On the Auditability of the Estonian IVXV System and an Attack on Individual Verifiability

Anggrio Sutopo, Thomas Haines, Peter Rønne

On the Auditability of the Estonian IVXV System

and an Attack on Individual Verifiability

Anggrio Sutopo[1], Thomas Haines[1*], and Peter Roenne[2**]

[1] Australian National University, Canberra, Australia thomas.haines@anu.edu.au
[2] CNRS, LORIA & Univ Lorraine, Nancy, France

Abstract. The development and auditing processes around electronic voting implementations are much too often deficient; this is particularly true for the measures taken to prevent cryptographic errors – potentially with grave consequences for security. To mitigate this, it is common to make the code public in order to allow independent experts to help uncover such flaws.

In this paper we present our experiences looking at the IVXV system used for municipal and national elections in Estonia as well as European Parliament elections. It appears that, despite the code being public for over five years, the cryptographic protocol has *not* seen much scrutiny at the code level. We describe in detail the (lack of) auditability and incentives which have contributed to this situation. We also present a previously unknown vulnerability which contradicts the claimed individual verifiability of the system; this vulnerability should be patched in the next version of IVXV system.

1 Introduction

Two fundamental requirements of any democratic election are the privacy of the voter and the integrity of the ballot. As many jurisdictions around the world move to electronic voting (e-voting), these two properties have to be guaranteed and here cryptographic techniques play a prominent role. But even the most sophisticated cryptographic techniques are useless if their software implementation contains bugs. The desire to ensure the integrity of elections, even in the presence of such bugs, no matter if accidental or malicious, has led to the notion of "software independence":

> "A voting system is software-independent if an (undetected) change or error in its software cannot cause an undetectable change or error in an election outcome." [Riv08]

[*] Thomas Haines is the recipient of an Australian Research Council Australian Discovery Early Career Award (project number DE220100595).

[**] This work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

One important way to produce publicly verifiable evidence while preserving privacy is via zero-knowledge proofs (ZKP) [GMR85], which allow the demonstration of the truth of a statement without leaking additional information. Despite the name “software-independent” these systems are still dependent on the correctness of the software verifying the evidence, as we will see.

At least in part due to the complexity of the requirements and cryptography involved, electronic voting systems have a long history of insecure implementations largely as a result of insecure implementation of cryptography. For example, the Scytl-Swiss Post system contained many components which were broken despite extensive review [HLPT20]. This has been true in many systems: the iVote system [HT15], the e-voting system previously used in national elections in Estonia [$\text{SFD}^{+}14$], the Moscow voting system [GG20], and the issues with Voatz [SKW20] and Democracy Live [SH20].

There is a widespread presumption that these errors are indicative of a lack of expertise within the vendors, and the companies which are paid to audit them, to develop and audit systems which securely implement cryptographic protocols. In an attempt to remedy this, it has become common to open the source code to public scrutiny to help find these errors. The IVXV system [oE] is a good example of this, having been developed and largely made public in the wake of the issues found in the previous system [$\text{SFD}^{+}14$].

Haines and Roenne [HR21] argue that making the system public appears to be a necessary condition at present for developing secure systems but not a sufficient one. They make nine recommendations which are aimed at ensuring that the public code is comprehensible, and capable of being checked for the most common errors in a reasonable time frame. They further argue that unless the code meets their requirements, little progress in security can be expected.

In this work, we analyse the available information on the IVXV system with respect to these standards; despite being available for five years the system has a paltry degree of auditability. Based on this, it is to be expected that the system has errors which would have been detected already in a system with a better auditability; to this point we found a vulnerability which breaks individual verifiability which should have been caught with a quick review of the system specification (without any need to look at the code). Since the system lacks an adequate specification, this error has gone undetected until now; we conjecture the system contains more similar vulnerabilities. In our conclusion (Sec. 5) we comment on the changes that Estonia could make to the process in order to improve this situation.

1.1 E-voting in Estonia

Estonia has been using internet voting since the early 2000s for elections. The system has had multiple changes but has maintained the same key designs, using the national ID card to verify the user’s identity in the voting application.

