Estonian Voting Verification Mechanism Revisited Again

Ivo Kubjas[1], Tiit Pikma[1], and Jan Willemson[2,3]

[1] Smartmatic-Cybernetica Centre of Excellence for Internet Voting
Ülikooli 2, 51003 Tartu, Estonia
{ivo,tiit}@ivotingcentre.ee
[2] Cybernetica AS
Ülikooli 2, 51003 Tartu, Estonia
janwil@cyber.ee
[3] Software Technology and Applications Competence Center
Ülikooli 2, 51003 Tartu, Estonia

Abstract. Recently, Muş, Kiraz, Cenk and Sertkaya proposed an improvement over the present Estonian Internet voting vote verification scheme [6, 7]. This paper points to the weaknesses and questionable design choices of the new scheme. We show that the scheme does not fix the vote privacy issue it claims to. It also introduces a way for a malicious voting application to manipulate the vote without being detected by the verification mechanism, hence breaking the cast-as-intended property. In addition, the proposal would seriously harm usability of the Estonian vote verification scheme.

1 Introduction

Estonia is one of the pioneers in Internet voting. First feasibility studies were conducted already in early 2000s, and the first legally binding country-wide election event with the option of casting the vote over Internet was conducted in 2005. Up to 2015, this mode of voting has been available on every one of the 8 elections. In 2014 European Parliament and 2015 Parliamentary elections, more than 30% of all the votes were cast over Internet [9].

During the period 2005–2011, the basic protocol stayed essentially the same, mimicking double envelope postal voting. The effect of the inner envelope was achieved by encrypting the vote with server's public key, and the signed outer envelope was replaced by using a national eID signing device (ID card, Mobile-ID or Digi-ID) [1].

In 2011, several potential attacks were observed against this rather simple scheme. The most significant one of them was developed by a student who implemented proof-of-concept malware that could have either changed or blocked the vote without the voter noticing it.

To counter such attacks, an individual verification mechanism was developed for the 2013 elections [5]. The mechanism makes use of an independent mobile computing device that downloads the vote cryptogram from the storage server and brute forces it using the encryption random seed, obtained from the voter's computer via a QR code. The value of the vote corresponding to the downloaded cryptogram is then displayed on the device screen, and the voter has to make the decision about its match to her intent in her head.

The complete voting and verification protocol is shown in Figure 1.

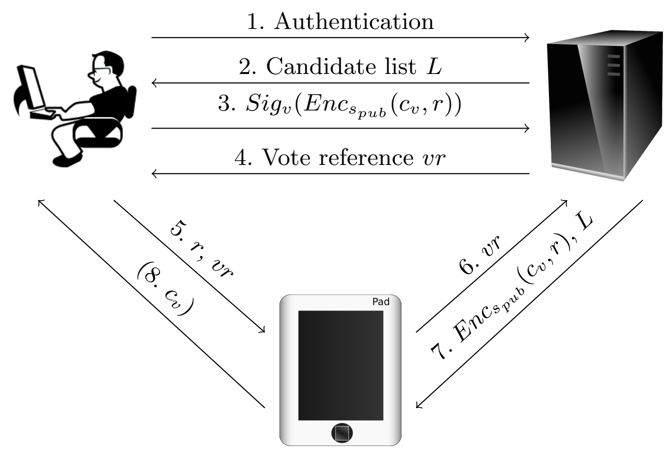

Fig. 1. Estonian Internet voting and verification protocol

In the figure, c_v stands for the voter's choice, r is the random seed used for encryption, vr is the vote reference used to identify the vote on the server and s_{pub} is the election system's public key.

A recent report by Muş *et al.* [6, 7] discusses the Estonian vote verification scheme and draws attention to its weak privacy properties. It also proposes an improvement over the existing system (we will give technical details of the proposal in Section 2.2). The objective of this paper is to dispute the motivation of [6] and show vulnerabilities of the proposed improvement.

2 Analysis of the scheme by Muş *et al.*

2.1 Assumptions and motivation of [6]

Individual vote verification was introduced to Estonian Internet voting scheme in 2013 to detect potential vote manipulation attacks in the voter's computer [1, 5]. It was never designed as a privacy measure for a very simple reason.

