CYBERNETICA

Mobile voting feasibility study and risk analysis

Technical document

Version 1.0

April 16, 2020

80 pages

Doc. T-184-5

Contents

1 Introduction

Voting is the core method of implementing public power in modern democratic societies. However, in the contemporary increasingly mobile world, paper-based elections, where every eligible voter has to come to the same physical location during a short period of time, is less and less of an option.

According to United Nations Migration Report of 2015 [UNM16], the number of people not living in their country (or even continent) of origin has increased by more than 30% worldwide during the period 2000–2015. This means that methods of remote vote casting have to be introduced sooner or later.

Physical polling station based elections were put into a completely new light due to the COVID-19 virus outbreak in early 2020 when it suddenly became strongly non-recommended for people to gather in small spaces. For example, there were local elections in France on March 15th which were held under a severe risk of spreading the virus [Cor20]. A reliable and widely used method of remote voting would have helped to alleviate the problem significantly.

Historically, remote paper voting goes back to late 19th-century Australia [SAL09], and it has been used ever since in many counties to allow for absentees to cast their ballots. However, this method has numerous shortcomings. It is hard to remotely authenticate the voter on paper, reliability of postal services varies a lot across the world, there are no good measures against vote selling, etc. This is why it is important to look for good alternatives for remote voting.

Recent decades have given us fast development in both computerized networks and strong electronic identification mechanisms. These together lay a foundation for vote casting over Internet. And indeed, this approach has been tried out in several parts of the World.

Casting a vote via Internet has been an option for Estonian electorate since 2005. By 2019, this option has grown to be the second favourite for Estonian voters, with about 44% of votes being cast this way during the 2019 Parliamentary elections[1].

During the whole period on 2005–2019, the only kind of devices supported for vote casting has been a PC running Windows, macOS or Linux operating system. However, recent rise in mobile platforms usage[2] has put forward a natural question whether casting votes from, say, a smartphone or tablet would be a viable option, too. The goal of this report is to seek answer to this and other implied questions.

[1] `https://rk2019.valimised.ee/et/participation/participation.html`

[2] `https://ec.europa.eu/eurostat/tgm/table.do?tab=table&init=1&plugin=1&language=en&pcode=tin00083`

2 Executive summary

When procuring for the current report, the procurers sought answers for a few specific questions. In this chapter, we will summarise the main conclusions that the report delivers in respect to these questions.

2.1 Security issues

2.1.1 What are the additional security risks of using smart devices for i-voting in general, and what risks are lost compared to the existing solution?

In many ways, mobile platforms of today are getting more and more similar to desktop platforms. This includes both security risks and defence mechanisms. However, there are still differences that may play a crucial role when taking decisions concerning their security level.

1. Android vendors are not always interested in keeping their platforms up-to-date and instead rely on selling new devices as their business model. As an unintended consequence, there are many outdated and vulnerable devices in use.
2. Small screens push vendors towards user interface trade-offs that may potentially harm security. E.g. mobile browsers may hide the URL bar, display the certificate information only partially, etc.

2.1.2 What specific security risks do we need to consider for different technical solutions?

The most significant attack vector against both PC- and mobile-based voting clients includes a potential malware that can make use of OS level vulnerabilities and take control over the voter's e-ID (e.g. by emulating the e-ID user interface). There are a number of possible mitigation measures:

- promote individual vote verification,
- promote usage of PIN-pad-enabled ID-cards and readers,
- add freshness notification to vote verification,
- establish a feedback channel notifying the voter when a vote has been cast on her behalf.

Note that the underlying problem and the respective mitigations are not specific to mobile voting and should be considered independent of whether mobile voting will be implemented or not.

Also, the vote collector service is currently trusted in correctly forwarding the mobile-ID signature hash. Ideally, this trust assumption should be removed by not relaying this hash through the vote collector service (see Section A.6.3.2 for further details).

2.2 Technical solution

2.2.1 Could the voter app work in a browser or should one create a separate voter app for each supported smartphone operating system (Android, iOS)?

In case of a separate app, we could use many of the proven security mechanisms of the current PC-based voter application. Introducing browsers into the stack would expose the solution to a number of additional problems and vulnerabilities (see Section 3.4). We advise against a browser-based voting application.

2.2.2 If one would choose a browser-based solution, will it replace the voter applications currently used on computers?

A browser-based solution indeed does have a potential to replace the current voting applications, but as said above, we advise against browser-based voting.