Estonia became the first country to offer the option of voting online nationwide after implementing the first version of their online voting system in 2005. This online voting system, called Internet voting or I-voting, was first used in

2005 to vote for the municipal elections [MM06]. The system has been overhauled multiple times due to security concerns, in particular concerning the possibility that a vote might be manipulated after it has been submitted without the organiser's or voter's knowledge. A mechanism to allow a voter to verify their votes was introduced in 2013 [HW14], but within a year it was shown to have some flaws [SFD$^+$14]. In response, a new system was implemented in 2017 in collaboration with the vendor Smartmatic - Cybernetica C.O.E for Internet Voting, called the IVXV system [HMVW16]. In the recent 2023 Estonian parliamentary election 312,181 votes were cast electronically, this means it was the first election with more than half of votes cast online [Wik]. Even further, to the best of our knowledge, this also makes it the largest online voting by participation.

1.2 The IVXV system

The IVXV system follows the basic concept of an envelope scheme [HMVW16]. In a physical election, the vote ballots are put into a sealed envelope that is only unsealed during the tally period. The IVXV system works in a similar way, where each voter sends their encrypted vote alongside their digital signature to the collector service [Sta20]. This would in theory allow the votes to remain secret while allowing the collector service to verify that the vote belongs to an eligible voter. In addition, the system also allows each voter to verify that the vote accepted by the collector service is as intended using the verification application [Sta20]. Finally, the system in theory now allows third party auditors to audit the tabulation process [HMVW16], but to our knowledge this has not occurred in the past elections using the system.

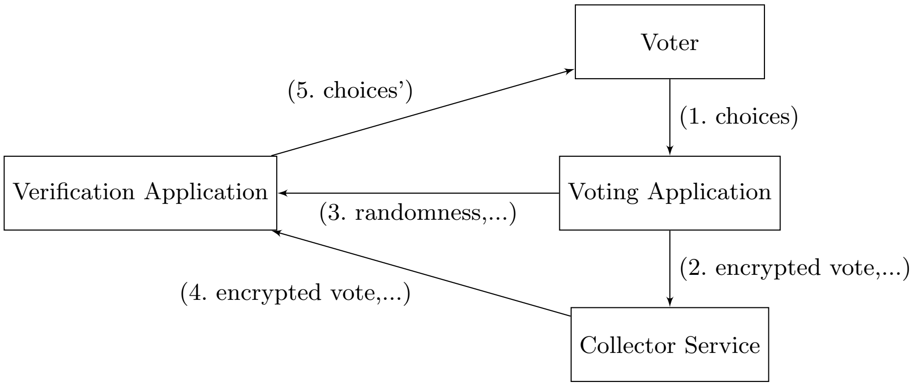

Fig. 1. Overview of vote casting and verification in the IVXV system

The IVXV system consists of a mix of services and applications that each handle a specific task, while also being connected to supporting external components [Sta19]; we highlight the main flow in Figure 1. The internal components include the collector service, the processing application, the key application, and the audit application. The external components include the identification service, the signing service, and the registration service. There are also independent but closely connected components such as the voting application, the verification application, and the mixing application. The source code of most parts of the system can be viewed publicly in the IVXV GitHub repository, but the exact code used in active development is kept private; nevertheless, the version of the code used in any given election should have a corresponding commit in the GitHub repository. In the remainder of the section, we will summarise the explanation found in [Sta19] regarding the architecture of the IVXV system which can be separated into the internal and external components.

Internal Components

Collector Service: The collector service is in charge of collecting the votes from the voting application and storing them before the tallying process. It is connected to external components that are focused on supporting identification, verification, and qualification. The service consists of several micro-services that are all programmed in Go and are closely connected with the external components. For brevity we omit a description of these micro-services.

Key Application: The key application is in charge of generating the vote encryption and decryption keys for each election as well as decrypting and counting the votes. The application is programmed in Java.

Processing Application: The processing application is in charge of verifying, cancelling, and anonymising the votes collected over the voting period. The application can generate a list of voters as well as the anonymised votes after receiving the information stored in the collector service and the registration service. The application is programmed in Java.

Audit Application: The audit application is in charge of mathematically verifying that the vote count and mixing is correct. The application generates a detailed log containing the assessment of the audit after receiving the anonymised votes, mixed votes, shuffle proof, and voting result. The application is programmed in Java.

