limited processes seem to be implemented to assess whether the chosen mechanisms work as expected or intended, and whether their efficiency could be improved.

Recommendation

54. [R-09] The examiners encourage Swiss Post to implement a procedure to assess the efficiency (and effective operation) of its various security assurance tools and methods.

For example, a first iteration of such process could aim at:

» Establishing an inventory of security assurance mechanisms used during the development process (and/or later). This would typically include tools and methods.
» Selecting tools or methods that should undergo continuous improvement. The examiners recommend targeting scanning tools (i.e., static analysers, dynamic analysers, software composition analysers, etc.) and the development process first.

» Implementing a repeatable technique or method to evaluate the efficiency and/or maturity of the tool[17] or method[18].

4.6 F-06 Insufficient security testing and attestation

Federal Chancellery requirement(s)

Key	Requirement
17.2	An analysis must be made of the test coverage. This includes evidence that: » the tests defined in the test documentation match the functional specifications of the interfaces (a); » **all interfaces have been fully tested (b).**

Severity

55. MODERATE.

Rationale

56. Through their observations (see Appendix: Observations - security testing), the examiners noted strong evidence of testing at the coding stage of the process (static analysis) but a lack of evidence of other forms of security testing, such as earlier testing activities (e.g., architecture or design security reviews or validations) and pre-release testing activities (e.g., dynamic/runtime application testing).

57. Both static and dynamic application testing platforms tend to promise complete coverage of the code's execution graph. Still, full coverage remains difficult to achieve in practice due to many variables (e.g., cyclomatic complexity thresholds, non-deterministic interfaces between components, incorrect identification of taint data/data sinks, hardware limitations or configured restrictions in the scanning engine, etc.) and a difficulty to capture certain types of vulnerabilities. For this reason, combining both approaches (static + dynamic/runtime testing) is generally recommended to reduce the risk of false negatives.

[17] One approach involves inserting vulnerabilities (or vulnerable components) intentionally to validate the capabilities of tools and methods to detect and report weaknesses. This method also helps assess depth & coverage of testing (e.g., blending vulnerabilities in deeper or more complex sections of the code) and to gather insight on which types of vulnerabilities may or may not be found (e.g., blending specific vulnerability classes into the code). Methods can go from manual (e.g., developer inserts broken code manually) to automated (e.g., build jobs that automatically insert vulnerable code snippets to ensure that automated scanning tools will detect these). Safety mechanisms such as context- or input-dependent triggers [20] can be implemented when testing third party attestation methods (e.g., penetration tests, bug bounties, etc.).

[18] Examples of quality-oriented approaches to assess the integration of security in the development process can be found in SAMM (software assurance maturity model) [11] and BSIMM (building security in maturity model) [14], which are both publicly available, peer-reviewed, and can help teams and organisations benchmark their development process against themselves or against other similar organisations.

Mitigating factors

58. The lack of runtime/dynamic testing is partially mitigated by the exposure of the releases to community testing efforts through the Swiss Post bug bounty program and by subjecting approved releases to penetration tests. Although these testing methods have been regularly proven to be fully capable of helping to identify undeterred flaws or vulnerabilities, even the most critical ones, their performance remains highly variable and susceptible to the human factor (e.g., availability of competent testers, curiosity of testers, reward model, etc.). As such, they should not be considered systematic, and their coverage may vary ostensibly.

59. The examiners noted that these various validation efforts appear to be in place for changes that affect or associate closely to the voting protocol, typically through requesting assessments or assistance from an external security advisor.

Recommendation

60. [R-10] The examiners encourage Swiss Post to include systematic and repeatable security validations at earlier stages of the development process of the e-voting system. These could take place during design stage when processing change requests, and take the form of controls (e.g., secure design guidelines) or a method, which must be formally validated (typically, by a security champion, see finding F-02).

The examiners recognise that including such reviews systematically for all change requests may be counter-productive and recommend using a risk-based approach. For example, implementing a rapid risk assessment *questionnaire* would allow any team architect or developer to rapidly assess whether a change request could pose a security risk and would facilitate triage of requests that need to undergo a more formal security design review.

61. [R-11] The examiners encourage Swiss Post to generalise and leverage dynamic or runtime security testing in the development process of the e-voting system. Most dynamic/runtime security tools offer modes of operation specifically designed to perform testing with minimal human intervention, thus allowing automation and integration in continuous integration and delivery pipelines.

More particularly, the examiners recommend integrating two types of runtime testing methods:

» Testing for well-known vulnerabilities, well-known attack classes and well-known errors with security-focused dynamic application security scanners,
» Testing software components for accrued resistance to fault through fuzz testing [21].

4.7 Summary of findings

Key	OEV key(s)	Finding	Severity
F-01	24.1.1	Insufficient integration of security in the software development lifecycle	Moderate
F-02	24.1.1	Conflicting / ambiguous attribution of security responsibilities	Moderate
F-03	24.1.1	Insufficient protection from risky third-party components	Moderate
F-04	24.1.20	Insufficient security documentation	Moderate
F-05	24.1.15 24.5	Insufficient quality control over security attestation operations	Moderate
F-06	17.2	Insufficient security testing	Moderate

Table 8 - Summary of findings

4.8 Summary of recommendations

Key	OEV key(s)	Recommendation	Finding
R-01	24.1.1	Formalise the integration of security development lifecycle guidance or best practices into the e-voting system's development process.	F-01
R-02	24.1.1	Ensure e-voting system personnel and stakeholders have received adequate role-based training on secure systems engineering.	F-01
R-03	24.1.1	Integrate threat modelling, or equivalent, in early stages of the development process.	F-01
R-04	24.1.1	Establish a baseline set of security principles or rules for each phase of the development lifecycle (e.g., requirements, architecture and/or design, coding, testing, build, deployment, etc.).	F-01
R-05	24.1.1	Establish a security champion program and appoint a champion in each e-voting system development team.	F-02
R-06	24.1.1	Establish a reference threat model for the use of third-party components in the e-voting system, maintained with the status of implementation of chosen controls and countermeasures.	F-03
R-07	24.1.1	Establish a security vetting process for the selection of new third-party components, and the review of existing ones, embedded in the e-voting system.	F-03
R-08	24.1.20	Establish a central document that describes how security assurance in the e-voting system is obtained (e.g., e-voting security whitepaper).	F-04
R-09	24.5 24.1.15	Establish procedures to confirm, review and improve the correct operation of security attestation measures, in particular automated measures.	F-05
R.10	17.2	Establish procedures to review and/or validate design proposals generated in response to change requests, prior to entering the coding phase.	F-06
R.11	17.2	Establish or improve runtime/dynamic application security testing procedures executed prior to release. For relevant components, extend these procedures with additional fuzz testing.	F-06

Table 9 - Summary of recommendations

5 References

[1] "Reorienting eVoting and ensuring stable trial operation," *www.egovernment.ch*. https://www.egovernment.ch/en/umsetzung/schwerpunktplan/vote-electronique/ (accessed Oct. 21, 2021).

[2] Swiss Federal Chancellery, Political Rights Section, "Redesign and relaunch of trials - Final report of the Steering Committee Vote électronique (SC VE)." Nov. 30, 2020. Accessed: Dec. 06, 2021. [Online]. Available: https://www.bk.admin.ch/dam/bk/en/dokumente/pore/Final%20report%20SC%20VE_November%202020.pdf.download.pdf/Final%20report%20SC%20VE_November%202020.pdf

[3] Swiss Federal Chancellery, Political Rights Section, "Partial revision of the Ordinance on Political Rights and total revision of the Federal Chancellery Ordinance on Electronic Voting (Redesign of Trials)." Apr. 28, 2021. Accessed: Dec. 06, 2021. [Online]. Available: https://www.bk.admin.ch/dam/bk/en/dokumente/pore/Explanatory%20report%20for%20consultation%202021.pdf.download.pdf/Explanatory%20report%20for%20consultation%202021.pdf

[4] Swiss Federal Chancellery, "Federal legislation." https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting/versuchsbedingungen.html (accessed Oct. 21, 2021).

[5] Swiss Federal Chancellery (FCh) - Political Rights section, "Audit concept for examining Swiss Internet voting systems - v1.3." May 18, 2021.

[6] Swiss Federal Chancellery, "Federal Chancellery ordinance on electronic voting (OEV)." Apr. 28, 2021. [Online]. Available: https://www.bk.admin.ch/dam/bk/en/dokumente/pore/OEV_draft%20for%20consultation%202021.pdf.download.pdf/OEV_draft%20for%20consultation%202021.pdf

[7] Swiss Post, "UP2021 - Mapping List VEleS.xlsx." Jul. 13, 2021.

[8] Payment Card Industry, "Secure software lifecycle (Secure SLC) requirements and assessment procedures - PCI-SSF v1.0." Jan. 2019.

[9] Center for Internet security, "CIS Controls Version 8," *CIS*. https://www.cisecurity.org/controls/v8/

[10] Software assurance forum for excellence in code (SAFECode), "Fundamental practices for secure software development - 3rd ed." Mar. 2018.

[11] Watson, Colin, Lynch, Aidan, Coblentz, Nick, Keary, Eoin, and Deleersnyder, Seba, "SAMM Assessment toolbox v1.5 final." OWASP, 2009. [Online]. Available: https://github.com/OWASP/samm/tree/master/Supporting%20Resources/v1.5

[12] Software assurance forum for excellence in code (SAFECode), "Managing security risks inherent in the use of third-party components." 2017.

[13] M. G. Jaatun and D. S. Cruzes, "Care and Feeding of Your Security Champion," 2021, pp. 1–7.

[14] BSIMM, "Building security in maturity model (BSIMM) Trends & insights report - ver.12," 2021. [Online]. Available: https://www.bsimm.com/download.html

[15] Alexander Antukh, "Security champions 2.0," presented at the OWASP Bucharest AppSec conference 2017, 2017. [Online]. Available: https://owasp.org/www-pdf-archive/OWASP_Bucharest_2017_Antukh.pdf

[16] Microsoft, "Simplified Implementation of the Microsoft SDL." Microsoft, Mar. 02, 2011. Accessed: Dec. 04, 2021. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=12379

[17] Swiss Post, "Protocol of the Swiss Post voting system, Computational proof of complete verifiability and privacy - v.0.9.11." Oct. 15, 2021.

