Individual Verifiability and Revoting in the Estonian Internet Voting System

Olivier Pereira

August 30, 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 elections 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 [12], 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 [9].

Nevertheless, in the 2013 elections, an extension of the Estonian voting system was introduced by Heiberg and Willemson [4], 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 2013 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 [11]. The individual verifiability mechanism was also questioned in 2016 by Mus et al. [8], 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 [7].

In the meantime, an almost complete redesign of the Estonian voting protocol was proposed in 2016 by Heiberg et al. [6]: the new IVXV protocol aims at making the server side operations verifiable by designated auditors, on top of offering a (revised) 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, which is precisely what individual verifiability is supposed to detect. The attack succeeds even if all the server side components and the vote verification app are perfectly honest. It is also independent of any weakness in the voter identification mechanism, and would then still work even if a stronger voter identification mechanism were implemented (e.g., real two-factor). Finally, it can be performed within a short time frame.

These last two features differ from the *Ghost Click* attack on individual verifiability that was proposed by Springall et al. [11] on the 2013 version of the Estonian system: that attack required the adversary to be able to intercept the voter eID PIN code, and the voter to leave his eID card on his computer for a long period of time, or to use the eID card on a regular basis. These differences come thanks to changes introduced in the IVXV protocol.

2 IVXV Individual Verifiability Protocol

The IVXV voting system relies on the following components – we largely use the terminology and notations from the original description [6], 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. A separate authentication key pair is also provided on the eID, and is used to identify a voter connecting to the vote collection server. Obviously, the reliance on such a public key infrastructure introduces challenges on its own [10] 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 ballot 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 $VoteApp$, creates a TLS channel with the VC, identify himself with the eID authentication key and PIN code, 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. Furthermore, 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, which we do not detail here.

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 may be expected 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 he 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 crashes may raise suspicions. However, we must remember that the $VoteApp$ is an application that is unique to each election. So, the crash would just happen on the first attempt at using this application, and would not repeat after that, making it harder to reproduce and report. And, even in case of reporting, a compromised voter platform could report running the honest $VoteApp$. The benign aspect of the crash would be further confirmed by a successful vote verification process. Eventually, the application crash may also happen with a limited probability only.

Random crashes may also be replaced by the compromised $VoteApp$ intercepting the eID PIN code, and revoting silently on the voter behave, as in the *Ghost Click* attack [11] on the 2013 version of the voting system. We further discuss this variation and the changes between the 2013 protocol and IVXV protocol in Section 5.

4 Mitigations

There many directions that can be explored in order to mitigate the attack described above, including those that are discussed by Heiberg et al. [5]. However, those that may be effective appear to come with relatively important changes in the system and to have a noticeable impact on other properties of the system.

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, and the vid could be included in the notification. The addition of vid, or of something to the same effect, is crucial in order to give the voter an opportunity to check whether he is verifying his first or second ballot. This feature is absent from previous discussions [5], which focused on a malicious app stealing voter credentials, in which case it is enough to inform the voter than *a* vote was submitted.

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 and delivered in the right order. For instance, if the VC is in charge of notifying 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.

This protection could however be defeated in several variations of our attack. For instance:

- If the VC is corrupted, then it can decide to participate in the verification of the first ballot only, which would make the voter wrongly believe that no second ballot was submitted. The corrupted VC could also receive the second ballot from the corrupted $VoteApp$ and wait for 30 minutes until it timestamps it and registers it with RS, so that this ballot would only start to "officially exist" after the end of the verification period of the first ballot.

- If the corrupted $VoteApp$ is able to capture the eID PIN code, and if the eID remains or becomes available on the voter computer more than 30 minutes after the submission of the initial ballot, then $VoteApp$ can silently submit a second ballot at that time, without the voter's knowledge. This is essentially the Ghost Click attack.

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. A new mechanism would then be needed in order to support the verifiability of the submitted ballot even if the $VoteApp$ or the network crashes after the ballot submission but before the confirmation message could be sent to the 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 not be too damaging.

An alternate option that would limit the privacy/coercion risk would be to switch to the so-called Benaloh challenge approach [1], in which voters can either verify a ballot produced by the voting client, or cast it as their vote, but not both. Extra care would then be needed in order to make sure that the expected ballot is considered as cast, and not a challenged one for instance.

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 with another one. However, this would also offer the possibility for a coercer to observe whether a voter revoted online. The implementation of a secure public bulletin board may also be a challenge.

Heiberg et al. [5] also discuss various directions linked to the use of the eID (e.g., require the signature of two independent devices, promote the use of card readers that have their PIN-pad, increase the control of the use of the eID, ...). These approaches may help against an attacker who targets voter credentials, but do not help in our case since, in our attack, the voter is willing to produce two signed ballots.

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, or creating new usability challenges.