External Components

Voting Application The voting application is the program used by voters to submit their votes to the system [HMVW16]. It is available in desktop systems such as Windows, macOS, and Linux and can be downloaded from the election authority's website [HW14]. *The source code of the voting application was/is not available for scrutiny and the description below has not been checked to match the implementation.* The system expects the process of voting to work as follows [Sta20]:

1. The voter uses the voter application to submit their vote.
2. The collector service stores the vote.

3. The voter can use the verification application to check if their vote has been stored properly.
4. At the end of the voting period, the collector service issues the ballot box to the election organiser while the registration service issues the list of registered votes collected in the collector service.
5. The election organiser calculates the voting result.

In this section, we adopt the convention of [HMVW16] in defining the encryption used by the IVXV system. The encryption used to encrypt the votes is a homomorphic public key cryptosystem.. The algorithm should follow a scheme of $\epsilon = (Gen_{enc}, Enc, Dec)$ with its key generation, encryption, and decryption functions as well as the cryptographic hash function $Hash$. The algorithm that is implemented in the current version of the system is the El-Gamal cryptosystem. The election organiser generates an election key pair that is used for encrypting and decrypting the votes

$$(ek^{elec}_{pub}, ek^{elec}_{priv}) \leftarrow Gen_{enc}$$

The public key ek^{elec}_{pub} is made available to everybody and is used by both the voting and verification application to encrypt and verify the votes, respectively, while the private key ek^{elec}_{priv} is stored securely by the organiser and is used for tabulating the voting results.

The certification authority is in charge of storing the keypair $(sk^{CA}_{pub}, sk^{CA}_{priv})$ and the corresponding certificate $Cert^{CA}_{CA}$, and each eligible voter posses a unique identifier $i \in I$ as well as a certified signature keypair:

$$\forall_i \in I, (sk^i_{pub}, sk^i_{priv}) \leftarrow Gen_{sig}, Cert^i_{CA} = Sign(sk^{CA}_{priv}, (i, sk^i_{pub}))$$

The application implements the double envelope format by first encrypting the candidate choice c_v as $ballot_{c,r} = Enc(c_v, r_v, ek^{elec}_{pub})$ where $r_v \leftarrow R$ is a random number, followed by signing the encrypted vote with the voter's private key such that $vote_v = Sign(sk^v_{priv}, ballot_{c,r})$. The application then sends the voter identifier v, certificate $Cert^v_{CA}$, and signed encrypted vote $vote_v$ to the collector service, and receives a unique identifier vid and the registration service confirmation reg_{vid} which is verified using the $Hash$ of the vote. In order to verify, the identifier vid and encryption randomness r_v are presented by the voting application as a QR code that can be scanned by the verification application.

Verification Application The verification application is used by voters to verify that their vote has been stored properly by the collector service [HMVW16]. It is available on mobile devices through either the Google App Store for Android or the Apple App Store for iOS [HW14]. The Android version is programmed in Java while the iOS version is programmed in Objective-C. The source code for the Android version is available publicly[3], as is the iOS version[4]. Even though the applications are developed on

[3] https://github.com/vvk-ehk/ivotingverification

[4] https://github.com/vvk-ehk/ios-ivotingverification

different platforms, they follow the same design for the cryptographic part, importantly they contain the same flaw, mentioned below. The description below of how the application functions is taken directly from [HMVW16].

1. The voter obtains from the voting application a QR code that encodes a unique vote identifier vid and randomness r_v after submitting their vote.
2. The voter scans the QR code using the verification application.
3. The verification application uses an authenticated TLS channel to connect with the collector service and sends the vid.
4. The collector service sends back the signed encrypted vote $vote_v$ as well as the registration service confirmation reg_{vid}. An error is returned if an unknown vid is used or if it exceeds the verification time limit.
5. The verification application verifies both the vote and the registration service confirmation then displays the identity v to the voter.
6. The verification application uses the list of candidates C and randomness r_v to find a $c' \in C$ such that $Enc(c', r_v, ek^{elec}_{pub}) = ballot_{c,r}$. The result either shows the decrypted vote or an error message. It is up to the voter to determine if their vote is correct or if they need to submit a new vote using the voting application.