[18] Kim Lewandowski and Mark Lodato, "Introducing SLSA, an End-to-End Framework for Supply Chain Integrity," *Google Online Security Blog*. https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html (accessed Dec. 01, 2021).

[19] *SLSA ("salsa") is Supply-chain Levels for Software Artifacts*. 2021. Accessed: Dec. 01, 2021. [Online]. Available: https://github.com/slsa-framework/slsa

[20] B. Dolan-Gavitt *et al.*, "Lava: Large-scale automated vulnerability addition," 2016, pp. 110–121.

[21] "Fuzzing," *Wikipedia*. Jan. 03, 2022. Accessed: Jan. 05, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Fuzzing&oldid=1063609411

[22] Swiss Post, "Release management service e-voting, v1.02." Jun. 25, 2021.

[23] Swiss Post, "Configuration management software development tools for e-voting service - v.17." Jun. 11, 2021.

[24] Swiss Post, "Software development process of the Swiss Post voting system," *GitLab*. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/fd5f9756cb633e683f144a6692e6e2ea1581406c/Product/Software%20development%20process%20of%20the%20Swiss%20Post%20voting%20system.md (accessed Nov. 18, 2021).

[25] Swiss Post, "Software development process of the Swiss Post voting system (Addition) - v41." Oct. 27, 2021.

[26] Jim Campbell, "Scrum Methodology: Breaking Down the Scrum Framework," Jun. 05, 2020. https://scrumexplainer.com/scrum/scrum-methodology/ (accessed Dec. 12, 2021).

[27] "JFrog Xray - Universal Component Analysis & Container Security Scanning," *JFrog*. https://jfrog.com/xray/ (accessed Dec. 28, 2021).

[28] Swiss Post, "SwissPost voting system architecture document - v.0.9.1." Aug. 17, 2021.

[29] "Swiss Post - E-Voting bug bounty program," *YesWeHack #1 Bug Bounty Platform in Europe*. https://yeswehack.com/programs/swiss-post-evoting (accessed Oct. 31, 2021).

[30] Swiss Post, "Infrastructure whitepaper of the Swiss Post voting system," *GitLab*, Nov. 15, 2021. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/master/Operations/Infrastructure%20whitepaper%20of%20the%20Swiss%20Post%20voting%20system.md (accessed Dec. 04, 2021).

[31] Swiss Post, "Swiss Post e-voting documentation (home)," *GitLab*. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation (accessed Nov. 21, 2021).

[32] Swiss Post, "Test Concept of the Swiss Post Voting System," *E-voting documentation*, Nov. 15, 2021. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/master/Testing/Test%20Concept%20of%20the%20Swiss%20Post%20Voting%20System.md (accessed Oct. 31, 2021).

[33] "SonarQube (editor's website)." https://www.sonarqube.org/ (accessed Dec. 05, 2022).

[34] "Burp Suite." https://portswigger.net/burp (accessed Dec. 05, 2021).

[35] OWASP, "Software assurance maturity model (SAMM)," *OWASP SAMM*, 2020. https://owaspsamm.org/ (accessed Dec. 08, 2021).

[36] Swiss Post, "Swiss Post e-voting - web portal," *Swiss Post*. https://www.evoting.ch/en (accessed Nov. 06, 2021).

[37] Swiss Post, "Issues · swisspost-evoting," *Swisspost-evoting*. https://gitlab.com/groups/swisspost-evoting/-/issues (accessed Nov. 15, 2021).

[38] Swiss Post, "Submitting findings (online form)," *Evoting-Community*. https://evoting-community.post.ch/en/contributions/submitting-findings (accessed Oct. 31, 2021).

[39] Swiss Post, "Code of conduct," *E-voting documentation*. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/master/CONTRIBUTING.md (accessed Oct. 31, 2021).

[40] GitLab, "Confidential issues," *GitLab Docs*. https://docs.gitlab.com/ee/user/project/issues/confidential_issues.html (accessed Oct. 31, 2021).

[41] Swiss Post, "Ways to submit - Severity of findings," *E-voting documentation*. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-

/blob/master/REPORTING.md#user-content-severity-of-findings (accessed Oct. 31, 2021).

[42] Swiss Post, “Change management - v1.03.” Mar. 05, 2020.

6 Appendixes

6.1 Observations

6.1.1 Development process and lifecycle

62. Swiss Post proposes documented evidence of its development lifecycle and maintenance processes in:

- » Release management concept for e-voting service [22]
- » E-voting configuration management software development tools [23]
- » Software development process of the Swiss Post voting system [24]
- » Software development process of the Swiss Post voting system (addition) [25]
- » ICT DEV - Architecture, processes and guidelines (no reference available)

63. The examiners recognise the definition and documentation of a software development lifecycle, which includes more than thirty steps, as depicted in Appendix 8.2.

64. Swiss Post designed its e-voting system development and maintenance process based on an agile-based software development methodology. The methodology is heavily inspired from Scrum [26] and adaptations from Swiss Post have resulted in its own implementation referred to as the Post Agile Methodology (PAM).

65. The process specifies multiple control gates ("quality gates") spread across all phases and aimed at ensuring that requirements are met (e.g., QG-R1, QG-R2, etc.). In some circumstances (when a release is "limited in size, has minor impact and associated risk is small"), all gates may not necessarily be all enforced and "product management decides which quality gates are conducted" [22, p. 6]. The process guarantees however that all gates are enforced when change affects the "core product" and results in a "major or minor version" [22, p. 6].

66. Each proposed feature or improvement can either be flagged for relevance "for IT security (infosec)", and/or flagged for relevance for risk ("risk if the change is implemented / risk if the change is not implemented"). The examiners did not identify documented evidence of the criteria that qualifies when a feature will or will not be flagged, it is assumed that this event occurs based on a per-case basis and contingent to the individual's own perceptions.

67. Several tools are deployed to support the development operation ("e-voting service toolchain" [24]). These include among others a code versioning and control system (Bitbucket) with access control and auditing capabilities, an automation server used for achieving continuous integration and delivery (Jenkins), several unit testing frameworks (xUnit, xRay test mgmt., Selenium) and an issue/ticketing and release management platform (Jira). The overall testing strategy and framework is documented (see Appendix: Observations - Testing).

68. The continuous integration and delivery pipeline established by the development team does not extend into production. This can be easily justified by the highly sensitive nature of the system, as many additional steps, third-party verifications and attestations are required prior to deploying releases into production. The examiners noted that Swiss Post replaced manual operations with automation whenever possible and adequate.

69. The examiners noted that new source code is verified by static source code analysers (SonarQube, Fortify) before its release. These tools are designed to automatically check source code for quality issues and known security vulnerabilities and errors.

70. The examiners noted that Swiss Post has implemented a software composition analysis (SCA)[19] sub-step in the build phase [25, p. 23] of its e-voting system development process. The tool [27] inventories third-party components[20] (TPCs) embedded in the e-voting system and is configured to report on possible security and licensing policy violations, thus likely offering an effective control against vulnerable third-party components.

71. The examiners noted that, although TPCs are monitored for known security vulnerabilities, Swiss Post failed to produce evidence of a satisfying process on the following:

- » How Swiss Posts protects itself from malicious TPCs both prior to their inclusion and after their inclusion into the e-voting system,
- » How existing TPCs were assessed and how will future TPCs be assessed, in addition to scanning for known vulnerabilities and licensing violations,
- » How violations, which cannot be resolved with an update of the component, will be addressed.

72. Although security appears to be widely recognised as paramount to the success of the project by all persons interviewed, the examiners noted that Swiss Post has not formally specified and assigned responsibilities related to the product's security to all individuals involved in the product development and maintenance. This is discussed in more detail in finding F-02.

[19] Software composition analysis (SCA) is the process through which one gains visibility over TPCs bundled into a software application. An efficient implementation of SCA not only inventories libraries explicitly imported by developers, but also reveals hidden sub-components, which could have been nested in TPCs themselves, often without knowledge of the component's publisher. SCA facilitates the production of the software bill of materials (SBOM), an inventory of components of which a software application is made. An SBOM typically includes unique component identifiers and their respective version. This enables an automated validation of components against various databases to detect policy violations. Modern SCA tools typically propose three types of policies to help application owners mitigate three risks:
- Licensing policies (e.g., identify components that enforce a licensing scheme which is incompatible with the licensing set forth in the software application),
- Security policies (e.g., identify components that have been reported vulnerable or malicious),
- Lifecycle policies (e.g., identify components that have reached or are nearing end-of-support by their editor).

[20] Third-party components: pre-made software building blocks (libraries, APIs, frameworks, etc.) imported or embedded into the e-voting system.

73. The examiners were unable to assess with confidence the actual level of maturity of the procedure to perform dynamic or runtime security testing tasks. Although a tool is mentioned (Burp Suite), the examinees failed to obtain conclusive evidence on whether this step is integrated and at what maturity level it is performed.

74. The examiners noted the absence of a formal threat modelling or abuse case identification activity, or equivalent, within the e-voting system project. Threat modelling can typically help both the development team and the product owner identify how a feature could be attacked or abused by various threat agents, and how it could be mitigated, prior to beginning its implementation. The use and inclusion of abuse cases (or *attacker stories*) as part of the regular development process was not identified either.

75. The examiners noted that guidance on the system's design or architecture security appears to be heavily focused on the cryptographic protocol of the e-voting system [17]. Additional information on how security impacted the design or development can be found in the system's reference architecture documentation, such as "architecture principles" [28, p. 49]. Still, the examiners note the absence of a formal secure design/architecture guideline or standard principles. The same limitation applies to secure coding guidelines for the developers, where only guidance on how to perform "argument checking using Guava preconditions" [28, p. 51] was identified.

76. The examiners noted that the developers and the architects directly involved in the e-voting system development and maintenance are trained on the development methodology. However, they did not receive formal role-based training on security in software engineering (i.e., secure development lifecycle, secure systems engineering, secure coding / defensive programming training, security testing, etc.).

77. The examiners noted that the members of the teams have unrestricted access to a trusted third-party security advisor with demonstrated experience in secure systems engineering, defensive programming and applied cryptography. The third-party is also involved in architecture, design, and code review operations for critical parts of the system.