2.2.3 Should the existing individual verification solution be changed?

There is an option of doing so and letting a PC or browser-based verification application to verify votes. Security level of such a solution is largely comparable to the present verification protocol (with the main difference that the voting device would become aware of the fact of verification, which in turn can lead to some new attack vectors from a malicious voting application, see Appendix A for further details on the proposed alternative solution).

However, whether a new verification *should* be introduced, depends on how many verifiers would be trying to trick the verification system to use the same device for both voting and verification (assuming both apps would be available for mobile platforms). The only way to answer this question is to run user studies either in a lab environment or on a real system. It is also possible to deploy the current and new verification mechanisms in parallel.

In any case, the real problem is how to increase the share of voters who actually verify their votes. In a free society this can only be done by educating the electorate.

2.3 I-voting platform

2.3.1 How does the usage and complexity of the voter application change for different solutions (installation, user experience, and accessibility)?

Installing and using a mobile voting app would be very much like in case of any other mobile app. What would potentially increase the complexity of user experience is introducing more secure ways of using e-ID solutions. This report makes a few such proposals:

- introduce PIN-pad enabled ID-card readers for both PC and mobile platforms,
- introduce ID-cards with integrated PIN-pads,
- require more authentication factors by requiring signing the votes by *both* ID-card and some mobile ID solution (mobile-ID or Smart-ID in case the latter will be supported).

2.4 Need for development and cost

2.4.1 What is the complexity and estimated cost of implementing / upgrading / maintaining different solutions?

We estimate that the approximate human resources required for developing the mobile voting application for iOS and Android taken together is in the order of magnitude of 9000-11000 person-hours. The detailed estimates concerning mobile voting application development are given in Appendix C.

However, the report also makes recommendations for a number of other updates on the system and protocol level (see Section 3.6 and Appendix A). The exact architectural requirements and hence the scope of the respective development efforts is too unclear at this point to give reliable estimates. Still, some of these updates are recommended even if no mobile voting will be developed. Thus the required effort needs to be estimated after the corresponding requirements will be specified.

3 General risks of m-voting

3.1 Threat actors

In order to adequately assess the risks, we first have to consider the potential threat actors, their motivation and capabilities. We note that major threat actors are probably not interested in attacking just remote Internet voting. They are likely to have wider political goals, and are willing to use whatever means of attacking are the easiest to achieve them (including manipulation of paper votes, launching media campaigns, spreading fake news, etc.). Many of these means do not have anything to do with the medium of voting. Thus, realistically speaking, we will never be able to secure elections against every possible threat. What we can aim at in case of electronic remote voting is that it is more difficult to attack than other parts of the general political and societal processes determining the outcome of elections.

We consider the main classes of threat actors to be the following.

- **Civil hacktivist seeking publicity.** Such an attacker may have limited to medium technical capability. His aim is not necessarily malicious, but he can cause unintended problems as side effects of his activities, such as spreading misinformation and panic.

- **Single candidate trying to get more votes.** Such an attacker acting alone has limited resources, and mostly also limited technical skills.

- **An organized group of people who have the same ideology, or who otherwise have the same political positions, and who field candidates for elections trying to increase the number of seats.** Such an attacker has medium level resources and technical skills. It may have significant organisational capability, enabling certain attacks to scale quite well.

- **Foreign state-level actor interested in gaining more control over the country.** Such an attacker may have significant resources and access to rare technical capabilities (like zero-day attacks against common operating systems (OS-es)).

In this report, we will try to avoid giving assessments of the form “XYZ is / is not secure”. We will rather describe the potential issues and possible mitigation mechanisms. Whether the residual risks are sufficiently low with respect to a certain threat actor is a policy decision that needs to be taken separately.

3.2 OS level risks

This report focuses on Android and iOS as they cover over 99% of the market share both in Estonia and worldwide[3].

3.2.1 Mobile OS security

It is very hard to rationally estimate security level of a (mobile) operating system or a particular version of it. There are several folk beliefs either based on common knowledge (“A newer version of OS should have less vulnerabilities”) or some sort of personal view (“iOS is more secure than Android”), but these beliefs are quite hard to quantify.

Concerning the more updated versions having less vulnerabilities we may look at published vulnerability reports[4]. However, even one critical zero-day flaw may be sufficient for a state-level attacker to implement an attack, so the number of unpatched vulnerabilities is not necessarily a good measure of security.