Since the verification application needs access to the QR code displayed on the screen of the voter's computer, verification can only happen in close physical proximity of the voting action.[4] But if this is the case, the verifier can anyway

[4] Of course we assume here that the voter's computer is honest in the sense that it does not send the QR code anywhere else. But if it would be willing to do so in order to break the voter's privacy, it could already send away the vote itself.

observe the vote on the computer screen. For this reason we disagree that the potential privacy leak from the verification application makes vote buying attacks easier, as claimed in [6].

It is true that a malicious verification application sending the vote out of the device would be unintended behavior. However, the authors of [6] make several false assessments analyzing this scenario.

Firstly they claim that "all voter details including the real vote are displayed by the verification device." In fact, the vote is the *only* piece of data actually displayed. Note that following the protocol [5], the verification device only obtains the vote encrypted with the voting system's public key. The signature is being dropped before the cryptogram is sent out for verification from the server, so the verification device has no idea whose vote it is actually verifying.

Sure, a malicious verification application can make some educated guesses about the owner of the device by looking at social media accounts, emails, etc. However, this is something a privacy-concerned voter can counter by using a truly independent verification application/device. The voter may for example borrow a mobile device from a friend or a family member.

Second, the authors of [6] argue that verification privacy leaks may be aggregated to obtain the partial results of the election before it has concluded. We feel that this scenario is too far-fetched. First, only about 4% of the Internet voters actually verify their votes [3]. Also, nothing is known about the preference biases the verifiers may have, so the partial results obtained would be rather low-quality. There are much easier, better-quality and completely legal methods of obtaining the result (like polls). Hence this part of the motivation is not very convincing.

Third, getting the user to accept a malicious verification application from the app store is not as trivial as the report [6] assumes. For example Google Play store displays various reliability information about the application like the number it has been installed and the average mark given by the users. When the voter sees several competing applications, a smaller number of installations should already give the first hint that this is not the officially recommended verification app.

At the time of this writing (December 2016), the official application "Valimised"[5] is the only one under that or similar name, with more than 10,000 installations and an average score of about 3.6 points out of 5. If the attacker wants to roll out his own version, he would need to beat those numbers first. Occurrence of an alternative verification app is completely acceptable *per se*, but it will be widely visible. App stores can and are being constantly monitored, and any independent verification apps would undergo an investigation. In case malicious behavior is detected, the malicious applications can be requested to be removed from the app store.

[5] "Valimised" means "Elections" in Estonian.

2.2 Description of the scheme

The scheme proposed in [6] extends the Estonian vote verification protocol by adding another parameter q to the scheme. The role of q is to serve as a random, voter-picked verification code that will be encrypted using the hash of the vote cryptogram $h = H(Enc_{s_{pub}}(c_v, r))$ as a symmetric key (see Figure 2).

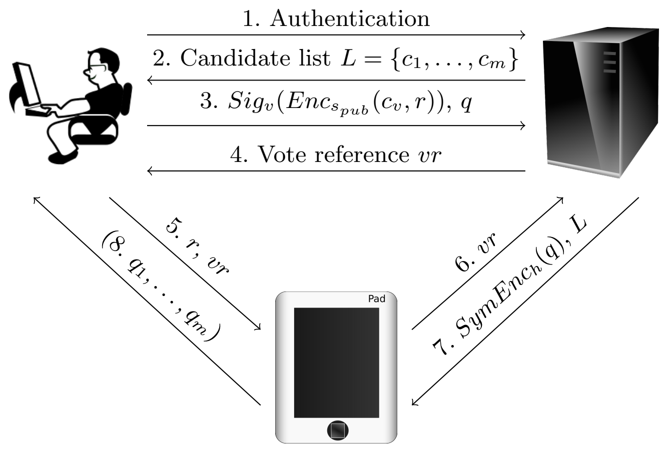

Fig. 2. Proposed update to the Estonian protocol

The verification mechanism will also be altered accordingly. In the original Estonian verification scheme, the verification application goes through the candidate list and tries to re-create vote cryptogram, using the random seed obtained from the voting application via a QR code. In the modification proposed by [6], the candidate list is also traversed in a similar manner, but the hashes of all the vote cryptogram candidates are used as symmetric keys to try to decrypt q.

The trick is that even an incorrect symmetric decryption key leads to some sort of a decrypted value q_i, so that the task of the verifier becomes recognizing the correct one in the list of decrypted values $q_1, q_2, \dots, q_m$ (where m is the number of election candidates) displayed to her.