78. The examiners noted that a trusted build procedure has been specified and documented. It allows third parties to reproduce binaries or components, which are identical to those in operation during a ballot, and thus facilitate the verification of the system. However, the examiners were unable to assess with confidence that deliverables are adequately certified and authenticated prior to leaving the build system (i.e., the examiners failed to obtain assurance that deliverables cannot be tampered after leaving the development environment).

79. It is expected that the system and its components will be regularly tested for vulnerabilities and errors by trusted third parties. Testing methods include penetration testing (also commonly referred to as *ethical hacking*), which involves hiring experts specialised in offensive security testing. Additionally, a bug bounty program [29] is in operation at time of editing this report. It allows individual security testers and researchers to test the security of the e-voting system in exchange for rewards when vulnerabilities are found and reported according to well-defined rules.

80. On a more general perspective, the examiners noted that the Swiss Post e-voting program did not identify or select a standard or reference model on which to benchmark its own development process in relation with security and measure its own maturity.

81. The examiners noted that security responsibilities inside the e-voting system software engineering team are implicitly carried out by the two architects already in charge of the software architecture and design.

82. In addition to the above, the examiners noted the absence of a formal role of "security champion", or equivalent, appointed within the e-voting software engineering team.

6.1.2 Software security documentation

83. Swiss Post proposes documented evidence of the security of the e-voting system in:

- Whitepaper - Infrastructure of the Swiss Post voting system [30]
- Software development process of the Swiss Post voting system [24]
- Software development process of the Swiss Post voting system (addition) [25]
- Trusted build of the Swiss Post voting system (no reference available)
- Release management concept for e-voting service [22]
- Swiss Post voting system architecture document [28]
- ICT DEV - Architecture, processes and guidelines (no reference available)
- E-voting configuration management software development tools [23]
- E-voting ISDS Konzept (no reference available)
- E-voting ISDS Risikoportfolio (no reference available)
- E-voting ISDS Risikoanalyse (no reference available)
- E-voting ISDS Risikomanagement (no reference available)

84. An essential part of evaluating the completeness of this requirement (24.1.20) came to identifying what could qualify as 'security documentation' and which software components needed to have their security documented. In order to help identify this answer, the examiners reviewed the computational proof of verifiability and privacy of the e-voting system [17]. Among other things, the document specifies three primary security goals for the e-voting system (individual and universal verifiability, and voting secrecy) [17, p. 11] and documents a threat model.

The examiners also noted the presence of a statement, which they deemed essential to the context and objectives of this examination:

"*We assume that the voting client and voting server are untrustworthy. There is a caveat regarding voting secrecy: obviously, an adversary controlling the voting client could spy on the voters’ choices. Therefore, the definition of vote secrecy assumes a trustworthy voting client*" [17, p. 6].

The examiners recognized the motivation behind this statement: for the protocol to offer voting secrecy just by itself, complete control over the voting server and the voting client would be necessary, which is both practically unfeasible and outside the scope of the protocol. Therefore, additional evidence of security may be deemed necessary to increase trustworthiness in the components involved in the e-voting system, which are not deemed trustworthy by the voting protocol. The protocol, therefore, only offers secrecy "where votes are centrally stored" [17, p. 16] and the burden of trustworthiness of the voting server and client are transferred to other actors or entities[21].

85. The examiners identified that establishing trustworthiness for the voting client required establishing trustworthiness for components under responsibility of the voter (i.e., the physical device, the operating system and its applications, the web browser, the network equipment, etc.) and for components under responsibility of Swiss Post (i.e., the voting server itself, its web application, its operating environment, and all the artefacts sent from the voting server to the voting client for rendering into the client's web browser).

The figure below illustrates this separation of responsibilities:

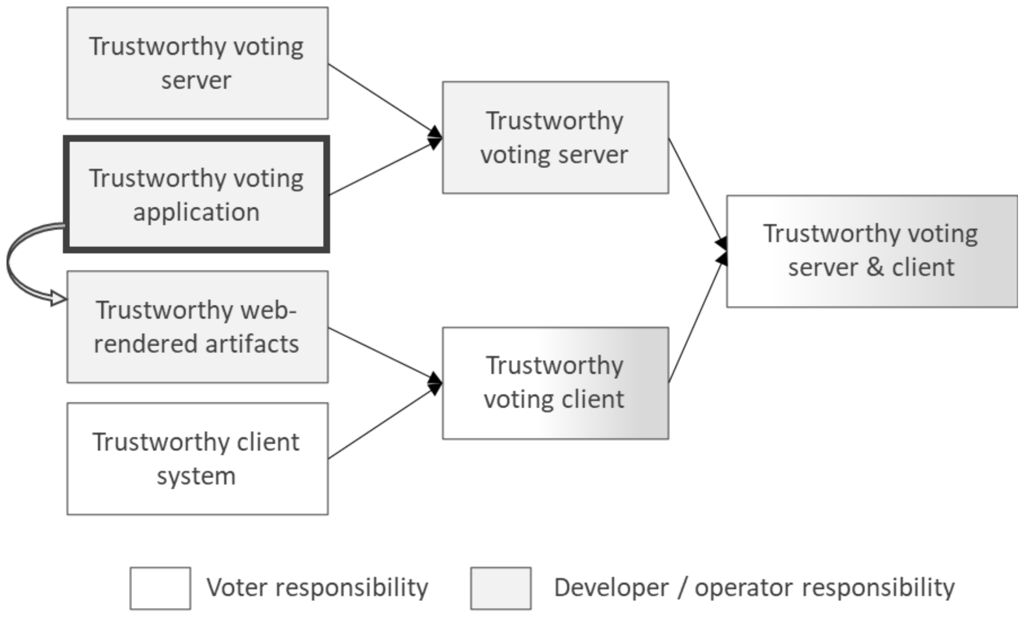

Trustworthy voting server
Trustworthy voting application
Trustworthy web-rendered artifacts
Trustworthy client system
Trustworthy voting server
Trustworthy voting client
Trustworthy voting server & client
Voter responsibility
Developer / operator responsibility

Figure 3 - Trustworthiness of the voting server and client - division of responsibilities

86. Based on the above, the examiners searched for evidence of documentation that offers both assurance on the security of the voting protocol itself and the security of the software applications (the "components"). They searched for, in particular:

- Documentation on threats to the e-voting software applications or components,
- Documentation on how these threats were addressed in the e-voting software applications or components,

[21] At this point, the authors deem necessary to emphasize that the Federal Chancellery's ordinance on electronic voting [6] proposed a conflicting vision on whether only the trustworthy elements of the voting system should offer voting secrecy (e.g., Art.7) or whether other components should be taken into consideration for the satisfaction of this requirement (i.e., can voting secrecy be achieved in the eyes of the Federal Chancellery when the voting server is compromised?). During the examination, the examiners privileged a conservative approach by adhering to the threat model proposed in the ordinance, which assumes that "a backdoor introduced into the system" could expose the "secrecy of the vote" [6, p. 26], and therefore searched for evidence of security documentation on other components that could be "backdoored", too.

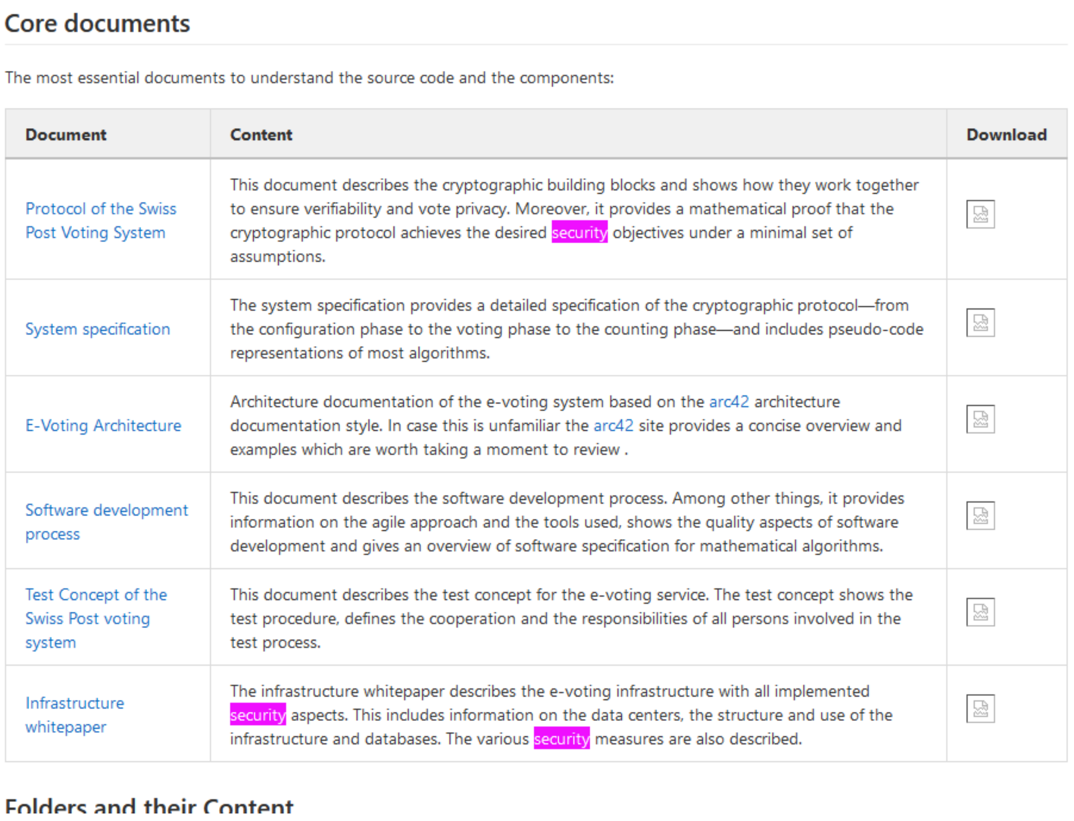
- Documentation on security-relevant design decisions, which had been taken throughout the course of designing, coding, testing and deploying the e-voting software applications and components,
- Documentation on security-relevant efforts or controls put in place to increase the security assurance of the e-voting software applications and components.

