Recommendations for Implementing Independent Individual Verifiability in Internet Voting

Florian Moser[1], Rüdiger Grimm[2], Tobias Hilt[3], Michael Kirsten[3], Christoph Niederbudde[3], and Melanie Volkamer[3]

Abstract: End-to-end verifiable systems are employed to safeguard the integrity of Internet voting. Voter-initiated verification for *individual verifiability* require that the ballot formed on the voter's device is audited on a second device, which is independent of a potentially manipulated voter's device. Further trust is gained by executing the verification procedure on a second device with independent implementations, in order to defend against a dishonest primary system operator. This paper formulates recommendations to implement such independent individual verifiability tools. Our recommendations are based on the experiences made in the GI elections 2023 where such independent tools were made available to the voters – to our knowledge the first project of its kind.

Keywords: Internet Voting, Individual Verifiability, Second Device

1 Introduction

Internet voting promises to improve access to elections, while reducing administrative effort, but also poses new challenges regarding security and trustworthiness of elections. To safeguard election integrity, end-to-end verifiable voting systems provide artifacts to independent parties to verify that the election proceeded correctly under given trust assumptions. End-to-end verifiability can be classified into *universal verifiability*, where anyone may verify, e. g., whether the votes are tallied as recorded, and *individual verifiability*, where voters verify individually, e. g., whether their vote is cast as intended. These checks must preserve the secrecy of the election and be performed under realistic trust assumptions.

Many approaches for individual verifiability perform an audit over the formed ballot. These approaches can be classified into *audit-or-cast* (also called the Benaloh challenge [Be87]), where the voter chooses to either audit or cast their ballot, and *cast-and-audit*, where the voter audits the actually cast ballot. It is crucial that the audit respects the given trust assumptions. To avoid full trust in the device which forms the ballot, the audit needs to be executed on a second, separate device. Further, to avoid full trust in the main system provider, the code running on the second device needs to be written independently and provided to the second device untampered (e. g., on servers operated by independent parties).

1 INRIA Nancy, France, florian.moser@inria.fr, https://orcid.org/0000-0003-2268-2367
2 Fraunhofer SIT, Darmstadt, Germany, grimm@uni-koblenz.de, https://orcid.org/0009-0005-5481-8419
3 KASTEL Security Research Labs, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, tobias.hilt@kit.edu, https://orcid.org/0000-0001-9267-5109; kirsten@kit.edu, https://orcid.org/0000-0001-9816-1504; udqps@student.kit.edu; melanie.volkamer@kit.edu, https://orcid.org/0000-0003-2674-4043

In this work, we formulate recommendations to implement such independent individual verifiability tools. These recommendations allow building upon the experiences we made, to continue what worked well, and at the same time to avoid potential pitfalls.

We base our recommendations on the experiences made developing and providing such tools for the German Informatics Society (GI) elections of 2023 using the Polyas system. In this election, three independently developed and hosted tools[4] allowed voters to perform individual verifiability checks on their second device. To the best of our knowledge, this is the first project of this kind, namely that the election organizer provided independently implemented individual verifiability tools to their voters as part of end-to-end verifiable Internet voting.

Related Work. Cortier et al. [Co23] report on a similar verifiability project for voting from abroad in the 2022 legislative elections in France, yet with a much more limited notion of verifiability and overall no end-to-end verifiability. Another related endeavor is Microsoft's open community project ElectionGuard [Be24], which – while mainly targeting in-place elections – also provides end-to-end verifiable voting and accompanied multiple pilot elections, e. g., the College Park 2023 General Election[5] with independent individual verifiability tools. Further note that, while not specifically reporting on similar projects, the Swiss Post System [Sw24], CHVote [Ha23] and Belenios [CGG19] are also extensively publicly documented end-to-end-verifiable voting systems, which allow implementing independent verifiability tools.

Moreover, we are aware of two other works that formulate recommendations within the development of Internet voting applications, which however have different foci. Hänni et al. [Ha20] give recommendations from the point of view of system developers, while Haines and Rønne [HR21] take the perspective of source code examiners. While these perspectives are quite different from ours (we neither design nor review the main system), some of our observations overlap with their reports, especially concerning the specification. As Hänni et al. describe election verifiers as "the ultimate way of challenging the protocol run", our hands-on experiences strengthen their findings. The recommendations by Haines and Rønne, however, might be possible next steps within the development process, if we were to have access to the source code of the voting system.

2 Context

We provide context to the independent individual verifiability tool implementations, which helps to understand the challenges faced by such a project.

4 See `https://github.com/kastel-security/polyas-core3-second-device-verification`, `https://github.com/famoser/polyas-verification`, and Koppenhöfer [Ko23] for the individual tools.

5 `https://www.electionguard.vote/elections/College_Park_Maryland_2023/`

2.1 Polyas Individual Verifiability Mechanism

The following gives a simplified overview of the individual verifiability mechanism used for the 2023 GI elections. We forgo a security analysis here and instead refer to Müller et al. [MT23] and the official specification [Tr24] for details.