More formally, let us have the candidate list $L = \{c_1, c_2, \dots, c_m\}$. The verification application computes $h_i = H(Enc_{s_{pub}}(c_i, r))$ for $i = 1, 2, \dots, m$ and displays the list $\{q_1, q_2, \dots, q_m\}$ where

$$q_i = SymDec_{h_i}(SymEnc_h(q)) \quad (i = 1, 2, \dots, m) \ . \tag{1}$$

The voter accepts verification if $q = q_i$, where c_i was the candidate of her choice.

2.3 Analysis of the scheme – privacy and usability

Even though clever conceptually, the scheme of Muş *et al.* fails in usability, and this will unfortunately lead to considerable weakening of the protocol.

First and foremost, humans are notoriously poor random number generators [10]. This is also acknowledged by the authors of the scheme, so they propose not to require the user to generate the entire value of q, but only 32 rightmost bits denoted as q_{right}. The remaining bits q_{left} would be generated by the voting application, so that $q = q_{\mathsf{left}} \parallel q_{\mathsf{right}}$. In the authors' vision, the 32 bits could be asked from the voter in the form of 4 characters, and these characters would later also be displayed on the screen of the verification device.

Such an approach would assume that every possible byte has a corresponding keyboard character. However, this is clearly not true. Capital and lower-case letters, numbers and more common punctuation marks altogether give about 70–75 symbols, which amounts to slightly over 6 bits of entropy. Hence, four-letter human entered codes can in practice have no more than 25 bits worth of randomness.

Achieving this theoretical maximum assumes that humans would select every character for every position equally likely and independently. This is clearly not the case, and a relatively small set of strings like "1234", "aaaa" or "qwer" may be expected to occur much more frequently than others. This observation gives the first simple attack against the proposed scheme – the attacker can observe the output of the verification application and look for some of these frequent codes.

Even if the voter takes care and selects a rather random-looking 4-character pattern, the attacker still has a remarkable edge. Namely, when the 32-bit parts of the decrypted values are converted into characters, some of these characters may fall out of the $\sim$75 character set. In fact, several of the 256 possible byte values do not have a printable character assigned to them at all. Spotting such a code, the adversary can disregard that one immediately.

To give a rough quantification of the attacker's success probability, assume that the set $\mathcal{C}$ of characters used by the voter to input a code consists of 75 elements. When in the equation (1) we have $h \neq h_i$, the resulting values q_i (and their 4-byte code parts $q_{i,\mathsf{right}}$) are essentially random (assuming the underlying symmetric encryption-decryption primitive behaves as a pseudorandom permutation).

This means that the probability that one single character of an incorrect $q_{i,\mathsf{right}}$ falls outside of the set $\mathcal{C}$ is $\frac{256-75}{256} \approx 0.707$. The probability that at least one of the four characters falls outside of this set is

$$1 - \left(1 - \frac{256-75}{256}\right)^4 \approx 0.993$$

which is very-very high. The attacker will have an excellent chance of spotting the correct code, since with very high probability there are only very few candidates $q_{i,\mathsf{right}}$ that have all the characters belonging to the set $\mathcal{C}$. This observation completely breaks the privacy claims of [6].

Another usability problem is the need to display the list of all candidate values of q_{right} on the screen of the verification device. The list has the same number of elements as there are candidates in a given district. In case of Estonian elections, this number varies between tens and hundreds, with the most extreme cases reaching over 400. It is unrealistic to expect the voter to scroll through this amount of unintuitive values on a small screen.

2.4 Vote manipulation attack

The core motivation of introducing an individual verifiability mechanism is to detect vote manipulation attacks by a malicious voting application. In this Section we show that with the updates proposed by Muş *et al.*, vote manipulation attacks actually become very easy to implement.

Consider an attack model where the attacker wants to increase the number of votes for a particular candidate c_j by manipulating the voting application or its operational environment. The key to circumventing detection by the verification mechanism is to observe that the voting application has a lot of freedom when choosing two random values – r for randomizing the encryption and q_{left} for padding the voter-input code. By choosing these values specifically (even the freedom of choosing r is sufficient), a malicious voting application can make the vote it submitted for c_j to verify as a vote for almost any other candidate c_i.