87. The examiners found extended evidence of security-relevant information about the Swiss Post e-voting system spread across a large set of documents:

- The specification of the protocol of the e-voting system, including its threat model and a computational proof[22] and a formal security analysis,
- Guidance on how to implement the protocol based on its specification,
- Documentation on the controls and countermeasures implemented in the operational data centres and hosting environment,
- Documentation on the software development and maintenance process,
- Documentation on the software release process,
- Documentation on the security tests performed on the software components,
- Documentation on the production of trusted and verifiable builds,
- Documentation on the architecture and infrastructure of the system, including quality requirements, architecture and design principles, and security requirements,
- A risk register, which lists the threats specified by the Federal Chancellery.

88. In the publicly accessible repository [31], the examiners noted that Swiss Post mentioned the presence of security-relevant information in two documents: the infrastructure whitepaper [30] and in the computational proof of the voting protocol [17], as illustrated on the screenshot below taken from the online documentation portal:

[22] The verification of the computational proof of the protocol fell outside the scope of this examination.



Core documents

The most essential documents to understand the source code and the components:

Document	Content	Download
Protocol of the Swiss Post Voting System	This document describes the cryptographic building blocks and shows how they work together to ensure verifiability and vote privacy. Moreover, it provides a mathematical proof that the cryptographic protocol achieves the desired security objectives under a minimal set of assumptions.	
System specification	The system specification provides a detailed specification of the cryptographic protocol—from the configuration phase to the voting phase to the counting phase—and includes pseudo-code representations of most algorithms.	
E-Voting Architecture	Architecture documentation of the e-voting system based on the arc42 architecture documentation style. In case this is unfamiliar the arc42 site provides a concise overview and examples which are worth taking a moment to review .	
Software development process	This document describes the software development process. Among other things, it provides information on the agile approach and the tools used, shows the quality aspects of software development and gives an overview of software specification for mathematical algorithms.	
Test Concept of the Swiss Post voting system	This document describes the test concept for the e-voting service. The test concept shows the test procedure, defines the cooperation and the responsibilities of all persons involved in the test process.	
Infrastructure whitepaper	The infrastructure whitepaper describes the e-voting infrastructure with all implemented security aspects. This includes information on the data centers, the structure and use of the infrastructure and databases. The various security measures are also described.	

Folders and their Content

Figure 4 - Security documentation on the e-voting system (13.12.2021)

89. The computational proof of the voting system protocol [17] provides information on the protocol itself. Although it includes a threat model and a formal security analysis, those specifically apply to the protocol itself.

90. Swiss Post produced a whitepaper on the operational infrastructure of its e-voting system [30] and its architecture [28], which answer questions an external auditor could have about the voting system infrastructure and architecture. The infrastructure whitepaper includes a section on e-voting security ("e-voting security"), which describes security measures deployed in the production environment. On the other side, there is no formal security whitepaper or similar that has been made available, except for the voting protocol.

91. Although information is provided about the security testing tools used to attest the security of the deliverables, the examiners did not identify details on the testing rules or profiles chosen by Swiss Post in the documentation.

92. Although the examiners found extended documentation on many aspects of the e-voting system, they failed to find evidence of the following:

» Which architectural and design principles were set forth as standards during the design and development of the voting components,
» Which coding principles or rules were set forth as standards during the implementation of the voting components,
» Which threats or risks were identified in each component, and how the architects or developers mitigated them,
» How is the security of the components tested,

- How the development environment, and more particularly its artefacts (e.g., source code, third party components and build environment), are protected from unauthorised access.

93. Overall, the examiners noted that Swiss Post extensively documented its processes and specifications surrounding both the voting protocol and the voting system build and production infrastructure. They note, however, that many internal documents released to the examiners included references to a document repository ("wikit[.]post[.]ch"), which was not accessible during the examination. While some of the gaps were identified during interviews, the examiners cannot exclude that part of the information that has been considered missing may in fact exist in those documents.

6.1.3 Quality assurance

94. Swiss Post proposes documented evidence of the security of the e-voting system in:

- Software development process of the Swiss Post voting system [24]
- Software development process of the Swiss Post voting system (addition) [25]
- KWP - Prozessowner-Zirkel - Link continual service improvement (no reference available)
- KWP - Prozessowner-Zirkel - Link Prozess Improvement Plan register - PIP (no reference available)

95. The examiners found extended documentation aimed at documenting the voting system and its processes surrounding the development, maintenance and operation of the system. The examiners were also shown proofs of testing (i.e., test reports) and the procedures set forth to carry out various sensitive operations (e.g., development process, release process, change management process, acceptance, and control quality gates, etc.).

96. The examiners failed to identify evidence of controls or processes to verify the effectiveness of the security attestation measures put in place to validate the voting system. In particular, the following evidence could not be verified:

- Evidence of testing or testing procedures on the static attestation tools (e.g., SonarQube and Fortify),
- Evidence of testing or testing procedures on dynamic attestation tools (e.g., Burp Proxy),
- Evidence of testing or testing procedures on community or third-party testing mechanisms (e.g., penetration tests, bug bounties, etc.).

6.1.4 Configuration management system

97. Swiss Post proposes documented evidence of the e-voting system configuration management system in:

- Release management concept for e-voting service [22]
- Configuration management software development tools e-voting service [23]

- E-voting configuration management software application & infrastructure (no reference available)

98. The examiners noted that the chosen configuration management system is designed to automatically authorise and trace all changes performed into the e-voting system's deliverables. Tracing data includes authorship (who made the change), timestamp (when the change was made), approval (who reviewed and validated the change), and the exhaustive nature of the change (what was changed).

99. Approval of changes is built on a four-eyes principle implemented in the configuration management system [25, Para. 8.3]:

- Changes are submitted to the configuration management system in the form of a request to accept the change (pull request),
- The change acceptation request is reviewed. Only a different individual than the one who submitted the request can process a change acceptation request.
- The second individual can reject the request or approve it.
- Once approved, the change is permanently entered (committed) into the configuration management system.

100. The examiners not that the process through which third party components are selected, vetted, imported and maintained into the configuration management system is poorly documented.

101. The examiners noted that the process through which changes to the change management system itself occur is poorly documented (e.g., modification of scripts or sequences in which build and deploy procedures are defined)[23]. In particular, whether or not these changes undergo the same four-eyes verification procedure remained ambiguous.

102. The examiners noted that deliverables produced by the configuration management system can be explained (i.e., exhaustive source code and components that compose the deliverable) and compared with other versions of the deliverables, future or previous.

103. The examiners noted that fingerprints are produced from the deliverables, using cryptographic hash functions. These fingerprints allow operators and systems in the e-voting production environment to validate the integrity of the deliverables running into production. This allows to both ensure that the external build process singularly matches the internal one (trusted build) and to detect whether its content was tampered prior to being deployed into production. Fingerprinting offers a simpler alternative to digital signatures (integrity vs. integrity + authenticity) and may be desirable in environments that offer a relatively short chain of custody between the build systems and the production environment.

[23] Deploying the four-eyes principle both upon submission of changes to the input (e.g., new source code, new components, etc.) but also upon submission of changes to the process itself (e.g., build engine scripts and job definitions) typically helps mitigating attacks against the build platform.

104. The reader should note, however, that examining whether these fingerprints are indeed verified, both during the external build and once the deliverables have reached the production environment, fell outside the scope of this examination.

6.1.5 Testing

105. Swiss Post proposes documented evidence of the security of the e-voting system in:

» Test concept of the Swiss Post voting system [32]

106. The examiners interpreted "security functions" (req.17.1 from the ordinance [6]) as the logical mechanisms built into the voting system with the specific intent to mitigate threats defined in Art. 4 of the OEV [6].

107. While reviewing the test concept, the examiners searched for information on the following and based on the ordinance's requirements (reqs.17.1, 17.2, 17.3, 25.13.3, 25.13.4. OEV):

» What are the functions relevant to the security of the system?
» Which components are subject to security testing?
» What tests will be performed?
» How will tests be performed?
» Does the testing approach match the recommended practice? (see: Criteria - Additional criteria)

108. The test concept describes, among other things, which artefacts are tested, which types of tests are carried out on the voting system, when and by whom. The concept also describes the process through which defects are handled, reported, and resolved. Finally, the concept also specifies the following:

» The test strategy includes 6 testing phases (stages), including tests carried out by developers (e.g., unit testing, integration testing and automated end-to-end testing), tests carried out by the testing management division (e.g., system installation tests, system integration tests, smoke tests, functional tests, regression tests, security tests, load & performance tests, penetration tests, disaster recovery tests, and accessibility tests), and acceptance tests carried out by customers (i.e., the Cantons).
» Security testing is mostly performed during stage 3 of the process, which is under responsibility of the test management division. This confers security testing independence from the developers.
» Security testing is performed both internally (e.g., pipelined execution of automated scanning tools) and externally (e.g., penetration tests).
» All identified defects are reported in an issue tracking tool and subjected to weighting, prioritisation, and reporting.

109. The examiners noted that the test concept lacks information on what qualifies for a security function in the system (req. 17.1.a). It mentions, however, that the admin portal, the voting portal, the secure data manager, and the integration tools are subject to security testing. Although this list creates some ambiguity with the list of components specified in the reference system architecture [28, Ch. 5.1.2], interviews indicate that security testing is

performed indiscriminately on all components instead of being strictly limited to their security functions.

110. The examiners noted that although the test concept refers to "security testing" multiple times, its definition is left to subjective interpretation. One section refers another type of testing under the term "business logic security testing", later defined as "tests [that] serve to ensure the security of the e-voting platform with regard to external access and manipulation risks" [32]. Overall, the examiners noted that the test plan does not mention any well-known method of security attestation performed during development, or earlier.

111. A review of the development process [24], [25], and more particularly its security tooling ("e-voting toolchain") indicates the use of tools typically aimed at delivering security assurance. These include static analysers (automated source code review) such as SonarQube [33] and Fortify, and a software composition analyser, Xray [27].

112. The documentation also indicates the use of a dynamic / runtime application testing tool (Burp Suite [34]). The examiners noted that, while the use of the static analysers was corroborated in other documents [23, p. 8], [28, p. 18], in shown reports, and during interviews, no other evidence seemed to corroborate the integration of dynamic testing in the development process.