Claims about Android vs iOS security are even more questionable. Due to its open development model, attackers have easier time of discovering weaknesses in Android, but at the same time public disclosures also speed up patching. As a result, the potential bounties paid out for a fresh Android zero-click exploits are even higher that those of iOS[5]. This may be interpreted as an indication that Android exploits are more rare.

We used vulnerability database VulDB[6] as a source and compared known vulnerabilities for Android 10 and iOS 13. Both of these operating systems were released in September 2019. As of March 30th 2020, VulDB contained 85 Common Vulnerabilities and Exposures (CVE-s) for Android 10 and 65 for iOS 13. However, the iOS CVE-s were more severe, which is illustrated by Figure 1. When considering the ease of exploiting, 13 out of the 85 Android 10 vulnerabilities required user interaction, whereas 23 out of 65 vulnerabilities required user interaction for iOS 13. Thus, the majority of these vulnerabilities can be exploited without relying on user interaction.

The list of vulnerabilities on its own is not sufficient to estimate the security level of a recent version of an operating system. Still, the historic data for older versions can give an indication of problems. To get a better overview of the overall security level such indications have to be combined with the information related to the security architecture of the given operating system.

3.2.2 The update policy of Android and iOS

Many of the security issues of Android devices are caused by the update policies. The problem lies in the difficulty of issuing software updates due to multiple dependencies between the software producer, device manufacturers and the chip manufacturers. The original vanilla version of Android is developed by Google, but there are many device manufacturers who can ship a modified version of Android on their devices. Thus, the updates and security patches are delayed and many Android devices do not get them at all. This is illustrated by Table 1. Apple’s iOS devices

[3]The data about the market share was collected on the 31th of March 2020 and originates from Statcounter Global Stats: `https://gs.statcounter.com/os-market-share/mobile/worldwide`

[4]See e.g. `https://www.cvedetails.com/`

[5]`https://zerodium.com/program.html`

[6]`https://vuldb.com/`

[7]Common Vulnerability Scoring System, `https://nvd.nist.gov/vuln-metrics/cvss`

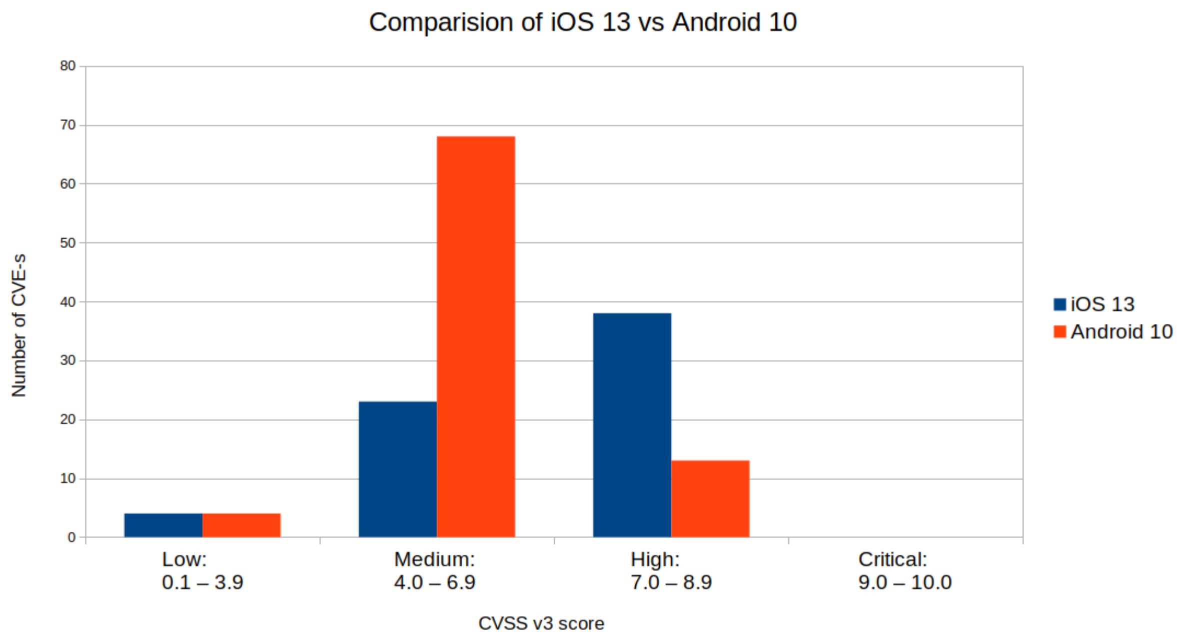


Figure 1. Number of vulnerabilities categorized according to their CVSS v3 rating[7]. The information was taken from VulDB on the 30th of March 2020.

do not have this problem as both the hardware and software are produced by the same vendor. This is illustrated by Table 2.