After casting their ballot, the voter is presented with the available individual verifiability tools. For each tool, a QR code and a PIN (which changes every 30 seconds) is displayed. The QR code consists of a URL to the verifiability tool, which includes the voter ID and some randomness r. The voter then reads out the QR code on their second device (e. g., a smartphone), which loads the verifiability tool (e. g., by opening a webpage in the browser). The voter then enters the currently displayed PIN, and the tool downloads the voter's ballot and a re-encryption of that ballot from the server. Then, the tool performs an interactive zero-knowledge proof of correct re-encryption with the server over the two ballots (to assert that the ballot and its re-encryption encrypt the same plaintext). If successful, the tool decrypts the re-encrypted ballot using the randomness r from the QR code. The voter is shown this plain vote and the internal voter ID of the ballot, which the voter checks for correctness, or complains if incorrect. Further, the voter may download a receipt of the ballot signed by the server, which they can send to the election organizer for universal verifiability (to assert that all ballots are in the tally and no invalid ones were added). The verification window is limited to 30 minutes, after which the server refuses to participate. Further, the individual verifiability tool allows the voter to modify the displayed plain vote (without impacting the to-be-tallied ballot), rendering screenshots of this view unconvincing.

2.2 GI Experience with Verifiable Elections

Since 2004, the annual presidential and board elections of the "GI – Gesellschaft für Informatik" (German Informatics Society) are performed in a hybrid form, online with Polyas and by postal voting. In 2008, the GI developed a protection profile for a basic set of security requirements for online voting products (CC-PP-0037-2008, which is today outdated[6]), and since 2010 approached independent verifiability [OSV11]. However, such certifications are limited [BNV14], as well as the particular approaches taken for independent verifiability [Ol12; OSV11].

In 2019, the GI switched to a new Polyas system, which provided cryptographic artifacts for universal verifiability, and could be extended towards full end-to-end verifiability. GI researchers of the Karlsruhe Institute of Technology (KIT) and University of Stuttgart developed universal verifiability tools, which were successfully used since the GI election 2019 [Be19; Be21], and formally verified parts of the system's software [Ki22]. For the 2023 elections, the GI decided to additionally use the individual verifiability mechanism provided by Polyas to achieve full end-to-end verifiability. We describe this project in Sect. 2.3.

6 https://www.bsi.bund.de/SharedDocs/Zertifikate_CC/PP/Archiv/PP_0037.html

2.3 Project Timeline

For the 2023 elections, the GI envisioned providing independent individual verifiability tools. Fig. 1 illustrates the most important milestones throughout the different phases of the project, which we briefly explain in the following.

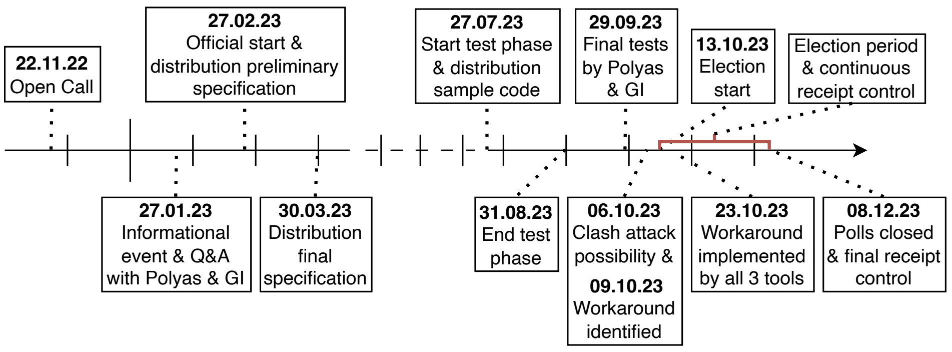
22.11.22
Open Call
27.02.23
Official start & distribution preliminary specification
27.07.23
Start test phase & distribution sample code
29.09.23
Final tests by Polyas & GI
13.10.23
Election start
Election period & continuous receipt control
27.01.23
Informational event & Q&A with Polyas & GI
30.03.23
Distribution final specification
31.08.23
End test phase
06.10.23
Clash attack possibility &
09.10.23
Workaround identified
23.10.23
Workaround implemented by all 3 tools
08.12.23
Polls closed & final receipt control

Fig. 1: Timeline of the most relevant milestones of the project.

Development Phase. In November 2022, the GI distributed a call[7] to develop an independent individual verifiability tool. The workload was described as appropriate for a thesis or a semester project for a computer science student. In January 2023, seven interested parties met with the GI and Polyas to outline tasks and schedules. The official start of the project was in February 2023, when the participants settled on a time schedule, and received the preliminary specification. In July 2023, the tools were almost finished. To help resolve the remaining issues, Polyas provided updated specifications, additional test data and the source code to their own individual verifiability tool. Further, access to a test election on their main voting system was granted. In September 2023, the tools were finished, and Polyas provided additional test elections for the final tests.

Pre-Voting Phase. In September 2023, the Polyas voting system and the verifiability tools were released for use by the GI. Subsequently, the election information and credentials were sent to the voters by postal mail.

However, after sending the letters, but before the voting phase had begun, the GI was made aware that the individual verifiability as implemented is susceptible against a clash attack. While each ballot contains an internal voter ID which binds the ballot to a specific voter, this internal voter ID was not given to the voters, hence the voters could not check that they indeed verify their own ballots. The GI found a workaround without sending out new election letters: The individual verifiability tools additionally print the internal voter ID

7 `https://gi.de/wahlen/verifikation-der-gi-wahlen-tools-gesucht`

on the receipt. When the voter sends this receipt to the GI, it is then verified by the GI whether the sender corresponds to the internal voter ID on the receipt. This process was communicated to the GI members via mail, together with the general explanation about the newly added cast-as-intended verifiability.