Security Model Within the repository, there are no clear claims regarding the security model desired of the system. The document available in the election information website, [Sta17], describes the system from a high level point of view, but also doesn't make any clear claims regarding which components are considered trusted. While there are several claims such as the cryptosystem used being secure and that each voter can verify their votes [Sta17], there are no information regarding which components are considered critical.

Reading between the lines, it appears that the system considers any attack that require the compromise of any part of the backend of the system to be outside the scope of the model. Whereas, an attack that can be launched by the voting device alone is in scope. Writing out the model in detail would be helpful in guiding examiners to the issues which the stakeholders care most about.

2 Scope, Methodology, and Contributions

Scope: Our work is based on version 1.7.7 of the IVXV system as it appeared at https://github.com/vvk-ehk/ivxv. In addition, we looked at selected parts of the Android verification application and the iOS verification application. Further, some information was retrieved from the election information website located on https://www.valimised.ee/en. For none of these sources was our review exhaustive; our review was limited to the documents which were available in English.

The key focus of our work is an examination of the auditability of the system with respect to the requirements listed in [HR21]. In addition, we focused on understanding the individual verifiability of the system at the code level. While we discovered an interesting vulnerability in the individual verifiability, our examination of this property was not adequate to establish the security of the system with respect to this property. In Sec. 5 we also comment on other areas of the code we are worried may negatively effect the verifiability of the system.

There is a wide range of security issues which are outside our scope. We believe that in many of these areas, the current process in Estonia does a better job of detecting vulnerabilities in the system than it does for the cryptographic core. Overall, the vulnerabilities in the cryptography are likely to require a more sophisticated attacker to understand and exploit than other kinds; however, it is hard to evaluate this assertion since we have little idea what kinds of vulnerabilities exist within the cryptographic implementation.

Methodology: Our review methodology focused on examination of the system documents and manual code review using an IDE. Though we were unable to build the system as a whole, we did write some unit tests to test particular functionalities of the system.

Contributions: Our work highlights deficiencies in the auditability of the IVXV system. We also reveal a significant vulnerability in the individual verifiability of the system which has gone undetected for many years. Overall, our work provides a helpful resource for understanding the security of the cryptographic implementation of the IVXV system and what can be done about it.

3 Flaw in Individual Verifiability

The system has two different parts that care about the decryption of the votes. First is the key application which needs to decrypt all votes in the system that are considered valid before sending them to the tallying service. Second is the verification application which needs to extract the plaintext vote that was previously submitted by the voter using the voting application. The verification application can be separated into the Android and iOS versions. The vulnerability in the system that we found is related to the three different implementations of vote decryption, which are incomplete and cause a vulnerability that can be exploited using at least three different possible attacks. The relevant files for the key application, which can be found in the IVXV GitHub repository [ivx], are

- RecoverDecryption.java
- ElGamalPrivateKey.java
- ModPGroup.java
- Plaintext.java

The relevant files for the Android version of the verification application which can be found in https://github.com/vvk-ehk/ivotingverification are:

- DecryptionActivity.java
- DecryptionTask.java
- ElGamalPub.java

The relevant files for the iOS version of the verification application which can be found in https://github.com/vvk-ehk/ios-ivotingverification are:

- Crypto.m
- Crypto.h
- ElgamalPub.m
- ElgamalPub.h

To understand the vulnerability, we must first understand the correct implementation of the ElGamal encryption mechanism. ElGamal encryption occurs over a cyclic group G of prime order q with a generator g. The private key x is an element of $\mathbb{Z}_q$, and the public key y is g^x. Encryption of a message $m \in G$ takes a random r from $\mathbb{Z}_q^*$ and computes the tuple $(g^r, y^r m)$. The ciphertext (c_1, c_2) under a given public key g^x can be decrypted by computing c_2/c_1^x. It also possible given knowledge of r such that a given ciphertext (c_1, c_2) is equal to $(g^r, y^r m)$ to compute the message as c_2/y^r.

