Individual Verifiability and Revoting in the Estonian Internet Voting System

Olivier Pereira

August 16, 2021

Abstract

Individual Verifiability remains one of the main practical challenges in e-voting systems and, despite the central importance of this property, countries that sought to implement it faced repeated security problems.

In this note, we revisit this property in the context of the IVXV version of the Estonian voting system, which has been in used for the Estonian municipal elections of 2017 and for the Estonian and European parliamentary election of 2019.

We show that a compromised voter device can defeat the individual verifiability mechanism of the current Estonian voting system. Our attack takes advantage of the revoting option that is available in the Estonian voting system, and only requires compromise of the voting client application: it does not require compromising the mobile device verification app, or any server side component.

1 Introduction

Estonia remains the only country in the world where Internet voting is used by a large proportion of voters, all along since an initial deployment in 2005.

This unique status brought numerous questions regarding the security of the system. In particular, during the 2011 parliamentary election, in which around 24% of the voters submitted their ballot on the Internet [10], Paavo Pihelgas developed and tested malware that would compromise a voting client and modify a vote in a way that would be undetectable by the voter. Pihelgas filed a complaint to the Estonian Supreme Court, asking that they nullify the votes submitted by Internet, but the complaint was dismissed [7].

Nevertheless, in the 2013 elections, an extension of the Estonian voting system was introduced by Heiberg and Willemson [3], in order to offer *individual verifiability*, that is, a mechanism by which a voter can verify independently that the ballot submitted by her voting device accurately reflects her voting intent, even if that voting device is compromised.

The security of the Estonian voting system, including the individual verifiability mechanism, was challenged in a detailed analysis by Springall et al. in

2014, who explained how the individual verifiability could be circumvented [9]. The individual verifiability mechanism was also questioned in 2016 by Mus et al. [6], with a focus on its privacy implications, and a variant of that mechanism was proposed. That variant was in turn demonstrated to be flawed by Kubjas et al. in 2017 [5].

In the meantime, an almost complete redesign of the Estonian voting protocol was proposed in 2016 by Heiberg et al. [4]: the new IVXV protocol aims at making the server side operations verifiable by designated auditors, on top of offering the individual verifiability mechanism that focuses on the verification of the client-side operations.

Contribution The present note demonstrates that the IVXV protocol, used for the Estonian municipal elections of 2017 and for the Estonian and European parliamentary election of 2019, does not offer individual verifiability.

The attack that we propose only requires the voting client to be compromised: it succeeds even if all the server side components and the vote verification app are perfectly honest. It is also performed within a short time frame: it does not require the voter to leave his eID card on his computer for a long period of time, or to use that eID card on a regular basis, contrary to the *Ghost Click Attack* proposed by Springall et al. [9] on the 2013 version of the Estonian system.

2 IVXV Individual Verifiability Protocol

The IVXV voting system relies on the following components (we largely use the terminology and notations from [4], and only mention the components that are relevant for our purpose):

Certification Authority A national certification authority issues, for each voter, a signature key pair $(sk^{id}_{pub}, sk^{id}_{priv})$ that is stored in the smartcard of each citizen's eID card, and can be used to sign documents, provided that the user provides a PIN code. Obviously, the reliance on such a public key infrastructure introduces challenges on its own [8] but, for our analysis, we will trust that it is correctly implemented.

Election Organizer The *EO* approves the election configuration, and produces an encryption key pair (ek_{pub}, ek_{priv}) that will be used to encrypt the votes.

Voting Application The *VoteApp* is a standalone application, available for Windows, macOS and Linux, that the voter can download from the election authority website, and that is signed. It is used by the voters to express their vote intent, prepare the ballot, have it signed by the eID, submit the ballot, and interact with the Vote Verification application. The purpose of individual verifiability is to be able to detect if a corrupted

$VoteApp$ tries to modify the vote intent expressed by the voter and cast a ballot supporting a different candidate.

Vote Verification Application The $VerApp$ is a mobile application available for Android and iOS, that the voter can obtain from the respective App Stores of these environments. It is trusted that the adversary cannot compromise both the $VerApp$ and the $VoteApp$. For our purpose, we will assume that the $VerApp$ is honest.

Vote Collector The VC interacts with the $VoteApp$ and the $VerApp$: it collects the encrypted and signed ballot submitted by the $VoteApp$, and interacts with the $VerApp$ when a voter wants to verify that her vote captures her vote intent, and was correctly recorded.

Registration Service The RS interacts with the VC: every ballot received by the VC must also be signed by the RS. This signature will be verified by the $VerApp$.

In terms of security model, it is assumed that that VC and RS are not jointly compromised, and that $VoteApp$ and $VerApp$ are not jointly compromised either.