Voting Phase. The voting phase was held from October 13 until December 8, 2023. Three independent individual verifiability tools were available, plus the one provided by Polyas. The voters could freely choose which verifiability tool(s) they want to use. Out of the 2,759 Internet voters, 689 (25%) used at least one of the available tools. 72 voters sent their verification receipt to the GI, from which 30 included the internal voter ID checked by the election board.

Tally Phase. After the end of the voting phase, the votes were tallied using the official counting service of Polyas. In this phase, the available universal verifiability tools first introduced in 2019 [Be19; Be21; Ki22] were reused, now extended to check that the ballots from the receipts are included in the tally.

3 Experience Report and Recommendations

We share our experiences and formulate recommendations to implement independent individual verifiability. We categorize in implementation, testing, quality, and responsibility topics, each consisting of several sub-topics. Each (sub-)topic is introduced, and corresponding recommendations are given. While the recommendations are given independently to the GI project, to motivate and give context, we interleave — in separate paragraphs — with concrete experiences made as part of the GI project.

Concerning the experiences made, while in this section we primarily elaborate on issues and potential areas for improvement, we note that overall, the GI project was successful. Despite the inherent complexity, several independent implementations were created, deployed, and used by voters in the real election.

3.1 Implementation

An individual verifiability tool needs to implement cryptographic algorithms, the message exchange between devices and servers, and a user interface.

Polyas provided two specifications:[8] The main specification [Tr23] describes the high-level protocol, intended security guarantees, algorithms, and serializations needed to

8 The specifications [Tr23; Tr24] only partially describe the protocol of the voting system; e. g., it remains unspecified how the ballot is encrypted on the voting device.

implement the verification. The second device specification [Tr24] describes the second device protocol, including the requests sent and received, while referring for the algorithms and serializations to the main specification. Polyas further provided code of their own second device implementation.

Algorithms. For Internet voting, cryptographic algorithms include domain-specific signatures, encryptions and zero-knowledge proofs, as well as auxiliary algorithms, e. g., pseudo-random generators, key-derivation functions, and message encoding. Available cryptographic libraries usually only cover parts of the necessary algorithms and often lack polish (e. g., insufficient documentation, unintuitive APIs, bugs).[9] A high-quality specification is, therefore, indispensable.

In the Polyas specification, parameters and return values are specified for every algorithm, as well as the algorithm itself as pseudocode or precise text. Complex algorithms are decomposed into multiple algorithms building upon each other. For most algorithms, the specification gives their context of use and some examples with input and output. Overall, the notation is consistent, and the presentation is compact. All these factors together contributed, such that in this part of the project, only few issues were encountered.

Algorithms

Clarity. Make parameters and return values explicit for each algorithm, and specify their implementation very clearly or use pseudo code.

Decomposition. Divide complex algorithms into smaller algorithms with a clear functionality, which can then build upon each other.

Context. Give context on the algorithm's usage, in particular by illustrating examples for input and output.

Serialization. Serialization and deserialization, respectively, define the conversion of an in-memory object (e. g., a public key) into a format to store it to a file, or send it over the network (e. g., as JSON). For interoperability of libraries and languages, this format needs to be precisely described, which should follow a clearly defined and ubiquitous standard.

In the main specification, Polyas specifies general serialization formats (e. g., the compressed form of ANSI X9.62 4.3.6 for representing elliptic curve points), which are implicitly used in the second device specification. Besides the specification, the source code by Polyas also provides insights for implementers. In this project, resolving serialization issues was

9 For example, one of the implementers found and fixed an issue in PHP itself within one of the second device implementations (parsing a public key of a particular format produced an error, even if successfully parsed, see `https://github.com/php/php-src/pull/11055`). Finding issues in the programming language itself is very unusual and indicates that such APIs are rarely used.

responsible for up to half of the total effort of implementing the cryptography.[10] We give some examples here:

- **Java's `BigInteger` Encoding.** The specification describes the conversion of large integers to bytes by calling the function `toByteArray()` of Java's `BigInteger`. `BigInteger` considers integers as *signed*, with the most-significant bit indicating whether a number is positive or negative. However, natural serializations in JavaScript and PHP implementations omit this bit.[11]
- **Url-Safe Base64 Encoding.** The second device specification references a `decodeBase64()` function, but the sample data contains characters that are not part of a standard base64 encoding. Inspecting the Polyas source code revealed the usage of a URL-safe variant, which replaces `+/` by `-_`.
- **PDF Receipt Encoding.** The second device implementation generates a PDF receipt with cryptographic values, which are read out again automatically during the tally phase for verification. However, it was not clearly defined how these values need to be written to the PDF file, and different PDF writers generally use different approaches.[12] As a result, many receipts could not be read out automatically by all tools during the tally phase.

Serialization

Fit-for-Purpose. Chosen serialization and deserialization need to be suitable for their respective use-case.

Standards. Every serialization and deserialization needs to specify which technology-independent and widespread standard is employed.

Protocol. The protocol defines how algorithms are executed and messages are exchanged to achieve the security guarantees under the chosen adversary model. For individual verifiability tools, this typically involves the voter, who exchanges messages with their voting device or with the election server. The protocol, the security guarantees, and the adversary model need to be clearly described to the implementers for ensuring their implementation does not sabotage the protocol's goals (e. g., by accidentally publishing values assumed to remain secret).

10 The diagnosis and repair of serialization and deserialization issues is time-consuming and difficult. Both usually involve extensive engagement with code, library and language, as well as studying appropriate standards.