113. The examiners also looked at signs of security testing or validation performed at earlier stages of the development process (before coding). These would typically take the form of architecture or design reviews [8, Sec. 3.2], [9, Sec. 16], [10, p. 10], [35, Sec. AA1], or equivalent. Evidence of such controls (threat modelling, design review) was only found in the voting protocol computational proof [17].

114. The examiners noted that security assurance also heavily relies on outputs produced by external contributors, such as third-party services suppliers (penetration tests) and the community (bug bounty program).

6.1.6 Transparency

115. Swiss Post proposes documented evidence of e-voting system's flaw remediation transparency in:

- E-voting portal [36]
- Test concept of the Swiss Post voting system [32]

116. The examiners interpret “transparent communication” as the quality that makes known flaws and their associated actions both visible to the public and sufficiently documented as to avoid unnecessary ambiguity. They also interpret “flaws” as properties of the system, whose intentional or accidental exercise may jeopardize the security objectives. Both the terms “flaw” and “vulnerability” are used interchangeably in this document.

117. The examiners noted that when a defect is found during testing, the defect and all relevant information “is documented as a bug in Jira”. Jira in this context refers to an issue tracking and project management tool, which is not accessible to the public.

118. The public is invited to report flaws to Swiss Post using one of three mechanisms: submitting an issue on the GitLab website[37], filling an online form hosted on the Swiss Post website[38] or filling a vulnerability report in the voting system's bug bounty program[29].

119. By design, submissions made through the bug bounty platform or through the online form remain entirely confidential, until eventually reported or disclosed by Swiss Post. The provided document does not indicate how or when disclosure to the public may occur.

120. Flaws reported through the GitLab platform [37] are visible by default to the community, unless the reporter has activated the "confidential" flag. When reporting a critical or high severity flaw through the GitLab platform, the code of conduct [39] provided by Swiss Post instructs reporters to activate the "confidential" flag.

121. According to GitLab's online documentation [40], the "confidential" flag makes an issue private by restricting it from public access. GitLab's confidentiality feature is implemented as a binary switch that either makes all issue details accessible to the community or makes it completely invisible and restricted to the project's members. Unless Swiss Post internally acknowledges the removal of the "confidential" flag, the public has no way of knowing if and when a critical issue has been reported through the GitLab issue reporting form[24].

122. The examiners noted that the Cantons have access to reports marked with the "confidential" flag, which allows them to conduct their own risk assessment following a report.

123. The examiners noted that maintaining secrecy over the details of a highly severe or critical flaw can be easily motivated by the general need to protect the voting system from attacks or fraud, and more particularly during ballots. Still, the chosen implementation not only hides the details of "confidential" flaws to the public, but also hides their mere existence entirely.

124. Swiss Post adopted a severity scale comprised of four (4) levels: critical, high, medium and low[41]. Critical severity findings include vulnerabilities or flaws whose exploitation "could compromise the application, the system or the voting process severely" and is "straightforward to set up", whereas high severity findings include those whose exploitation "could compromise the application, the system or the voting process" and is "hard to set up".

125. Although the security scale offers sufficient granularity, the examiners claim that an ambiguity arises from whether Swiss Post and the reporter will evaluate the finding based on identical perceptions of what constitutes a "compromise of the system" or "a severe

[24] GitLab's issue numbering scheme is based on a sequential integer (1, 2, ..., n). Assuming that only issues of the highest severity will be flagged as confidential, a motivated public auditor could monitor the list of issues for the presence of missing "IDs" or create a new issue and observe its assigned identifier to infer on the existence of a hidden "high severity" issue.

compromise of the system." This ambiguity could affect whether some flaws will be reported as high or critical, and consequently remain invisible to the public.

126. The examiners reviewed a sample[25] of the issues published on GitLab [37] that were marked as closed at the time of examination. All the issues reviewed had their associated actions documented and were submitted for approval to the issue reporter.

127. The process through which flaws, that were submitted through the online form or the bug bounty program, get copied into the public repository [6] remains manual. The process through which confidential flaws submitted through the GitLab platform are turned visible to the community also remains manual.

128. In the examines' opinion, the transparency objective is achieved. However, should the Federal Chancellery seek absolute transparency towards the public, limited assurance is provided to the public auditor. The multiplicity and design of the three vulnerability disclosure channels still allow Swiss Post and Cantons to retain complete control over when and how flaws become known to the public and how their severity gets ranked.

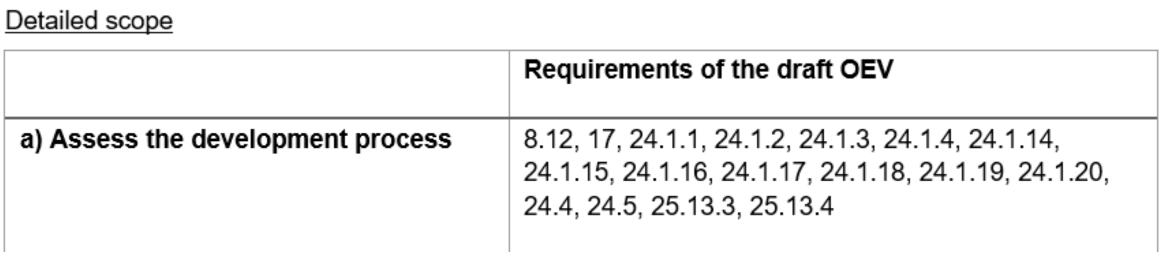
6.1.7 Systematic correction of flaws

129. Swiss Post proposes documented evidence of the systematic correction of flaws in the e-voting system in:

- » E-voting change management (reference not available)
- » Issue management - competence centre e-voting (reference not available)
- » Test concept of the Swiss Post voting system [32]
- » Release management concept for e-voting service [22]
- » E-voting QCV cheat sheet (reference not available)

130. For additional information, refer to "Appendix: Observations - Transparency".

6.2 Audit scope criteria



Detailed scope

	Requirements of the draft OEV
a) Assess the development process	8.12, 17, 24.1.1, 24.1.2, 24.1.3, 24.1.4, 24.1.14, 24.1.15, 24.1.16, 24.1.17, 24.1.18, 24.1.19, 24.1.20, 24.4, 24.5, 25.13.3, 25.13.4

Figure 5 - Audit scope 2a OEV requirements (source: Audit concept [5])

[25] Population: 14 issues, sample size: 7 (50%), selection method: odd issue identifiers.

6.3 Interview sessions log

Date	Objective	Participants
19.07.2021	Kick-off	- Project team (Swiss Post) - All scope 2 examinators
19.08.2021	General scope 2 information session (project organisation, standards and processes, systems architecture)	- Project team (Swiss Post) - All scope 2 examinators
03.09.2021	General scope 2 information session (evidence requirements and interview planning)	- Project team (Swiss Post) - All scope 2 examinators
19.11.2021	Scope 2a interview: development process and lifecycle	- Project manager + product owner + developer + SCRUM master (Swiss Post) - Scope 2a examinator
23.11.2021	Scope 2a interview: developer cross-examination	- Project manager + product owner + 2 backend developers + 1 frontend developer + SCRUM master (Swiss Post) - Scope 2a examinator
29.11.2021	Scope 2a interview: testing	- Project manager + product owner + developer + SCRUM master (Swiss Post) - Scope 2a examinator
01.12.2021	Scope 2a interview: quality assurance	- Project manager + product owner + developer + SCRUM master (Swiss Post) - Scope 2a examinator

Table 10 - Interview sessions log

6.4 Development process

The table below proposes a summarised description of the steps composing the e-voting development process [24], [25], [42]:

Action nr.	Description
1	Input change request
2	Review and assess change request
3	*Takeover by software development*
4	Transfer into backlog (story / feature / task input)
5	Formulate release vision (if not hotfix)
6	Formulate release roadmap (if not hotfix)
7	Update feature(s) / task(s)
8	Select task(s)

9	Assign task(s)
10	Reset code
11	Change task status
12	Create feature / hotfix / bugfix
13	Create branch in the e-voting repository
14	Checkout new branch
15	Develop feature
16	Run local build
17	Run local end2end tests
18	Document change request and/or changes made
19	Manage code change: - Rebase / merge, - Manage conflicts (optional), - Squash (optional), - Commit, - Push
20	Run unattended build and deployment test: - Build artifacts - Assess artifacts (composition analysis, quality and security static analysis) - Generate artifacts integrity hashes - Anonymize source code authorship - Deploy into containers
21	Test builds
22	Release builds
23	Transfer to software operations repository
24	Takeover by software operations department

Table 11 - Summary of Swiss Post e-voting development process steps

6.5 Reference evidence documents

Below, the inventory of evidence documents proposed by Swiss Post in the context of the examination of scope 2A (development processes), based on requirements set forth in the Federal Chancellery e-voting audit concept [5].