To implement the attack, the attacker needs a pre-computation phase. During this phase, the attacker fixes his preferred choice c_j and the encryption randomness $r^\star \in \mathcal{R}$, and computes $h = H(Enc_{s_{pub}}(c_j, r^\star))$. The attacker can also set his own q arbitrarily, say, $q = 00\ldots0$.

For every possible pair of voter choice $c_i \in L$ and $q_{\mathsf{right}} \in \{0, 1, \ldots, 2^{32} - 1\}$, the attacker tries to find a suitable encryption randomness $r_{i,q_{\mathsf{right}}}$ that would give the last 32 bits of q' being equal to q_{right}, where

$$h' = H(Enc_{s_{pub}}(c_i, r_{i,q_{\mathsf{right}}})) \quad \text{and} \quad q' = SymDec_{h'}(SymEnc_h(00\ldots0)) \,. \tag{2}$$

If the attacker succeeds in finding such a $r_{i,q_{\mathsf{right}}}$, then later during the voting phase he casts his vote to the server as $Enc_{s_{pub}}(c_j, r^\star)$, but sends $r_{i,q_{\mathsf{right}}}$ to the verification application. This random seed will cause the voter picked q_{right} to occur next to the voter's choice c_i. The leftmost non–voter chosen bits of q' would not match, but they are not important, since they are not shown to the voter anyway.

The pre-computed values of encryption randomness for all candidates can be tabulated as in Table 1.

Note that only the last column of this table needs to be stored. Hence the size of required storage is $2^{32} \log_2 |\mathcal{R}| m$, where $\log_2 |\mathcal{R}|$ is the number of bits required for representing elements in the randomness space $\mathcal{R}$. In practice, the length of the random value is not more than 2048 bits. This means that the size of the database is $1024m$ GB. By restricting the randomness space (for example, by fixing some bits of the random value), we can decrease the table size.

Another option of limiting the storage requirement is referring to the observations described in Section 2.3. Human users will not be able to make use of the whole 2^{32} element code space, but at most 2^{25}. This will bring the storage requirement down 2^7 times to only $8m$ GB. If the attacker is willing to settle only with the most common codes, the table will become really small.

Table 1. Pre-computation dictionary

q_{right}	choice c_i	$r_{i,q_{\mathsf{right}}}$
0	c_1	$r_{1,0}$
$\vdots$	$\vdots$	$\vdots$
$2^{32}-1$	c_1	$r_{1,2^{32}-1}$
0	c_2	$r_{2,0}$
$\vdots$	$\vdots$	$\vdots$
$2^{32}-1$	c_2	$r_{2,2^{32}-1}$
$\vdots$	$\vdots$	$\vdots$
0	c_m	$r_{m,0}$
$\vdots$	$\vdots$	$\vdots$
$2^{32}-1$	c_m	$r_{m,2^{32}-1}$

Even without reducing the table size, storing it is feasible as hard drives of several TB are readily available. A malicious voting application only needs one online query per vote to this database, hence the attacker can for example set the query service up in a cloud environment.

There are several possible strategies for filling Table 1. We suggest starting from the choice and randomness columns (selecting the randomness truly randomly) and computing the corresponding q_{right} values. In the latter case the computation complexity of the pre-computation phase is $2^{32}m$ times one asymmetric encryption, one hash function application and one symmetric decryption (see equations (2)). This amount of computation is feasible even for an ordinary office PC.

This strategy is not guaranteed to 100% succeed, since we may hit the same value of q_{right} for different inputs $r_{i,q_{\mathsf{right}}}$. To estimate the success probability, consider generating the table for a fixed election candidate c_i. Let us generate $N = 2^{32}$ random values and use them to compute the corresponding values q_{right} using the equations (2).

The probability of one specific q_{right} *not* being hit in one attempt is $\frac{N-1}{N}$. Consequently, the probability of *not* hitting it in N attempts is

$$\left(\frac{N-1}{N}\right)^N \approx \frac{1}{e}\,.$$

Hence, the expected probability of hitting one specific value at least once is $1-\frac{1}{e}\approx 0.63$.

By linearity of expectation, we may conclude that using $2^{32}m$ computation rounds, about 63% of the whole table will be filled.

This percentage can be increased allowing more time for computations. For example, if we would make twice as many experiments, we would get the expected success probability

$$1 - \left(\frac{N-1}{N}\right)^{2N} \approx 1 - \frac{1}{e^2} \approx 0.86\,.$$

Allowing four times more computation time would give us already more than 98% of the values for q_{right} filled.