11 See, e. g., the PHP replication of the Java implementation, which prepends another zero-byte after the PHP-natural serialization in order to simulate the most-significant signed bit: `https://github.com/famoser/polyas-verification/blob/v1.3.2/src/Crypto/POLYAS/Utils/Serialization.php#L27`.

12 In PDF, text is usually printed using the `Tj` function. By extracting the arguments to this function (e. g., using a regular expression), one can retrieve the printed text (e. g., `https://github.com/famoser/polyas-verification/blob/v1.3.2/src/Workflow/VerifyReceipt.php#L77`). However, this approach does not work in general, for example some PDF writers use compression or a custom encoding.

The Polyas second device specification describes the protocol, but it does not clarify the security claims and the adversary model. Instead, the specification references an academic publication [MT23] that describes a cast-as-intended mechanism which is the basis for the second device protocol. Neither having the full protocol specification of the system nor a security analysis with the adversary model and its security claims, we encountered the following issues:

- **Inconsistency of Specification with Published Protocol.** The second device specification is inconsistent with the referenced academic publication: In the implemented Polyas system, a performance optimization for big ballots is applied on the voter's device, which requires structural changes to the initially-published mechanism. Only one team discovered this discrepancy, notably *after* the election. At implementation time, the second device implementers were not fully aware of the protocol behavior, which could have resulted in implementation mistakes.
- **Unclear Receipts Handling.** To prevent a potentially dishonest Polyas server from dropping ballots, the second device implementations allowed voters to download a receipt (which they could then send to the election organizers, see Sect. 2.1). However, the exact process for collecting receipts is not described in the specification. The implementations then handled this functionality differently regarding how prominently the download was recommended and how well the whole process was explained (i. e., that the voter needs to send the receipt to the election organizers). Further, one of the implementations even deviated from the intended user flow, as it additionally offered voters to deliver the receipt to the election organizers within the implementation itself (thereby also unknowingly circumventing the clash attack mitigation, see Sect. 2.3).

Protocol

Complete. Specify the full protocol (in terms of involved parties, algorithms and message exchanges).

Security Claims. State the protocol's precise security claims (e. g., using a formal definition), possibly together with appropriate proofs.

Security Model. Define the adversary model and trust assumptions under which the security claims are achieved.

Intuition. Provide an intuition of the purpose of the executed algorithms and exchanged messages to prevent accidental mistakes.

User Interface. As individual verifiability directly involves the voter, the second device implementation needs to also provide a user interface. This interface needs to guide the voter to execute their part of the protocol efficiently and effectively (i. e., for detecting when their vote was not cast as intended). For the voter, the individual verifiability tool is part of their overall voting process. Correspondingly, the tool needs to be aligned with the main system, notably concerning terminology and general usage patterns.

Within the second device specification, Polyas included an example of how a ballot is displayed. However, the specification does not recommend specific wording, layout or interaction patterns, which, e. g., led to the following issues:

- **Inconsistent Wording.** When Polyas tested the second device implementations, they noticed that the wording differs from their main system. This included basic terminology such as "ballot" or "receipt", but also the ballot layout. While most deviations were unintentional, notably for the ballot layout, some implementations initially chose to use a different layout than in the Polyas system on purpose, e. g., in an attempt for ballots which are easier to understand for the voter. For consistency, all second device implementations aligned wording and layout to the Polyas system.
- **Unclear Voter Instructions.** Second device implementations were often unclear in how a voter is expected to behave, both in normal circumstances (e. g., checking both the ballot and the voter ID), and in exceptional cases (e. g., error messages which do not properly advise the voter how to proceed).

User Interface

Consistency. Define wording, layout, and interaction to be used consistently throughout all implementations for reducing friction for the users.

User Interaction. Specify the high-level interaction of the user with the system (e. g., executed checks) for achieving the security guarantees.

Error States. Describe error states that the user might experience and how the second device implementation needs to advise the user to proceed.

3.2 Testing

During and after the implementation, the tools need to be tested. In general, tests cannot use the exact election setting (as testing in the actual election is likely impossible), and monitoring needs to be kept to a minimum to safeguard the users' privacy. At the same time, mistakes are costly; they happen directly at the voter's side and need an individual investigation. Moreover, any issue or corresponding investigation may prevent successful verification or break vote secrecy. Further, while false positives may undermine trust in the system, false negatives may even lead to missing potential attacks. Therefore, proper testing is vital.

Samples. For checking the implementation's correctness, it is very useful to have execution samples of a presumed correct implementation. When given the same input as in the samples, an implementation needs to return the corresponding output. Samples may be given at different abstraction layers; e. g., both for some particular part of an algorithm and even for a full protocol execution.

Within the second device specification, Polyas uses an election as a running example, and for this election, samples for the algorithms are provided. Further, the main specification provides additional in- and output samples for some of the lower-level algorithms. The given samples focus on input and expected output, but do not provide intermediate values. We give examples of where we encountered issues with the given samples:

- **Limited Sample Set.** Samples for only a single election were given. Hence, the samples did not cover the full range of possible scenarios. Notably, this election only used a single ballot, while two ballots were used during the real election. Further, some behavior may only be observable when given many samples (e. g., the particular behavior of Java's `BigInteger` serialization), or some particularly-crafted samples (e. g., craft an empty ballot to discover crashes due to off-by-one errors).
- **Missing Intermediate Values.** The protocol also involves computing a ballot's hash value over its serialization. While the final hash value is given, the intermediate serialization value is not. During development, when the hash turns out to be wrong, the developer only knows that the input of the hash function was incorrect, but misses any efficient way to learn *which part* of the implementation was wrong.