The IVXV voting process is then as follows (we simplify several internal server side message flows, in order to focus on the relevant aspects of the protocol):

Ballot submission A voter who would like to submit a vote v installs the $VoteApp$, obtains the election encryption public key ek_{pub}, picks randomness r from the appropriate group, and produces an ElGamal encryption of his vote as $c = Enc_{ek_{pub}}(v; r)$. The voter then uses his PIN code and eID to produce a signature $\sigma = Sign_{sk^{id}_{priv}}(c)$, and submits the resulting ballot $b = (c, \sigma)$ to VC.

Ballot registration VC creates an identifier vid for the ballot, verifies the eligibility of the voter and validity of the signature, obtains a time stamp on $H(c)$, and sends a registration request on a signature $\sigma_{VC} = Sign_{sk^{VC}_{priv}}(H(b))$ to RS, who returns a signed confirmation $reg = Sign_{sk^{RS}_{priv}}(H(\sigma_{VC}))$ to VC. This confirmation reg is sent back to the $VoteApp$ together with vid.

Vote verification When the $VoteApp$ displays the pair (vid, r) to the voter, as a QR code, the voter uses $VerApp$ to capture the (vid, r) pair. The verification app now queries VC in order to obtain the (b, reg) pair associated to vid. The $VerApp$ verifies the validity of $b = (c, \sigma)$ and reg, and displays the identity obtained through σ to the voter. Finally, it performs an exhaustive search on all the candidates until it finds a vote v such that $c = Enc_{ek_{pub}}(v; r)$. The matching v is displayed to the voter, who has to decide whether it is correct.

There are several remarks to be made about this process:

- An important feature of the Estonian Internet voting protocol is that voters have the possibility to submit as many ballots as desired. The last registered ballot will be included in the tally. This offers some level of protection against coercion: if a voter is forced to vote for a given candidate, it may remain possible to submit another ballot that will replace the first one in the tally.

- The Vote verification feature is obviously very sensitive from a privacy point of view: anyone in possession of the verification QR code has the possibility to obtain, in clear, the author and the content of a ballot. For that reason, the individual verifiability process is only authorized during a limited time frame, usually 30 minutes, after ballot submission. Besides, verification can only be performed a limited number of times (usually 3).

- The ballot verification process will not tell the $VerApp$ whether a ballot has been overwritten or not. In this way, a voter who was coerced into submitting his QR code to a third party can still submit a new ballot immediately, and the QR code will not inform the coercer of this revote. For similar reasons, there is no feedback channel to the voter, that would inform him that a ballot was submitted on his behave.

3 Revoting and Individual Verifiability in IVXV

The revoting feature of the IVXV Internet voting protocol raises particular difficulties for individual verifiability.

In particular, since there is no way for a voter to know whether a vote that is verified will be tallied, there are a variety of attack scenarios in which a compromised $VoteApp$ could fool the verification process. Assuming a compromised VC opens for even more attack scenarios.

Let us consider a voter who uses a compromised $VoteApp$ and intends to express vote v. A compromised $VoteApp$ could cast of vote v' on behave of the voter as follows:

1. When the voter expresses his vote intent v, $VoteApp$ runs the Ballot submission process honestly, and lets the server-side components run the Ballot registration phase. However, as soon as $VoteApp$ collects vid from VC, and before displaying it to the voter, it crashes. At this point, the voter has no idea whether his vote was actually registered or not.

2. As a result, the voter will be very likely to retry voting. But, this time, the $VoteApp$ will instead prepare a ballot encrypting the vote intent v', which will be signed with the voter's eID (since the voter intends to sign a ballot) and has no way of verifying the content of the ballot that it signs. That second ballot is again submitted to VC and registered, returning vid' to the $VoteApp$. From the point of view of the server, this is a normal revote, and that second ballot is the one that will be included in the tally.

3. Now, instead of displaying a QR code with the vid' reference, the $VoteApp$ displays the QR code corresponding to the first ballot, that is, vid and the randomness used to encode the first ciphertext.

4. When the voter scans this QR code, the first vote will be downloaded, and the vote intent v will be displayed to the voter, who will accept the verification steps, even though it is a vote for v' that will be tallied on his behave.

Of course, a $VoteApp$ that would systematically crash on the first voting attempt may raise suspicions. But random crashes are much less likely to attract attention, especially since the voter would still be able to vote and verify his vote.

In the *Ghost Click Attack* by Springall et al. [9], the corrupted user device would intercept the PIN code typed by the voter. Then, by emulating the PIN code typing process without any voter intervention, the corrupted $VoteApp$ can just obtain the signature of two ballots, as described above.