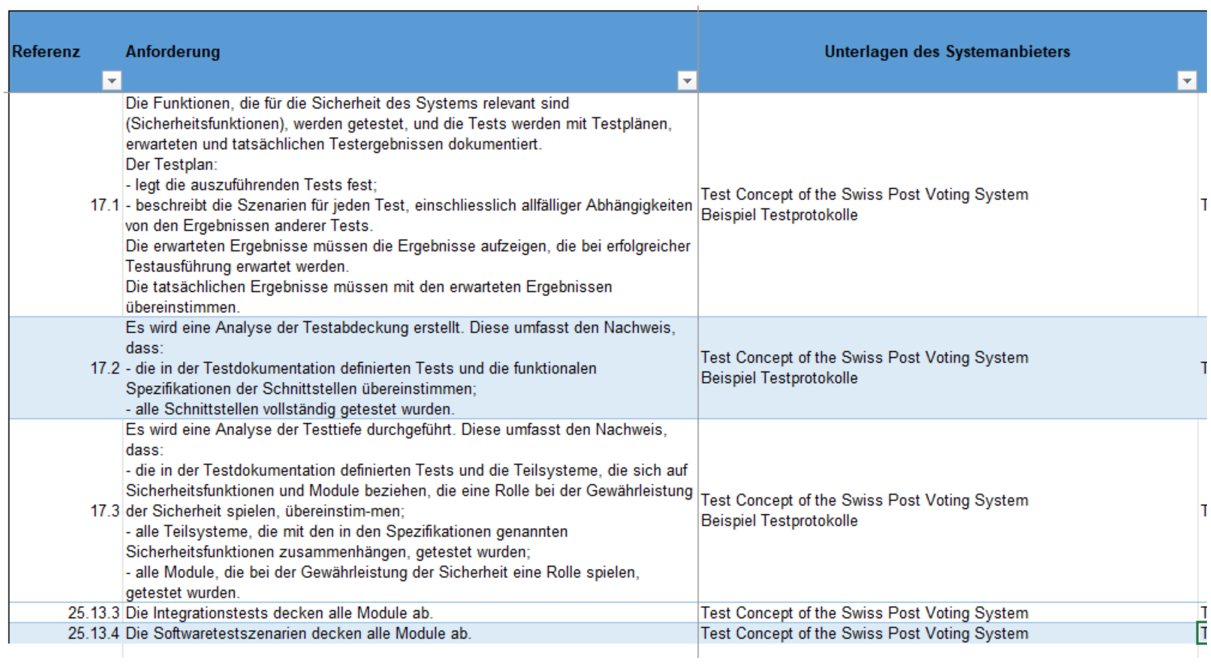
Topic: development process

Referenz	Anforderung	Unterlagen des Systemanbieters
24.1.1	Es wird ein Lebenszyklusmodell definiert. Das Lebenszyklusmodell: - wird für die Entwicklung und Wartung der Software verwendet; - sieht die notwendigen Kontrollen bei der Entwicklung und Wartung der Software vor; - wird dokumentiert.	Release Management Concept E-Voting Service
24.1.2	Es wird eine Liste der eingesetzten Entwicklungswerkzeuge sowie der Konfigurationsoptionen, die für den Einsatz der einzelnen Entwicklungswerkzeuge gewählt wurden, erstellt.	E-Voting Configuration Management Software Development Tools
24.1.3	Die Dokumentation der Entwicklungswerkzeuge umfasst: - eine Definition des Entwicklungswerkzeugs; - eine Beschreibung aller Konventionen und Richtlinien, die bei der Implementierung des Entwicklungswerkzeugs verwendet werden; - eine eindeutige Beschreibung der Bedeutung aller Konfigurationsoptionen für die Anwendung des Entwicklungswerkzeugs.	E-Voting Configuration Management Software Development Tools
24.1.4	Es wird festgelegt, welche Implementierungsstandards angewendet werden.	Software development process of the Swiss Post Voting System Software development process of the Swiss Post Voting System (Addition) ICT DEV - Architecture, Processes and Guidelines

Topic: Software security documentation

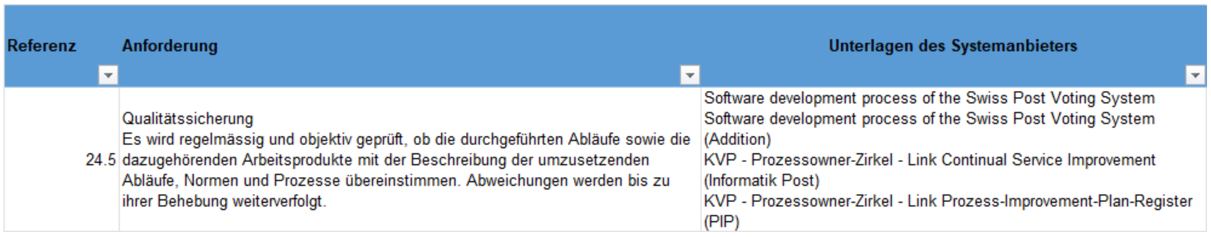
Referenz	Anforderung	Unterlagen des Systemanbieters
24.1.20	Die Dokumentation zur Sicherheit der Softwareentwicklung umfasst: - die Beschreibung der physischen, verfahrenstechnischen, personellen und sonstigen Sicherheitsmassnahmen, die zum Schutz und zur Integrität der Ausgestaltung und der Implementierung der Software in ihrer Entwicklungsumgebung erforderlich sind; - den Nachweis, dass die Sicherheitsmassnahmen das erforderliche Schutzniveau bieten, um die Integrität der Software zu wahren.	Whitepaper Infrastructure of the Swiss Post Voting System Software development process of the Swiss Post Voting System Software development process of the Swiss Post Voting System (Addition) Trusted Build of the Swiss Post Voting System Release Management Concept E-Voting Service E-Voting Architecture documentation of the Swiss Post Voting System E-Voting ISDS Konzept E-Voting Risikoportfolio E-Voting Risikoanalyse E-Voting Risikomanagement

Topic: testing



Referenz	Anforderung	Unterlagen des Systemanbieters	
17.1	Die Funktionen, die für die Sicherheit des Systems relevant sind (Sicherheitsfunktionen), werden getestet, und die Tests werden mit Testplänen, erwarteten und tatsächlichen Testergebnissen dokumentiert. Der Testplan: - legt die auszuführenden Tests fest; - beschreibt die Szenarien für jeden Test, einschliesslich allfälliger Abhängigkeiten von den Ergebnissen anderer Tests. Die erwarteten Ergebnisse müssen die Ergebnisse aufzeigen, die bei erfolgreicher Testausführung erwartet werden. Die tatsächlichen Ergebnisse müssen mit den erwarteten Ergebnissen übereinstimmen.	Test Concept of the Swiss Post Voting System Beispiel Testprotokolle	T
17.2	Es wird eine Analyse der Testabdeckung erstellt. Diese umfasst den Nachweis, dass: - die in der Testdokumentation definierten Tests und die funktionalen Spezifikationen der Schnittstellen übereinstimmen; - alle Schnittstellen vollständig getestet wurden.	Test Concept of the Swiss Post Voting System Beispiel Testprotokolle	T
17.3	Es wird eine Analyse der Testtiefe durchgeführt. Diese umfasst den Nachweis, dass: - die in der Testdokumentation definierten Tests und die Teilsysteme, die sich auf Sicherheitsfunktionen und Module beziehen, die eine Rolle bei der Gewährleistung der Sicherheit spielen, übereinstim-men; - alle Teilsysteme, die mit den in den Spezifikationen genannten Sicherheitsfunktionen zusammenhängen, getestet wurden; - alle Module, die bei der Gewährleistung der Sicherheit eine Rolle spielen, getestet wurden.	Test Concept of the Swiss Post Voting System Beispiel Testprotokolle	T
25.13.3	Die Integrationstests decken alle Module ab.	Test Concept of the Swiss Post Voting System	T
25.13.4	Die Softwaretestszenarien decken alle Module ab.	Test Concept of the Swiss Post Voting System	T

Topic: quality assurance



Referenz	Anforderung	Unterlagen des Systemanbieters
24.5	Qualitätssicherung Es wird regelmässig und objektiv geprüft, ob die durchgeführten Abläufe sowie die dazugehörenden Arbeitsprodukte mit der Beschreibung der umzusetzenden Abläufe, Normen und Prozesse übereinstimmen. Abweichungen werden bis zu ihrer Behebung weiterverfolgt.	Software development process of the Swiss Post Voting System Software development process of the Swiss Post Voting System (Addition) KVP - Prozessowner-Zirkel - Link Continual Service Improvement (Informatik Post) KVP - Prozessowner-Zirkel - Link Prozess-Improvement-Plan-Register (PIP)

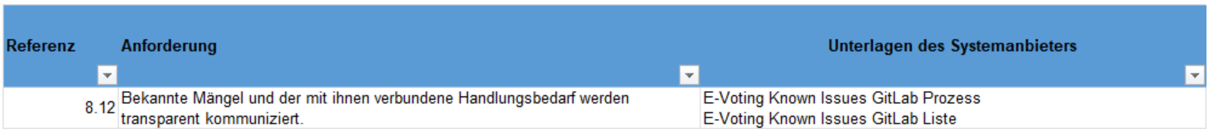
Topic: configuration management system

Referenz	Anforderung	Unterlagen des Systemanbieters
24.1.14	Die Software wird mit einer eindeutigen Kennzeichnung versehen.	Release Management Concept E-Voting Service Source Code
24.1.15	Die Dokumentation des Konfigurationsmanagements enthält: - eine Beschreibung, wie Konfigurationselemente identifiziert werden; - einen Konfigurationsmanagementplan, der beschreibt, wie das Konfigurationsmanagementsystem bei der Entwicklung der Software eingesetzt wird und welche Verfahren für die Übernahme von Änderungen oder neuen Elementen angewendet werden; - einen Nachweis, dass die Verfahren für die Übernahme eine angemessene Prüfung der Änderungen für alle Konfigurationselemente vorsehen.	E-Voting Configuration Management Software Application & Infrastructure
24.1.16	Das Konfigurationsmanagementsystem: - identifiziert alle Konfigurationselemente eindeutig; - stellt automatisierte Massnahmen bereit, damit nur autorisierte Änderungen an Konfigurationselementen vorgenommen werden; - unterstützt die Entwicklung der Software durch automatisierte Verfahren; - stellt sicher, dass die Person, die für die Abnahme des Konfigurationselements verantwortlich ist, nicht dieselbe Person ist, die es entwickelt hat; - identifiziert die Konfigurationselemente, aus denen sich die Sicher-heitsfunktionen zusammensetzen; - unterstützt die Prüfung aller Änderungen an der Software mit auto-matisierten Verfahren, einschliesslich der Protokollierung des Verfassers sowie des Datums und der Uhrzeit der Änderung; - stellt ein automatisiertes Verfahren zur Identifizierung aller Konfigurationselemente bereit, die von einer Änderung an einem be-stimmten Konfigurationselement betroffen sind; - kann die Version des Quellcodes identifizieren, auf dessen Basis die Software generiert wird.	E-Voting Configuration Management Software Application & Infrastructure
24.1.17	Alle Konfigurationselemente werden im Konfigurationsmanagementsystem inventarisiert.	E-Voting Configuration Management Software Application & Infrastructure
24.1.18	Das Konfigurationsmanagementsystem wird in Übereinstimmung mit dem Konfigurationsmanagementplan verwendet.	E-Voting Configuration Management Software Application & Infrastructure
24.1.19	Es wird eine Konfigurationsliste erstellt, die die folgenden Elemente enthält: - die Software; - Nachweise der erforderlichen Überprüfungen zur Einhaltung der Sicherheit; - die Teile, aus denen die Software besteht; - den Quellcode; - Berichte über Sicherheitsmängel und über den Stand der Behebung. Für jedes Element, das für Sicherheitsfunktionen relevant ist, wird die Entwicklerin oder der Entwickler genannt. Jedes Element wird eindeutig identifiziert.	E-Voting Configuration Management Software Application & Infrastructure