Samples

Diversity. Ensure samples are numerous, structurally diverse, cover both realistic scenarios and edge-cases, and both successful and failing cases.

Abstraction. Provide samples for all levels of abstraction (serialization, algorithms and protocol) to enable targeted testing and diagnostic.

Intermediate Values. Include intermediate values for complex algorithms to ensure that the debugging of (partially) wrong results remains efficient.

Self-service. Support implementers in crafting their own samples to check for surprising behaviors under particular edge cases.

System Testing. Besides testing (parts of) the implementation from samples, the second device implementations also require tests that involve the user interface in order to include the user's point of view. Moreover, such tests must take the larger user interaction with the main system into account, e. g., by also considering the ballot casting before verification.

The main system developed and operated by Polyas is proprietary, and the implementers hence neither had access to the code nor could they operate or configure the system themselves. Thus, the system tests build on test elections that Polyas had set up for the implementers. We give some examples of where this setup led to issues:

- **Communication Overhead.** When implementers wanted to perform system tests, they needed to ask Polyas to set up a test election and deliver the corresponding election configuration (i. e., the voter credentials and second device configuration). The test elections then ran on Polyas infrastructure, hence in case they had to investigate any issues (e. g., error messages from the server), implementers needed to consult Polyas (e. g., for accessing their server logs in order to find the exact error). Polyas was generous with their time, but the communication overhead had a chilling effect on running system tests; e. g., many edge cases such as empty or very big ballots were not tested.
- **Missing Independence to Mitigate Clash Attack.** For preventing clash attacks, the implementations needed to integrate a fix on short notice (see Sect. 2.3). Once the fix was applied, a careful re-test that involved system testing was necessary. As no test election was available at this time, Polyas had to be asked to set up a new test election. This delayed rolling out the fix but also shifted the system's theoretical trust assumptions: If Polyas were a dishonest system provider and had been actively performing clash attacks, the request for setting up a new test election would have made Polyas aware that "something was up" and could have changed their tactic.
- **Complications in Demos and User Studies.** For gathering feedback on the second device implementations, performing user studies and demonstrations before (expert) audiences is useful. Ensuring that a test election was available when needed did require keeping Polyas in the loop. Besides the accompanying planning and communication overhead, misunderstandings were inevitable, and, e. g., resulted in a forced pause of a user study until its test election was again made available. Some tools instead opted to simulate the main system, and thereby gained flexibility, but accepted additional effort and complexity of developing such a simulation.

System Testing

Availability. Provide system tests continuously before, during, and after the elections to enable maintenance and reproducibility of tool behaviors.

Independence. Ensure that performing system tests is independent of the system provider for an efficient process and to prevent interference.

Self-service. Permit implementers to freely configure the system and observe its behavior (e. g., by monitoring the logs or the database).

3.3 Quality

Besides implementation and testing, many other factors contribute to the project's success. As the individual verifiability tools are integrated into the Internet voting system, they thus need to be a good fit to avoid becoming the system's weak link. This holds particular importance for security-critical aspects, as the tools might otherwise compromise the voter's privacy or verifiability.

Quality of Instructions. For a successful implementation of the tool, implementation instructions need to be precise without being overwhelming. This may include written specification, source code and specific answers to individual questions. High-quality instructions optimize work efficiency by freeing up resources that are better spent on other aspects of the project. For this matter, providing the source code of the underlying system may give a substantial orientation, as it serves as a definitive interpretation of all other instructions. If implementers can navigate the code freely, run tests against it, or inspect it with a debugger, they can resolve ambiguities and misunderstandings independently. Further, they may be inspired to improve their own implementation or identify bugs in the provided code to report to the provider.

Polyas provided the main system specification focusing on algorithms and the separate second device specification focusing on the second device protocol. Further, Polyas responded to clarification requests per email and updated the specifications to resolve identified gaps and ambiguities throughout the project. Since resolving issues on the cryptographic layer was time-consuming without access to the source code, Polyas later-on additionally distributed the source code and tests of their own second device implementation. Below are some examples of how instructions could have been improved:

- **Informal API Definitions.** Around half of the pages in the second device specification describe its API. However, describing and implementing API endpoints is rather mechanic and can be largely automated from formal API specification (e. g., using OpenAPI), which also avoids ambiguities and gaps. In one of the tools, insufficient API implementation led to a broken display of the election description (fixed shortly after the election started).
- **Clarifications via Email.** Describing and answering highly technical issues in an email is difficult and time-intensive. Further, the results of such discussions are impractical to share with other parties.

Quality of Instructions

Dogfooding. Use as a system provider the published specification as the basis of their own implementations to reduce gaps and ambiguities.

Formalizations. Define formally where reasonable (e. g., OpenAPI) to enable automation (e. g., code generation) and reduce gaps and ambiguities.

Efficient Q&A. Provide an efficient process to communicate issues and solutions, and, if necessary, update the specification to clarify.

Source Code. Provide source code and tests of the specification to efficiently resolve even highly technical issues.

Quality of Implementation, Maintenance, and Operation. The correctness of implementations for functional requirements can be mostly covered by tests. However, tools also need to fulfill non-functional requirements, which include, e. g., understandability of source code, compliance with security best practices, and usability and accessibility of the tool. Well-prioritized non-functional requirements lead to a system that performs well even in exceptional circumstances. After implementation, the tool needs to be provided to the end user. If it is executed in the browser, this requires to securely operate a server. If it is a smartphone app, this needs publication in the app stores. Both options come with individual challenges, e. g., proper configuration of the web server, or the publication process of the app stores.