4 Mitigations

Feedback channel The addition of an independent vote confirmation channel could possibly help: the voter could be warned every time that a ballot is registered on her behave. In the security model considered for the IVXV voting system, such a mechanism may however be non trivial to design in a secure way: if the voter uses a compromised voting device, we must be sure that the malware present on the device is unable to delete and/or alter vote confirmations. Besides, since server side compromises are in-scope, we must be sure that the notifications that would alert a voter are actually sent. For instance, if the VC is in charge to notify the voter using a second communication channel, then a corrupted VC could simply choose to delay the notification for the first ballot of our scenario, and to send it when the second ballot arrives.

Verifying the last ballot only Another approach would be for the server to only answer verification queries for ballots that have not been replaced through a revote: in the scenario described above, a honest VC would refuse the verification of the first ballot after submission of the second.

However, in a variation on the attack scenario described above, the $VoteApp$ could submit the first ballot after obtaining the signature on the second ballot, wait until the end of the 30 minutes time period during which verification can be performed, and submit the second ballot at that time. This only requires the malware to be able to access the Internet for a slightly longer time period.

Accepting a single ballot In another approach, the voting protocol could be modified so that VC and RS only accept one single ballot per voter. This would prevent the attack scenarios described above, but would also decrease the coercion resistance of the voting system. Given that the level of coercion

resistance offered by an Internet voting system is low anyway, and that voters still have the option to vote in person and cancel their Internet ballot, this may be a viable option.

Creating a public bulletin board One more option would be to create a public bulletin board, on which a hash of each registered ballot would be displayed, and the last ballot of each voter clearly marked. This would make it possible for a voter to check at any time whether his ballot has been replaced by another one. However, this would also offer the possibility for a coercer to observe whether a voter revoted online.

Overall, it seems fairly non trivial to address the individual verifiability issue described above without affecting the receipt freeness of the protocol at the same time.

5 Conclusions

Individual verifiability is a central property that many recent Internet voting systems deployed for government elections tried to offer, and it is no surprise given the high risks of the presence of a malware on the voter's device.

It however showed to be particularly challenging to obtain an effective verification process: the first mechanism proposed for the Estonian voting system, used in the 2013 elections, was shown to be broken by Springall et al. [9]. In 2015, Halderman and Teague [2] showed how to circumvent the verification mechanisms of the iVote system used in New South Wales. In 2019, Haines et al. [1] demonstrated weaknesses in the individual (and universal) verifiability mechanisms of the Swiss Post/Scytl Internet voting system used in Switzerland (for individual verifiability only). The current paper shows that the latest IVXV protocol used in Estonia still fails to offer individual verifiability.

The situation definitely prompts for more caution regarding the security properties that are announced for voting systems, in particular when they are proposed for use in public elections. Proper security definitions, proofs, and careful reviews seem to remain the best strategy we have.

Acknowledgement

We would like to thank Sven Heiberg for confirming that the attack scenario described above would work on the current implementation of the IVXV protocol and for his helpful and constructive comments. We also would like to thank Vanessa Teague for so many interesting discussions on the security of voting systems and for her review of a previous version of this document. Any error would of course be the responsibility of the author.

References

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

[2] J. Alex Halderman and Vanessa Teague. The new south wales ivote system: Security failures and verification flaws in a live online election. In *E-Voting and Identity - 5th International Conference, VoteID 2015*, volume 9269 of *Lecture Notes in Computer Science*, pages 35–53. Springer, 2015.

[3] S. Heiberg and J. Willemson. Verifiable internet voting in estonia. *6th International Conference on Electronic Voting: Verifying the Vote (EVOTE)*, pages 1–8, 2014.

[4] Sven Heiberg, Tarvi Martens, Priit Vinkel, and Jan Willemson. Improving the verifiability of the estonian internet voting scheme. In Robert Krimmer, Melanie Volkamer, Jordi Barrat, Josh Benaloh, Nicole J. Goodman, Peter Y. A. Ryan, and Vanessa Teague, editors, *Electronic Voting - First International Joint Conference, E-Vote-ID 2016*, volume 10141 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 2016.

[5] Ivo Kubjas, Tiit Pikma, and Jan Willemson. Estonian voting verification mechanism revisited again. In *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017*, volume 10615 of *Lecture Notes in Computer Science*, pages 306–317. Springer, 2017.

[6] Koksal Mus, Mehmet Sabir Kiraz, Murat Cenk, and Isa Sertkaya. Estonian voting verification mechanism revisited. *CoRR*, abs/1612.00668, 2016.

[7] OSCE/ODIHR. Estonia parliamentary elections – 6 march 2011 – osce/odihr election assessment mission report. `https://www.osce.org/files/f/documents/a/9/77557.pdf`, May 2011.

[8] Arnis Parsovs. Estonian electronic identity card: Security flaws in key management. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020*, pages 1785–1802. USENIX Association, 2020.

[9] 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 Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 703–715. ACM, 2014.

[10] Valimised. Statistics about internet voting in estonia. `https://www.valimised.ee/en/archive/statistics-about-internet-voting-estonia`.