Hence we obtain a vote manipulation attack by a malicious voting application with very high success rate, essentially invalidating Theorem 2 of [6].

Note that in order to implement this attack, it is not necessary to manipulate the actual voting application. It is sufficient for the attacker to be able to only change the values of the vote, random seed and q. He can achieve this e.g. by manipulating suitable bytes in voter computer's memory, similar to the vote invalidation attack from 2015 Estonian Parliamentary elections [4]. The random value transferred from the voting application to the verification application can be manipulated by overlaying the QR code that carries it on the voter computer's screen similar to the Student's Attack of 2011 [1].

3 Conclusions and further work

Even though vote privacy was not the primary design goal of the Estonian vote verification application, it would of course be nice to have extra privacy protection capabilities. Unfortunately, the proposal made in [6] does not satisfy elementary usability requirements and is even at the voter's best effort still completely vulnerable by just looking at the characters used by the code candidates.

Also, we have demonstrated a vote manipulation attack that can be implemented with reasonable amount of pre-computation by an attacker who manages to compromise the voting application or voter's computer. As a result, the verification application does not fulfill its purpose of ensuring correct operation of the voting application.

Of course there is still a lot to do to improve the Estonian vote verification scheme. Better privacy protection would be desirable, but a much more urgent open problem is guaranteeing independence of the voting and verification platforms [8]. This problem may find its solution when hardware isolation capabilities of mobile platforms improve, but the level of this improvement remains to be seen in near future.

The upcoming new Estonian Internet voting protocol will introduce better digital ballot box integrity measures [2]. One crucial part of these measures is also augmenting the vote verification app with the capability to check the correctness of vote registration with the new Registration Authority. This part of the verification process remains the subject for future development as well.

References

1. Heiberg, S., Laud, P., Willemson, J.: The application of i-voting for Estonian parliamentary elections of 2011. In: International Conference on E-Voting and Identity. LNCS, vol. 7187, pp. 208–223. Springer (2011)

2. Heiberg, S., Martens, T., Vinkel, P., Willemson, J.: Improving the verifiability of the Estonian Internet Voting scheme. In: The International Conference on Electronic Voting E-Vote-ID 2016. pp. 213–229. TUT Press (2016)
3. Heiberg, S., Parsovs, A., Willemson, J.: Log Analysis of Estonian Internet Voting 2013–2014. In: Haenni, R., Koenig, R.E., Wikström, D. (eds.) E-Voting and Identity: 5th International Conference, VoteID 2015, Bern, Switzerland, September 2-4, 2015, Proceedings. LNCS, vol. 9269, pp. 19–34. Springer International Publishing (2015)
4. Heiberg, S., Parsovs, A., Willemson, J.: Log Analysis of Estonian Internet Voting 2013–2015. Cryptology ePrint Archive, Report 2015/1211 (2015), http://eprint.iacr.org/2015/1211
5. Heiberg, S., Willemson, J.: Verifiable Internet Voting in Estonia. In: Electronic Voting: Verifying the Vote (EVOTE), 2014 6th International Conference on. pp. 1–8. IEEE (2014)
6. Muş, K., Kiraz, M.S., Cenk, M., Sertkaya, I.: Estonian Voting Verication Mechanism Revisited. Cryptology ePrint Archive, Report 2016/1125 (2016), http://eprint.iacr.org/2016/1125
7. Muş, K., Kiraz, M.S., Cenk, M., Sertkaya, I.: Estonian Voting Verication Mechanism Revisited. arXiv:1612.00668v1 (2016), https://arxiv.org/abs/1612.00668v1
8. Springall, D., Finkenauer, T., Durumeric, Z., Kitcat, J., Hursti, H., MacAlpine, M., Halderman, J.A.: Security analysis of the Estonian internet voting system. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 703–715. ACM (2014)
9. Vinkel, P., Krimmer, R.: The How and Why to Internet Voting: An Attempt to Explain E-Stonia. In: The International Conference on Electronic Voting E-Vote-ID 2016. pp. 239–253. TUT Press (2016)
10. Wagenaar, W.A.: Generation of random sequences by human subjects: A critical survey of literature. Psychological Bulletin 77(1), 65 (1972)