In the IVXV system, a voter is able to verify that their vote has been correctly submitted by the voting application by using the verification application to compute c_2/y^r such that it will display the message m which is equivalent to their recorded vote v. The interactions between a voter and the system according to the code is shown in the following process:

1. The voter submits their vote v to the voting application.
2. The voting application independently samples the random number r.
3. The voting application computes the ciphertext $(c_1, c_2) = (g^r, y^r v)$ and sends it to the collector service.
4. The voting application generates a QR code for the vote for verification.
5. The voter tries to use the verification application to verify their vote by scanning the QR code generated by the voting application.
6. The verification application receives the random number r from the voting application via the QR code.
7. The verification application receives only part of the ciphertext, $c_2 = y^r v$ from the collector service.
8. The verification application recovers v by computing c_2/y^r which the voter can check against the intended choice.
9. At the end of the election, the key application recovers v by decrypting the ciphertext $(c_1, c_2) = (g^r, y^r v)$ using the private key x.
10. The recovered vote v is sent to the tallying service.

The current design doesn't verify that c_1 is equal to g^r, and the system seems to be running under the assumption that the verification of c_1 isn't required. However, this verification is crucial in ensuring that the cryptosystem is functioning properly, and the lack of it results in a vulnerability that could be exploited in three different possible attacks of various severity. To highlight the difference in the process for each attack, the diverging step is written in italic.

3.1 Attack 1: Discarding a Vote

In the first potential attack method, an attacker is able to discard the vote of an existing voter by sending a different random number r' in place of r. The attack is executed in the following process:

- *The voting application independently samples the random numbers r and r', which will be different with overwhelming probability.*
- *The voting application computes the ciphertext $(c_1, c_2) = (g^r, y^{r'}v)$ and sends it to the collector service, note the different random numbers used.*
- *The verification application receives r' from the voting application.*
- *The verification application receives only part of the ciphertext, $c_2 = y^{r'}v$ from the collector service. Note that the application doesn't receive $c_1 = g^r$.*
- *The verification application recovers v by computing $y^{r'}v/y^{r'}$. The voter feels assured that their vote has been properly stored in the system.*
- *The key application decrypts the ciphertext $(c_1, c_2) = (g^r, y^{r'}v)$. However, since this will evaluate to $y^{r'-r}v$, which is a random element of the group, the decrypted text will not be well formed with overwhelming probability.*
- *The vote is then discarded by the key application and isn't counted by the tallying service.*

The result of this attack is that the voter is disenfranchised as their vote is discarded without their knowledge by the key application. This potential attack violates the security model as it doesn't require the attacker to be able to compromise the backend of the system; the attack only requires the adversary to control the voting application. If this attack occurred, it would result in invalid votes appearing in the output. Based on the information provided by the maintainers, this attack hasn't occurred during any of the previous elections.

3.2 Attack 2: Changing a Vote with Knowledge of the Private Key

In the second attack method, an attacker is able to manipulate the vote of an existing voter into a different valid choice but requires knowledge of the private key x. The attack is executed in the following process:

- *The attacker obtains the private key x by some means. For example, accessing the part of the backend of the system where it is stored.*
- *The voting application computes the ciphertext $(c_1, c_2) = ((y^r v/v')^{1/x}, y^r v)$ and sends it to the collector service. Note that the plaintext vote in the ciphertext that is sent is now changed from v to v'.*
- The verification application recovers v by computing $y^r v/y^r$. The voter *feels* assured that their vote has been properly stored in the system.
- *At the end of the election, the key application in an offline environment recovers v' by decrypting the ciphertext $(c_1, c_2) = ((y^r v/v')^{1/x}, y^r v)$.*
- *The recovered vote v' is sent to the tallying service.*

The result of this attack is that the voter will think that their preferred choice has been stored in the system when in reality the choice that is counted is entirely different. This potential attack is considered to be outside of the security model as it requires the attacker to obtain the private key.

3.3 Attack 3: Changing a Vote without Knowledge of the Private Key

In the third attack method an attacker is able to manipulate the vote of an existing voter into a different valid choice but requires knowledge of the discrete log relationship between the possible candidate choices in the base of the public election key. The attack is executed in the following process:

- *The attacker, by some means, gains knowledge of the discrete log relationship between two possible voting choices v and v' in the base of the public election key y, i.e. the attacker knows s in $v/v' = y^s$. For example, if the attacker is able to choose the vote encoding configuration of the system, then the relationship can easily be known.*
- *The voting application computes the ciphertext $(c_1, c_2) = (g^r, y^r v')$ and sends it to the collector service. Note that this encrypts a different vote v'.*
- The voting application generates a QR code for verification *but the QR code contains $r' = r - s$ (modulo the order of the group) instead of r.*
- The voter tries to use the verification application to verify their vote by scanning the QR code generated by the voting application.
- *The verification application receives $r' = r - s$ from the voting application.*
- *The verification application receives only part of the ciphertext, $c_2 = y^r v'$ from the collector service. The verification application doesn't receive $c_1 = g^r$.*
- *The verification application recovers v by computing $y^r v'/y^{r'} = y^r v'/y^{r-s} = v' y^s = v$. The voter feels assured that their vote has been properly stored.*
- *At the end of the election, the collector service attempts to decrypt the ciphertext $(c_1, c_2) = (g^r, y^r v')$ and will instead recover v' instead of v.*
- *The recovered vote v' is sent to the tallying service.*

The result is similar to the previous attack method, but now the attacker can execute it without prior knowledge of the private key x. We wish to thank Vanessa Teague for pointing out this variant of the attack. We believe this attack is also outside the security model because of how the encoded voting options are configured; however, we have not done a full investigation of this since the underlying vulnerability should be patched before the next election.

This attack is more interesting and concerning than the others since it is not intuitive that choosing the encodings of voting options should allow one to break individual verifiability and hence this avenue might be easier to exploit than an attack which requires knowledge of the secret key. Further, the idea of encoding votes in terms of y^v was recently used for efficiency reasons in [DPP22].

3.4 Computational condition for precision attacks

We can give a precise computational necessary and sufficient condition for launching attacks changing an intended vote encoded by v into a vote encoded by v':

Theorem 1. *An adversarial algorithm can compute an ElGamal encryption of v' under the public key $y = g^x$ that will verify as a vote for v using only the second part of the ciphertext, as above, if and only if $(v'/v)^{1/x}$ can be computed.*

To see this, first assume we the adversary has created a ciphertext of v' as $(c_1, c_2) = (g^r, y^r v')$ and at the same time outputs r' such that $c_2 = y^{r'} v$. Then $v'/v = y^{r'-r} = g^{x(r'-r)}$ and hence $(v'/v)^{1/x} = g^{r'-r} = g^{r'}/c_1$ can be computed as well. On the other hand, given $(v'/v)^{1/x}$, we can choose any r' to compute $(g^{r'}(v/v')^{1/x}, y^{r'} v)$ which is a ciphertext decrypting to v' but verifying as v. If a plaintext-knowledge proof was required, such an attack would not be possible.

Given knowledge of the secret election key x, as in attack 2, or the discrete log in attack 3 immediately allows to compute $(v'/v)^{1/x}$ and launch the attack. If nothing is known about v/v' then this assumption is closely related to the 1-Diffie-Hellman Inversion Problem [PS00] of computing $g^{1/x}$ from g^x. If v'/v is directly a known power of the generator g then they are indeed equivalent. Since this is a known hard problem, also used in other voting schemes, e.g. [RRI16], we would not expect attacks from external attackers in this case.

Note that if the implementation also verified the first part of the ciphertext, it would be impossible to find an r defeating verification, and no computational assumption would be needed for the soundness of the verifiability check.

Finally, we might wonder if *auxiliary information*, such as access to decryption outcomes, would help an attacker. In general, this seems unlikely, however, there are corner cases: Consider an attacker as in Attack 3 who controls the vote encoding. If this attacker tries to launch an attack before y^x is known, it will not be possible. However, if a simple decryption of any arbitrary ciphertext (c_1, c_2) is known, then the attack can be launched since the decryption reveals c_1^x and the attacker could set $v'/v = c_1^x$ to later use it in the attack.

3.5 Solution

The solution to all three potential attacks is to redesign the verification application such that it receives c_1 which would enable it to verify that the random number r sent by the voting application satisfies the condition $c_1 = g^r$. We have informed the maintainers of the existence of the vulnerability as well as the various potential attack methods that exploit it, and they have assured us that they have started working on implementing the solution. However, at the time of writing, they haven't published their implementation of the solution, so we couldn't determine if the vulnerability is fixed or not.

The relevant stakeholders should also consider revising the security model of the system so that they and the system are prepared for more potential attacks and scenarios, especially those that involve attacks on the backend of the system.