Topic: testing

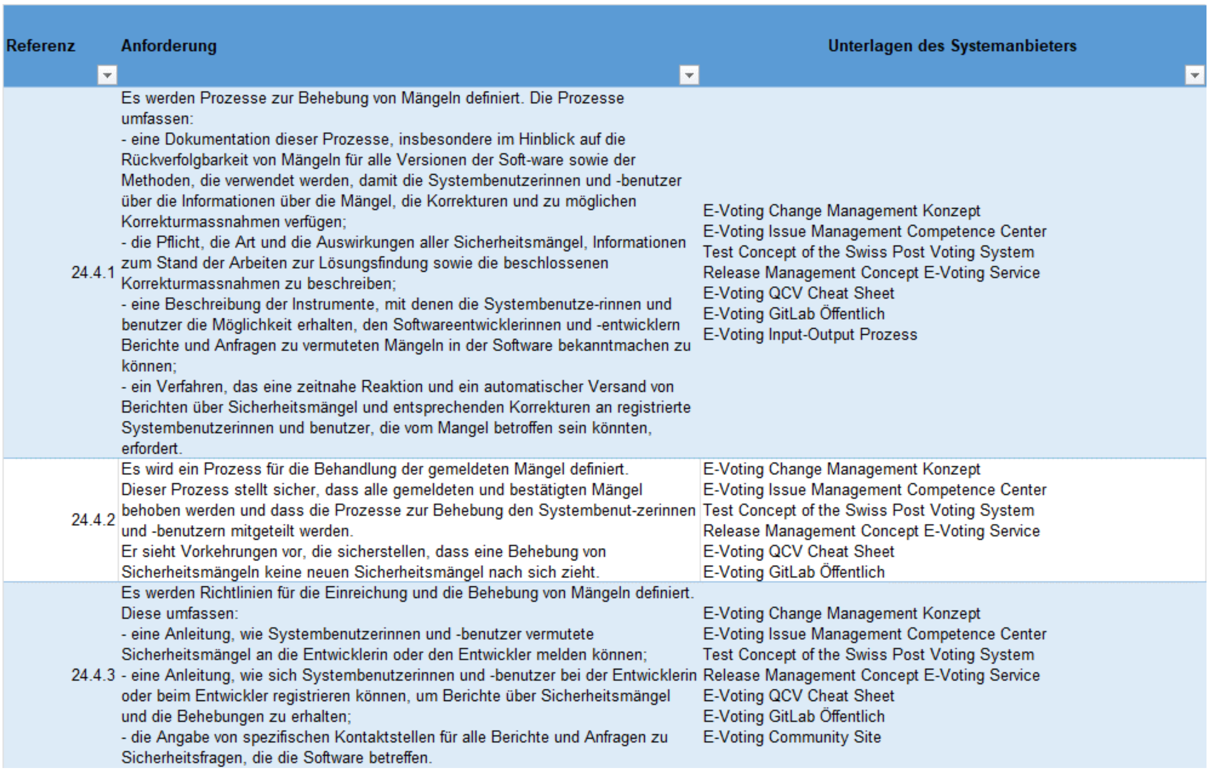
Referenz	Anforderung	Unterlagen des Systemanbieters
17.1	Die Funktionen, die für die Sicherheit des Systems relevant sind (Sicherheitsfunktionen), werden getestet, und die Tests werden mit Testplänen, erwarteten und tatsächlichen Testergebnissen dokumentiert. Der Testplan: - legt die auszuführenden Tests fest; - beschreibt die Szenarien für jeden Test, einschliesslich allfälliger Abhängigkeiten von den Ergebnissen anderer Tests. Die erwarteten Ergebnisse müssen die Ergebnisse aufzeigen, die bei erfolgreicher Testausführung erwartet werden. Die tatsächlichen Ergebnisse müssen mit den erwarteten Ergebnissen übereinstimmen.	Test Concept of the Swiss Post Voting System Beispiel Testprotokolle
17.2	Es wird eine Analyse der Testabdeckung erstellt. Diese umfasst den Nachweis, dass: - die in der Testdokumentation definierten Tests und die funktionalen Spezifikationen der Schnittstellen übereinstimmen; - alle Schnittstellen vollständig getestet wurden.	Test Concept of the Swiss Post Voting System Beispiel Testprotokolle
17.3	Es wird eine Analyse der Testtiefe durchgeführt. Diese umfasst den Nachweis, dass: - die in der Testdokumentation definierten Tests und die Teilsysteme, die sich auf Sicherheitsfunktionen und Module beziehen, die eine Rolle bei der Gewährleistung der Sicherheit spielen, übereinstim-men; - alle Teilsysteme, die mit den in den Spezifikationen genannten Sicherheitsfunktionen zusammenhängen, getestet wurden; - alle Module, die bei der Gewährleistung der Sicherheit eine Rolle spielen, getestet wurden.	Test Concept of the Swiss Post Voting System Beispiel Testprotokolle
25.13.3	Die Integrationstests decken alle Module ab.	Test Concept of the Swiss Post Voting System
25.13.4	Die Softwaretestszenarien decken alle Module ab.	Test Concept of the Swiss Post Voting System

Topic: transparency



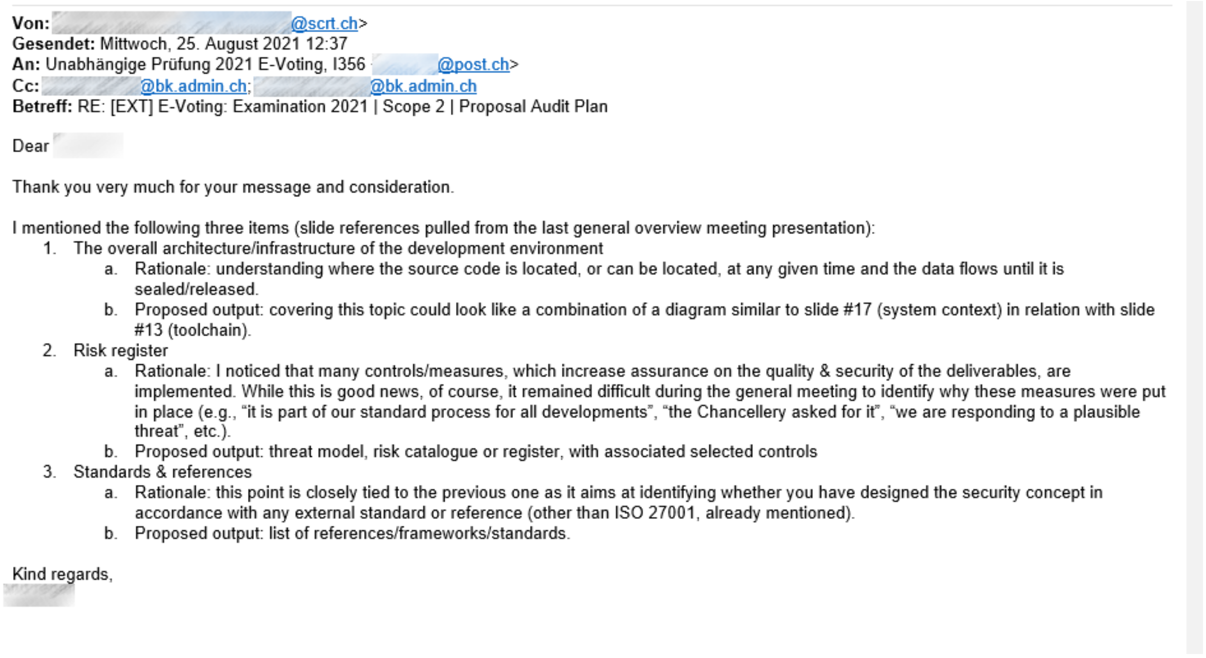
Referenz	Anforderung	Unterlagen des Systemanbieters
8.12	Bekannte Mängel und der mit ihnen verbundene Handlungsbedarf werden transparent kommuniziert.	E-Voting Known Issues GitLab Prozess E-Voting Known Issues GitLab Liste

Topic: systematic correction of flaws



Referenz	Anforderung	Unterlagen des Systemanbieters
24.4.1	Es werden Prozesse zur Behebung von Mängeln definiert. Die Prozesse umfassen: - eine Dokumentation dieser Prozesse, insbesondere im Hinblick auf die Rückverfolgbarkeit von Mängeln für alle Versionen der Soft-ware sowie der Methoden, die verwendet werden, damit die Systembenutzerinnen und -benutzer über die Informationen über die Mängel, die Korrekturen und zu möglichen Korrekturmassnahmen verfügen; - die Pflicht, die Art und die Auswirkungen aller Sicherheitsmängel, Informationen zum Stand der Arbeiten zur Lösungsfindung sowie die beschlossenen Korrekturmassnahmen zu beschreiben; - eine Beschreibung der Instrumente, mit denen die Systembenutze-rinnen und benutzer die Möglichkeit erhalten, den Softwareentwicklerinnen und -entwicklern Berichte und Anfragen zu vermuteten Mängeln in der Software bekanntmachen zu können; - ein Verfahren, das eine zeitnahe Reaktion und ein automatischer Versand von Berichten über Sicherheitsmängel und entsprechenden Korrekturen an registrierte Systembenutzerinnen und benutzer, die vom Mangel betroffen sein könnten, erfordert.	E-Voting Change Management Konzept E-Voting Issue Management Competence Center Test Concept of the Swiss Post Voting System Release Management Concept E-Voting Service E-Voting QCV Cheat Sheet E-Voting GitLab Öffentlich E-Voting Input-Output Prozess
24.4.2	Es wird ein Prozess für die Behandlung der gemeldeten Mängel definiert. Dieser Prozess stellt sicher, dass alle gemeldeten und bestätigten Mängel behoben werden und dass die Prozesse zur Behebung den Systembenut-zerinnen und -benutzern mitgeteilt werden. Er sieht Vorkehrungen vor, die sicherstellen, dass eine Behebung von Sicherheitsmängeln keine neuen Sicherheitsmängel nach sich zieht.	E-Voting Change Management Konzept E-Voting Issue Management Competence Center Test Concept of the Swiss Post Voting System Release Management Concept E-Voting Service E-Voting QCV Cheat Sheet E-Voting GitLab Öffentlich
24.4.3	Es werden Richtlinien für die Einreichung und die Behebung von Mängeln definiert. Diese umfassen: - eine Anleitung, wie Systembenutzerinnen und -benutzer vermutete Sicherheitsmängel an die Entwicklerin oder den Entwickler melden können; - eine Anleitung, wie sich Systembenutzerinnen und -benutzer bei der Entwicklerin oder beim Entwickler registrieren können, um Berichte über Sicherheitsmängel und die Behebungen zu erhalten; - die Angabe von spezifischen Kontaktstellen für alle Berichte und Anfragen zu Sicherheitsfragen, die die Software betreffen.	E-Voting Change Management Konzept E-Voting Issue Management Competence Center Test Concept of the Swiss Post Voting System Release Management Concept E-Voting Service E-Voting QCV Cheat Sheet E-Voting GitLab Öffentlich E-Voting Community Site