In the GI project, implementers were mostly students with little industry experience in software development. The project's focus was primarily on the cryptographic part, where non-functional requirements such as usability were of lower priority. Further, some implementers were initially unaware that operating their own server would be part of the project (no implementer chose to publish in the app stores). We encountered the following challenges:

- **Low UI/UX Polish.** Not all second device implementations invested much effort in polishing their user interface (e. g., choosing appropriate colors, animations, or margins) or the user experience (e. g., giving clear instructions to the voters). Hence, voters might be unable to use the tools appropriately and, therefore, not achieve the intended security guarantees.
- **Dependency with a Vulnerability in the Active Election.** During the election, the widely-used *axios* JavaScript dependency disclosed a vulnerability. The vulnerability did not impact the second device implementations, as it was only relevant when using a *XSRF-TOKEN* cookie.[13] Still, as it is good practice to not use dependencies with known vulnerabilities, all second device implementations were patched nonetheless.

13 For details, see `https://nvd.nist.gov/vuln/detail/CVE-2023-45857`.

- **Improper Header Configuration.** One of the implementations was operated using weak `http` header configurations, e. g., a header which helps to enforce `https` named `Strict-Transport-Security` remained unset. While assessing the impact of omitting this header is hard, such best-practice oversights should not happen.

Quality of Implementation, Maintenance and Operation

Non-functional Requirements. Appropriately identify and prioritize non-functional requirements (e. g., maintainability and usability).

Continuous Maintenance. Continuously observe for open flaws, and deliver the resolution to the voters in a timely manner.

Secure Operation. Provide the second device implementation in such a way that the voters can access it securely (e. g., no code manipulation).

3.4 Responsibility

Together, the main system provider, the election organizers, and the individual verifiability tool implementers hold the election. They rely on each other for a successful and trustworthy election, while their interests and capabilities differ. In consequence, they need to clarify, which party takes responsibility for which aspect early during the planning phase.

In the GI project, there was no contract or other explicit agreement between the involved parties. Responsibilities were not clearly defined and assigned. Instead, parties trusted the other parties to proactively contribute towards a successful election.

Responsibilities of the Main System Provider. The system provider can be expected to have deep domain knowledge in Internet voting and, in particular, know their own system best. As such, the provider needs to take the main responsibility to hold a secure and successful election using their system. In the context of the implementation of independent individual verifiability tools, this responsibility includes enabling the implementers to succeed in correctly and securely interfacing with their main system. It is in the main system provider's best interest that the implementers deliver their best possible work: A failure in the individual verifiability tools reflects poorly on the system as a whole, even if it is a false-positive. Further, well-constructed individual verifiability tools may uncover issues or potential for improvement in the main system.

Polyas provided sufficient specification, testing opportunities, and technical support to successfully implement several second device implementations. Polyas further tested the tools before the election and monitored them for availability during the election. Still, we identified the following issues:

- **Missing Quality Control.** To our knowledge, while Polyas tested the tools from the point of view of a voter, they did not perform other forms of quality control (e. g., code review, penetration testing). This would have likely reduced the observed quality issues (see Sect. 3.3).
- **Unclear Conditions of Endorsement.** In the Polyas main system, for each second device application, a corresponding QR code was shown after casting a ballot to start the verification. Polyas configured for which second device implementations such a QR code would be shown, while the conditions for which tools they selected to be included or not remained unclear. In the end, all tools were included, but the implicit possibility of exclusion gave Polyas negotiation power when requesting changes to the tools (e. g., concerning the wording used), jeopardizing the independence of the implementations.

Responsibilities of the Main System Provider

Technical Support. Provide appropriate resources to the implementers to enable them to deliver high-quality implementations in time.

Quality Assurance. Set clear quality thresholds over the implementations, e. g., based on code reviews, code scans, user testing, or penetration tests.

Responsibilities of the Election Organizers. While the main system provider has the general technical knowledge of the Internet election process, the election organizers know the specific context of the election, e. g., voter capabilities or the legal framework. Based on the context, the organizers decide on the process of the election, which includes both the choice of the Internet voting system (if any), and the particular system configuration. Both the system provider and the independent experts may contribute with their technical knowledge so that these decisions are taken in an informed manner in the given context.

In this project, the elections were organized by the GI. They were challenged in particular by the implementers' discovery of the possibility of a clash attack shortly before the election, and had to decide on a mitigation (see Sect. 2.3). We identified the following issues:

- **Unclear Project Target.** The purpose of the independent second device implementations was not clearly communicated.[14] Hence, the second device implementers could not consciously contribute towards fulfilling that purpose.
- **Ballot Validation.** The Polyas system does not prevent submitting invalid ballots (e. g., with too many candidates), but shows a corresponding warning to the voters when casting. However, the second device implementation was instead expected to

14 The GI's election FAQ states that the verification's target was to "check the source code of Polyas" (see `https://gi.de/wahlen/faq`, visited on: 04/24/2024.).

display the ballot without a warning, even if it was invalid. The decision whether to show this warning or not should have been an active decision by the GI.

- **Receipts Not Checked Diligently.** In the tally phase, many receipts could not be read out automatically (see Sect. 3.1), and consequently were not validated. Further, around 30 receipts were directly stored by one of the second device implementations and also not validated. Although overall, much effort was spent on the receipts process, the validation of the receipts was not fully seen to the end.