3.6 Why wasn't this already noted?

The vulnerability is straightforward which makes it more concerning. The fact that is hasn't been noticed until now we put down to the fact examinations of the IVXV system, for example [Per21] focused largely on the specification level. In the next section we comment on some of the issues which make examining the code of the IVXV system so painful.

4 Analysis with Regards to Haines and Roenne 2021

This section will comment on the quality of the IXVX public information based on the standard in [HR21]. We summarise the results in Table 1 and give details in the following paragraphs.

Property	Result
Clear claims	✗
Thorough documentation	✗
Minimality	✗
Buildable	✗
Executable	✗
Exportable	✗
Consistent documentation and source	✗
Regularly updated	✗
Minimal restriction on disclosure	✓

Table 1. Summary of IVXV with respect to requirements listed in [HR21].

Clear Claims As we noted in Sec. 1.2, the system lacks a thorough security model and claims. As such, we can consider the system to not follow this standard. We would encourage adopting standards with a similar degree of granularity as those used in Switzerland.[5]

Thorough Documentation Within the repository, the documentation is very poor. There are no clear high level descriptions of the components contained in the repository. Inside each component's directory there is only a small description of what the directory is supposed to be inside some of the makefiles and README files. The auditor, key, processor, and voting directories seem to have a more detailed description compared to the other directories, but the additional information is about the usage of the directories and not much about the code themselves. The remaining documentation can be found in parts of the code that explain some of the functions and classes. To increase the available expertise for auditing the system it would be useful to increase the share of information that is available in English.

There are only three documents available in English: a high level overview of the system [Sta17], and the same two documents which can be generated using the source code. As such, we can consider the system to not follow this standard as the documentation is hard to find and doesn't properly describe the system as well as it could have.

Minimality The system contains significant amounts of unused and redundant code; this unnecessary clutter hinders auditing. We are cautious about giving

[5] https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting/versuchsbedingungen.html

a specific quantitative metric for this criterion because the hindrance to auditing of unused and redundant code is not independent of other issues; for example, if it was clear from the documentation what code was relevant, and what wasn't, the same level of redundant code would be less obstructive. Nevertheless, we can consider the system to not follow this standard.

Buildable Within the repository, there are limited instructions regarding building the project. There is a main README and makefile that describes some instructions to gather the system's external dependencies, build, and test the system, but we didn't manage to pass the installation of external dependencies phase. The external packages are divided into the Java, Go, and Python dependencies. The installation instructions for these dependencies were either not working, missing key files, or unclear.

At the time of writing this paper, we have not received a response after providing the maintainers with the error messages on August 2022, so we cannot be certain if the dependencies were causing the issue. As such, we can consider the system to not follow this standard as we couldn't build the system using the provided instructions.

Executable Since the system didn't follow the previous standard in that it couldn't be built, it is also considered to not follow this standard as the executable cannot be produced. The system does provide some documentation for running the executable files, but only for the auditor, key, and processing directories. As such, we can consider the system to not follow this standard.

Exportable Since the system didn't manage to follow the previous standard in that it couldn't be executed, it is also considered to not follow this standard as no auditable output could be produced. The system does have some test files but we didn't manage to read through or use them as we were focused on finding the correct method of building the program. As such, we can consider the system to not follow this standard.

Consistent Documentation and Source The system has several examples of having inconsistencies between the written documentation and the actual source files. For example, the dependencies installation instructions couldn't actually be followed because a file was missing from the repository. Another example would be the lack of clear instructions on how to run each component of the project. Most directories also lack a main overview file, and while some files that we read through had some explanation of its contents, some also lack any supporting comments on its functions. As such, we can consider the system to not follow this standard.

Regularly Updated The system's update history can be seen through the commits page in the GitHub repository. It is very rarely updated, we can consider the system to not follow this standard.

Minimal Restrictions on Disclosure We are not aware of any restrictions on disclosure that the system has regarding vulnerabilities. As such, we can consider the system to follow this standard.

Since the project only manages to fulfil one of the nine standards, minimal restrictions on disclosure, we foresee, inline with prior work, that the system likely has vulnerabilities. This conjecture is supported by the vulnerability discussed.