6.6 Request for additional information on risk management



Von: [redacted]@scrt.ch>
Gesendet: Mittwoch, 25. August 2021 12:37
An: Unabhängige Prüfung 2021 E-Voting, I356 [redacted]@post.ch>
Cc: [redacted]@bk.admin.ch; [redacted]@bk.admin.ch
Betreff: RE: [EXT] E-Voting: Examination 2021 | Scope 2 | Proposal Audit Plan

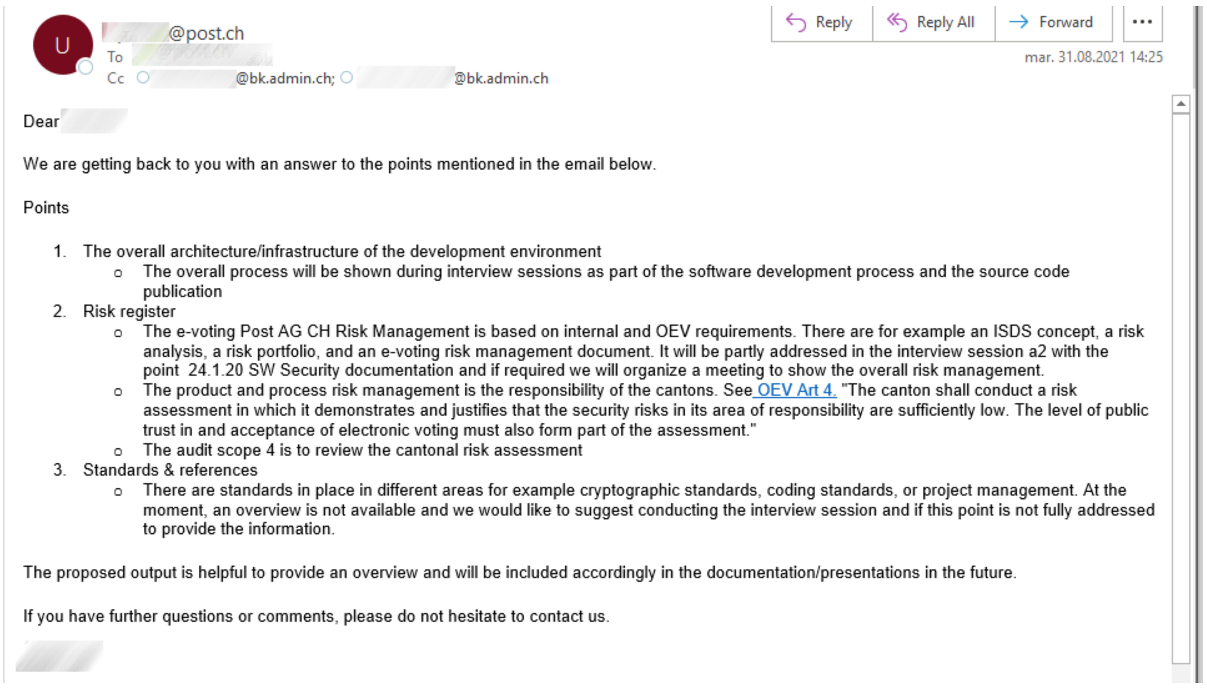
Dear [redacted]

Thank you very much for your message and consideration.

I mentioned the following three items (slide references pulled from the last general overview meeting presentation):

1. The overall architecture/infrastructure of the development environment
   a. Rationale: understanding where the source code is located, or can be located, at any given time and the data flows until it is sealed/released.
   b. Proposed output: covering this topic could look like a combination of a diagram similar to slide #17 (system context) in relation with slide #13 (toolchain).
2. Risk register
   a. Rationale: I noticed that many controls/measures, which increase assurance on the quality & security of the deliverables, are implemented. While this is good news, of course, it remained difficult during the general meeting to identify why these measures were put in place (e.g., "it is part of our standard process for all developments", "the Chancellery asked for it", "we are responding to a plausible threat", etc.).
   b. Proposed output: threat model, risk catalogue or register, with associated selected controls
3. Standards & references
   a. Rationale: this point is closely tied to the previous one as it aims at identifying whether you have designed the security concept in accordance with any external standard or reference (other than ISO 27001, already mentioned).
   b. Proposed output: list of references/frameworks/standards.

Kind regards,
[redacted]



[redacted]@post.ch
To [redacted]
Cc [redacted]@bk.admin.ch; [redacted]@bk.admin.ch
mar. 31.08.2021 14:25

Dear [redacted]

We are getting back to you with an answer to the points mentioned in the email below.

Points

1. The overall architecture/infrastructure of the development environment
   - The overall process will be shown during interview sessions as part of the software development process and the source code publication
2. Risk register
   - The e-voting Post AG CH Risk Management is based on internal and OEV requirements. There are for example an ISDS concept, a risk analysis, a risk portfolio, and an e-voting risk management document. It will be partly addressed in the interview session a2 with the point 24.1.20 SW Security documentation and if required we will organize a meeting to show the overall risk management.
   - The product and process risk management is the responsibility of the cantons. See OEV Art 4. "The canton shall conduct a risk assessment in which it demonstrates and justifies that the security risks in its area of responsibility are sufficiently low. The level of public trust in and acceptance of electronic voting must also form part of the assessment."
   - The audit scope 4 is to review the cantonal risk assessment
3. Standards & references
   - There are standards in place in different areas for example cryptographic standards, coding standards, or project management. At the moment, an overview is not available and we would like to suggest conducting the interview session and if this point is not fully addressed to provide the information.

The proposed output is helpful to provide an overview and will be included accordingly in the documentation/presentations in the future.

If you have further questions or comments, please do not hesitate to contact us.

[redacted]

6.7 Interview plan - development process (additional criteria) + developers

Stage 0: introduction

» Inform developers of the objective of the interview.
» Inform developers that the interview will remain anonymous, only the date and observations collected during the session will be publicly reported.
» Confirm with developers that they are informed of the context of the examination and that their answers may be publicly disclosed.

Stage 1: role and responsibilities

» Q: What is your role and position in the e-voting project?
» Q: What specific components/projects do you work on?
» Q: Can other developers in your team work on the same code as you do?
» Q: Can you take a 4-weeks long leave of absence (e.g., accident, parental/maternity leave, etc.) without detrimental impact to the team or the project?

Stage 2: control question

» Q: How do you ensure that your deliverables do not contain vulnerabilities or errors?

Stage 3: threat awareness

» Q: What threat are you dealing with, as a developer of the voting system?
» Q: How do you respond to / manage these threats?

Stage 4: format security training

» Q: What forms of training did you receive in relation with secure development?
» Q: What topics / syllabus did you cover during this/these training(s)?

Stage 5: tooling

- Q: What tools are at your disposal to help you write you better/more secure code?
- Q: Do you have access to security experts or specialists that can help you on software engineering security? How/who?
- Q: Are you given access to an enterprise level security API or library?

Stage 6: dev workstation

- Q: From what locations can you work and submit source code? (e.g., office, home, etc.)
- Q: Can you work on the code from an unmanaged device? (e.g., private computer)
- Q: How do you log into the source code repository?
- Q: Are there particular rules you must comply with when working remotely?

Stage 7: process

- Q: What sort of security-related information do you receive when processing a work item?
- Q: Are there rules or standards you must adhere to?
- Q: Are there security constraints enforced onto your development tools (e.g., banned functions, commit locks, local static scanning, etc.)?
- Q: Do you attend threat modelling sessions or equivalent?
- Q: How do you reduce the risk of writing vulnerable code?
- Q: Can you introduce third-party libraries or components in your code? If yes, how?
- Q: What do you verify/validate when committing code or reviewing someone else's?
- Q: How is your code tested for vulnerabilities / errors?
- Q: Is your code reviewed or tested by security-focused people or tools?

6.8 Interview plan - testing and quality requirements

- Q: How are security relevant functions identified and where?
- Q: Which tests are always performed?
- Q: Which tests are sometimes performed?
- Q: How are abuse cases / attacker stories implemented?
- Q: How are tests specified?
- Q: What are the expected results of these tests? Are they specified?
- Q: What is the current testing coverage? How is it measured?
- Q: What is tested precisely?
- Q: What tools do you use for testing?
- Q: What customizations were made in the testing tools?
- Q: How are new design proposals tested for vulnerabilities or errors?
- Q: How is the source code tested for vulnerabilities or errors?
- Q: How are deliverables tested for vulnerabilities or errors?
- Q: Who performs security tests?
- Q: How is the final platform security tested?
- Q: What happens when a defect is identified?
- Q: What determines if a defect is a security defect?
- Q: How are security tests reported?

- Q: Who approves these reports?
- Q: Is there anything you wish you were testing but are not yet testing?

6.9 Interview plan - quality assurance requirements

- Q: How do you review and improve your development process?
- Q: Which third-party standards or references do you rely on to define your development process?
- Q: How do you verify that your security testing tools work as expected?