Responsibilities of the Election Organizers

Communication. Ensure that all parties have timely access to all necessary information, especially in case of known flaws.

Purpose. Clearly communicate what is to be achieved by the independent implementations (e. g., more independence from the system provider).

Configuration. Take informed decisions about the employment of the security mechanisms, adjusted to the election context.

Supervision. Observe and enforce that decisions on the security mechanisms are respected and implemented by all parties.

Responsibilities of the Independent IV-Tool Implementers. The individual verifiability tool implementers' responsibility is that their tool works as intended. Depending on the project's purpose chosen by the election organizer, additional responsibilities need to be assumed, e. g., to vouch for the implemented protocol, or that the respective main system (respectively the observed interaction) is of high quality. With growing responsibilities, the remuneration of the implementers becomes necessary, which then needs to be carefully arranged so as not to jeopardize the intended independence.

In the GI project, the second device implementers were volunteers not paid by and without any contractual binding to the GI or to Polyas. Naturally, assigning responsibilities to or even holding implementers accountable for unmet expectations are both very difficult in this scenario, both ethically and practically. Nevertheless, the implementers were implicitly responsible that their part of the election could proceed as planned. Besides respectful and transparent communication, the implementers reliably provided (and improved on short notice) their independent tools while respecting the surrounding conditions by Polyas and the GI. In the following, we list some of the occurred issues nonetheless:

- **Aborted Implementations.** Out of seven initial parties, three provided a working implementation in time. In this project, the four aborted implementations did not lead to any issues, as all parties communicated openly about the state of their project ahead of time.

- **Limited Availability of Implementers.** In the first election week, the implementer of one of the second device implementations was unreachable. This absence was planned and communicated beforehand, but still impacted the election: This tool received the clash attack mitigation patch only after the other tools were already patched, notably during the active election.
- **Limited Tool Availability.** During the election, Polyas kept monitoring the running instances of the second device implementations to ensure availability. At least one instance was detected to be unavailable at some point, and the implementers needed to react to restore the availability of their tool.

Responsibilities of the Independent IV-Tool Implementers

Transparency. Communicate transparently about the tools (e. g., whether development is on track with respect to the election's timeline).

Availability. Implement necessary changes, and guarantee the availability of the tools, especially while the election is active.

Respect Surrounding Conditions. Consent to the surrounding conditions, e. g., concerning availability, functionality, or responsible disclosure.

4 Conclusion

To make independent individual verifiability tools a reality, detailed and exact documentation is necessary. We reported on our experiences of developing and deploying three independent individual verifiability tools for the 2023 GI elections with recommendations regarding implementation, testing, quality, and responsibilities.

Besides a precise description of the protocol and its algorithms, a precise treatment of how to interface with the main system (e. g., the serialization standards) and with the voter (e. g., consistent wording between both systems) is vital. Moreover, in order to reduce the effort of both the system provider and the independent implementers and prevent (too) late surprises, a diverse set of test data and self-service access to the main system are needed. For the system provider, the independent implementations not only benchmark whether the documentation is complete and accurate and whether their system behaves as expected, but also result in a very detailed review of the part of their system that interfaces with the individual verifiability tool. This may result in helpful feedback to improve their system; as demonstrated in the GI project by the discovery and mitigation of a potential clash attack.

All in all, successful employment of independent tools is more than just the provisioning of a functional implementation. The tools become part of the voters' voting experience and must not become the weak link, both from a security perspective and a user experience point of view. Further, the tools also need to fulfill the targets of the election organizers,

e. g., the distribution of trust away from the system provider. This needs to clearly set quality targets and assign responsibilities without jeopardizing the independence of the independent implementations.

Many of the recommendations appear common sense. Still, some aspects are easily forgotten, apparent in the GI project primarily in the quality and responsibility topics. While we reported on the experiences of the first project of this kind, we hope other projects will emerge, find the recommendations helpful, and report on the results.

References

[Be19] Beckert, B.; Brelle, A.; Grimm, R.; Huber, N.; Kirsten, M.; Küsters, R.; Müller-Quade, J.; Noppel, M.; Reinhard, K.; Schwab, J.; Schwerdt, R.; Truderung, T.; Volkamer, M.; Winter, C.: GI Elections with POLYAS: a Road to End-to-End Verifiable Elections. In: Fourth International Joint Conference on Electronic Voting (E-Vote-ID 2019), Lochau / Bregenz, Austria Oct. 1–4, 2019. TalTech Press, pp. 293–294, 2019, URL: https://digi.lib.ttu.ee/i/?13563.

[Be21] Beckert, B.; Budurushi, J.; Grunwald, A.; Krimmer, R.; Kulyk, O.; Küsters, R.; Mayer, A.; Müller-Quade, J.; Neumann, S.; Volkamer, M.: Aktuelle Entwicklungen im Kontext von Online-Wahlen und digitalen Abstimmungen, tech. rep., Karlsruhe Institute of Technology (KIT), 2021, DOI: 10.5445/IR/1000137300.

[Be24] Benaloh, J.; Naehrig, M.; Pereira, O.; Wallach, D.: ElectionGuard: a Cryptographic Toolkit to Enable Verifiable Elections. In: 33rd USENIX Security Symposium (USENIX Security 2024), Philadelphia, PA, USA Aug. 14–16, 2024. https://eprint.iacr.org/2024/955, USENIX Association, 2024, IACR: 2024/955, URL: https://www.usenix.org/conference/usenixsecurity24/presentation/benaloh.