5 Conclusion

Our work highlights the significant deficiencies in the source code of the IVXV system which has been made available. These deficiencies increase the effort of examiners to audit the system and seem to have been fairly effective in preventing even simple vulnerabilities from being discovered. As an example of this, we point to the vulnerability in verifiability which should have been apparent even at the specification level. We make the following recommendations:

Revise system to allow better auditability The system and documentation need to be reworked according to the points raised above to allow better auditability. We highlight in particular the need for clear security claims and system description. We also encourage the removal of unnecessary duplication of code from the system, particularly the numerous encoders and decoders of ballots in the different parts of the system.

Incentives examination Further investigation is needed to determine what other unexposed issues exist in the system. We suggest encouraging such examination by introducing a bug bounty, or similar, with rewards based on the severity of vulnerabilities reported; this would be easy to define once the security requirements had been more clearly articulated. Alternatively, an approach similar to Switzerland's could be considered where auditors are asked to comment both on security of the code but also the specification.

References

[DPP22] Henri Devillez, Olivier Pereira, and Thomas Peters. How to verifiably encrypt many bits for an election? In *Computer Security–ESORICS 2022: 27th European Symposium on Research in Computer Security, Proceedings, Part II*, pages 653–671. Springer, 2022.

[GG20] Pierrick Gaudry and Alexander Golovnev. Breaking the encryption scheme of the Moscow internet voting system. In *Financial Cryptography*, volume 12059 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2020.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304. ACM, 1985.

[HLPT20] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *IEEE Symposium on Security and Privacy*, pages 644–660. IEEE, 2020.

[HMVW16] Sven Heiberg, Tarvi Martens, Priit Vinkel, and Jan Willemson. Improving the verifiability of the Estonian internet voting scheme. In *E-VOTE-ID*, volume 10141 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 2016.

[HR21] Thomas Haines and Peter B. Rønne. New standards for e-voting systems: Reflections on source code examinations. In *Financial Cryptography Workshops*, volume 12676 of *Lecture Notes in Computer Science*, pages 279–289. Springer, 2021.

[HT15] J Alex Halderman and Vanessa Teague. The New South Wales iVote system: Security failures and verification flaws in a live online election. In *International Conference on E-Voting and Identity*, pages 35–53. Springer, 2015.

[HW14] Sven Heiberg and Jan Willemson. Verifiable internet voting in Estonia. In *EVOTE*, pages 1–8. IEEE, 2014.

[ivx] IVXV GitHub repository. https://github.com/vvk-ehk/ivxv.

[MM06] Ülle Madise and Tarvi Martens. E-voting in Estonia 2005. the first practice of country-wide binding internet voting in the world. In *Electronic Voting*, volume P-86 of *LNI*, pages 15–26. GI, 2006.

[oE] State Electoral Office of Estonia. Ivxv online voting system. https://github.com/vvk-ehk/ivxv.

[Per21] Olivier Pereira. Individual verifiability and revoting in the Estonian internet voting system. *IACR Cryptol. ePrint Arch.*, page 1098, 2021.

[PS00] Birgit Pfitzmann and Ahmad-Reza Sadeghi. Anonymous fingerprinting with direct non-repudiation. In *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 401–414. Springer, 2000.

[Riv08] Ronald L Rivest. On the notion of 'software independence' in voting systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3759–3767, 2008.

[RRI16] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In *Financial Cryptography Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 176–192. Springer, 2016.

[SFD$^+$14] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J Alex Halderman. Security analysis of the Estonian internet voting system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 703–715. ACM, 2014.

[SH20] M Specter and J Halderman. Security analysis of the democracy live online voting system, 2020.

[SKW20] Michael A. Specter, James Koppel, and Daniel J. Weitzner. The ballot is busted before the blockchain: A security analysis of voatz, the first internet voting application used in U.S. federal elections. In *USENIX Security Symposium*, pages 1535–1553. USENIX Association, 2020.

[Sta17] State Electoral Office of Estonia. General framework of electronic voting and implementation thereof at national elections in estonia, 2017.

[Sta19] State Electoral Office of Estonia. *IVXV-architecture*. State Electoral Office of Estonia, 2019.

[Sta20] State Electoral Office of Estonia. *IVXV-protocol*. State Electoral Office of Estonia, 2020.

[Wik] Wikipedia. The 2023 Estonian parliamentary election - Wikipedia. https://en.wikipedia.org/wiki/2023_Estonian_parliamentary_election. (Accessed 12/03/2023).