5 Comparison with the 2013 Internet voting protocol

The individual verification process that was introduced in the 2013 version of the Internet voting protocol [4] differs in several significant ways from the IVXV protocol that was deployed in 2017.

First, the trust model is different: the 2013 protocol assumes the honesty of the server receiving the ballots and playing the ballot verification protocol, while the IVXV protocol is expected to remain secure as long as only one of VC and RS is honest.

The protocol is also different:

- The $VerApp$ in the 2013 protocol would only receive the ciphertext c from the voting server, and not the voter signature σ (or the RS signature reg), while c, σ (and reg) are sent and verified in IVXV. Keeping the signature off would limit the risks of taking advantage of the verification process in order to prove how a voter voted, since the verification data remain anonymous. This of course introduced the risks of clash attacks: several voters voting in the same way could be pointed to one single ciphertext, while a different ciphertext would be signed by all but one voter. A limit of 3 verifications for each ballot was introduced in order to limit the scale of such an attack.

- The server of the 2013 protocol would only allow the verification of the last vote cast by the voter, that is, the one that will be included in the tally. In contrast, the IVXV protocol allows the verification of any ballot as long as the verification takes places within 30 minutes after submission. As a result, a voter who submits a first ballot under the watch of a coercer does not need to wait for 30 minutes before submitting a new ballot: a verification of the first ballot by the coercer would still succeed.

Those two differences only make sense because the voting server is trusted in the 2013 protocol. A corrupted server could indeed send any ciphertext encrypted with the right randomness to the $VerApp$,[1] since it is not bound to send a full ballot signed by the voter, and the voter would have no way to determine whether the ciphertext that is verified is the one that will be included in the tally. Similarly, a corrupted verification server would also have no reason to restrict the verification process to the last submitted ballot.

It turns out that the changes introduced in the IVXV protocol seem to weaken the individual verifiability of the protocol rather than to strengthen it.

The addition of the signature in IVXV does not change anything to our attack, and it does not do anything to prevent the Ghost Click attack [11] either: in both attacks, the ballots that are submitted have a legitimate signature.

The fact that the verification of any submitted ballot is possible for 30 minutes (and not just of the verification of the last ballot) simplifies the attack process. We take advantage of this by having the voter submitting two ballots within a short time window, and letting the voter verify a vote that has been replaced by a more recent one.

It would be possible to adapt our attack to the 2013 version of the protocol if the voting server is corrupted: the malicious $VoteApp$ could obtain two voter signed ballots within a short time frame, and the voting server would just keep the second ballot in memory for 30 minutes before registering it within the system. (In the presence of a honest voting server, we could not have an attack in which the $VoteApp$, or a third party, would keep the second ballot in memory for 30 minutes and submit it then: the eID is also needed for the voter authentication to the server in order to submit a ballot. Of course, if the eID is left on the device and if the corrupted $VoteApp$ has the PIN code and can silently interact with the eID, then the problem is solved.)

Similarly, the Ghost Click attack would work more easily on the IVXV protocol than on the 2013 version of the protocol: if the eID PIN code can be intercepted, then two signed ballots can be obtained in a matter of seconds, and the corrupted $VoteApp$ can first submit the ballot reflecting the voter intent, then show the verification data to the voter, and immediately submit the second signed ballot without letting the voter know that it happened.

Whether or not our attack is easier to deploy than the Ghost Click attack depends on whether the eID PIN code of the voter can be intercepted or not. This may not be particularly challenging when the client is a stand-alone application that is corrupted, or when the voter device is completely under adversarial control. It could make a difference if the voting client were a web application, in which case the introduction of the PIN code would be handled directly by a

[1]This is slightly more challenging in the case of the 2013 protocol which uses RSA-OAEP for encryption, compared to the ElGamal encryption used in the IVXV protocol. Indeed, the server would need to obtain the randomness used for encryption in order to produce a different ciphertext encrypted with the same randomness, while ElGamal encryption is just malleable.

browser extension interacting with the eID card, and not visible by a corrupted voting application. Besides, many of the mitigations explored by Heiberg et al. [5] would protect against the ghost click attack, but they would be ineffective against the attack proposed here: they seek to inform a voter when ballots are submitted without the voter's knowledge, but this is not the case here.

6 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. [11]. In 2015, Halderman and Teague [3] showed how to circumvent the verification mechanisms of the iVote system used in New South Wales. In 2019, Haines et al. [2] 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 in this note 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] Josh Benaloh. Simple verifiable elections. In Dan S. Wallach and Ronald L. Rivest, editors, *2006 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06*. USENIX Association, 2006.

[2] 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.

[3] 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.

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

[5] Sven Heiberg, Kristjan Krips, and Jan Willemson. Planning the next steps for Estonian Internet voting. In *Proceedings E-Vote-ID 2020*, pages 82–97, 2020.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

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