[Be87] Benaloh, J.: Verifiable Secret-Ballot Elections, PhD thesis, Yale University, 1987, URL: https://www.microsoft.com/en-us/research/publication/verifiable-secret-ballot-elections.

[BNV14] Buchmann, J.; Neumann, S.; Volkamer, M.: Tauglichkeit von Common Criteria-Schutzprofilen für Internetwahlen in Deutschland. Datenschutz und Datensicherheit-DuD 38 (2), pp. 98–102, 2014, DOI: 10.1007/s11623-014-0040-x.

[CGG19] Cortier, V.; Gaudry, P.; Glondu, S.: Belenios: A Simple Private and Verifiable Electronic Voting System. In: Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows. Vol. 11565. Lecture Notes in Computer Science, Springer, pp. 214–238, 2019, DOI: 10.1007/978-3-030-19052-1_14.

[Co23] Cortier, V.; Gaudry, P.; Glondu, S.; Ruhault, S.: French 2022 legislatives elections: a verifiability experiment. In: 8th International Joint Conference on Electronic Voting (E-Vote-ID 2023), Luxembourg City, Luxembourg Oct. 3–6, 2023. Lecture Notes in Informatics, Gesellschaft für Informatik (GI) e.V., 2023, URL: https://inria.hal.science/hal-04205615.

[Ha20] Haenni, R.; Dubuis, E.; Koenig, R.; Locher, P.: CHVote: Sixteen Best Practices and Lessons Learned. In: 5th International Joint Conference on Electronic Voting (E-Vote-ID 2020), Bregenz, Austria Oct. 6–9, 2020. Vol. 12455. Lecture Notes in Computer Science, Springer, pp. 95–111, 2020, DOI: 10.1007/978-3-030-60347-2_7.

[Ha23] Haenni, R.; Koenig, R.; Locher, P.; Dubuis, E.: CHVote Protocol Specification. IACR Cryptology ePrint Archive, 2023, IACR: 2017/325, URL: https://eprint.iacr.org/2017/325.

[HR21] Haines, T.; Rønne, P.: New Standards for E-Voting Systems: Reflections on Source Code Examinations. In: International Workshops on Financial Cryptography and Data Security (FC 2021), Revised Selected Papers, Virtual Event Mar. 5–5, 2021. Vol. 12676. Lecture Notes in Computer Science, Springer, pp. 279–289, 2021, DOI: 10.1007/978-3-662-63958-0_24.

[Ki22] Kirsten, M.: Formal Methods for Trustworthy Voting Systems: From Trusted Components to Reliable Software, PhD thesis, Karlsruhe Institute of Technology (KIT), 2022, DOI: 10.5445/IR/1000155115.

[Ko23] Koppenhöfer, O.: Individual verifiability in the Polyas e-voting system, Bachelor's Thesis, Institute of Information Security, University of Stuttgart, 2023, DOI: 10.18419/opus-13838.

[MT23] Müller, J.; Truderung, T.: CAISED: A Protocol for Cast-as-Intended Verifiability with a Second Device. In: 8th International Joint Conference on Electronic Voting (E-Vote-ID 2023), Luxembourg City, Luxembourg Oct. 3–6, 2023. Vol. 14230. Lecture Notes in Computer Science, Springer, pp. 123–139, 2023, DOI: 10.1007/978-3-031-43756-4_8.

[Ol12] Olembo, M.; Kahlert, A.; Neumann, S.; Volkamer, M.: Partial Verifiability in POLYAS for the GI Elections. In: 5th International Conference on Electronic Voting 2012 (EVOTE2012). Vol. P-205. Lecture Notes in Informatics, Gesellschaft für Informatik (GI) e.V., pp. 95–109, 2012, URL: https://dl.gi.de/handle/20.500.12116/18228.

[OSV11] Olembo, M.; Schmidt, P.; Volkamer, M.: Introducing Verifiability in the POLYAS Remote Electronic Voting System. In: Sixth International Conference on Availability, Reliability and Security (ARES 2011), Vienna, Austria Aug. 22–26, 2011. IEEE Computer Society, pp. 127–134, 2011, DOI: 10.1109/ares.2011.26.

[Sw24] Swiss Post Ltd.: Swiss Post Voting System – System Specification, tech. rep., version 1.4.1, Swiss Post Ltd., 2024, URL: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/5a0987fb206510c5494312f505b7d56a5e2c1624/System/System_Specification.pdf, visited on: 06/24/2024.

[Tr23] Truderung, T.: POLYAS 3.0 Verifiable E-Voting System, tech. rep., version 1.3.2, Berlin, Germany: POLYAS GmbH, 2023, URL: https://github.com/kastel-security/polyas-core3-second-device-verification/blob/82a74d576469deadbcda600ded81f13f0de98f1c/doc/polyas3.0-verifiable-1.3.2.pdf, visited on: 07/31/2023.

[Tr24] Truderung, T.: Polyas-Core3 Second Device Protocol, tech. rep., version 1.1, Berlin, Germany: POLYAS GmbH, 2024, URL: https://github.com/famoser/polyas-verification/blob/7a0f708f41e471a0208c02c989ec727f236e349e/spec/second-device-spec-1.1.pdf, visited on: 07/24/2024.

Acknowledgement

This work was supported by funding from the topic Engineering Secure Systems, subtopic 46.23.01 Methods for Engineering Secure Systems, of the Helmholtz Association (HGF) and by KASTEL Security Research Labs.