Before Android 8 (Oreo), in order to issue an Android update, the device manufacturer had to take the update from Google or chip manufacturer and integrate it with its own specific modifications. Only then was it possible to distribute the update to the clients. Since Android 8, the hardware specific code is separated from the rest of the Android, which makes it easier to distribute Android updates[8]. The change in the architecture makes it easier for the vendors to create updates, but it does not force the vendors to actually patch their Android version and to distribute the updates to the clients. Thus, whether the clients get the updates and patches from the vanilla version of Android depends on the vendor of the device.

In 2018, Google announced a new program called Android Enterprise Recommended[9]. It attempts to motivate the device manufacturers to follow certain requirements[10]; if they do, they get a special label. E.g. complying vendors are required to provide regular security patches that are delivered to the devices within 90 days, and they have to support patching for at least three years starting from the launch of the device. However, there are still only a limited number of devices that are labelled to follow the Android Enterprise Recommended program. As of March 2020, only 148 Android devices have this label of out the 7844 listed[11].

3.2.3 Other topics

Third party applications. By default, iOS devices are only able to get applications from Apple's App Store. This restriction can be bypassed by jailbreaking the device, which is done by exploiting

[8]https://source.android.com/devices/architecture#hidl
[9]https://www.android.com/enterprise/recommended/
[10]https://www.android.com/enterprise/recommended/requirements/
[11]https://androidenterprisepartners.withgoogle.com/devices/

Table 1. The information about the worldwide distribution of Android versions comes from Google's dashboard[1], which was last updated on the 7th of May 2019. Thus, to get an updated estimate, we included Statcounter's information about the Android version distribution in Estonia in February 2020[2].

Android version	**Codename**	**Release date**	**Worldwide distribution[1] (May 2019)**	**Estonian distribution[2] (February 2020)**
2.3.3 - 2.3.7	Gingerbread	12.2010	0.3%	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	10.2011	0.3%	0.11%
4.1 - 4.3	Jelly Bean	07.2012	3.2%	0.32%
4.4	KitKat	10.2013	6.9%	1.13%
5.0 5.1	Lollipop	11.2014	3.0% 11.5%	1.1% 1.98%
6.0	Marshmallow	10.2015	16.9%	3.97%
7.0 7.1	Nougat	08.2016	11.4% 7.8%	5.98% 2.91%
8.0 8.1	Oreo	08.2017	12.9% 15.4%	11.65% 4.01%
9	Pie	08.2018	10.4%	51.05%
10	Android Q	09.2019	N/A	15.72%

[1] `https://developer.android.com/about/dashboards`
[2] `https://gs.statcounter.com/os-version-market-share/android/mobile-tablet/estonia`

an existing vulnerability in iOS in order to escalate privileges. However, jailbreaking an iOS device is non-trivial and may not be available for all iOS versions. While iOS makes it difficult to use third party applications, Android does not prevent third party applications from being installed and also makes it possible to root the device. Rooting the device breaks Android's security model and allows applications to interfere with other applications. However, root access is not required in Android to install third party application stores and to manually install applications. The latter makes it possible to download and install pirated applications. Third party application stores, which have few restrictions for uploading applications, provide a good distribution channel for malware. The same holds for web sites which share pirated applications. When installing third party software, it may be difficult to detect malicious applications. One of the reasons is that anti-virus or security software running on Android or iOS has limited ability to inspect other applications. The restrictions are set by the security features of the operating system. For example, both iOS and Android applications are isolated from each other (sandboxed) and they are not allowed to be executed with administrative privileges.

Can we restrict access from vulnerable devices? A significant percentage of devices in Estonia run old Android versions that contain severe vulnerabilities. Vulnerable device may be infected with malware, which takes control over the device. Depending on the level of access the

Table 2. The information about iOS version distribution in Estonia was collected by Statcounter and reflects the situation in February 2020[1]. The table shows that based on Statcounter's data about 93% of iOS devices in Estonia have been updated (includes devices bought with a new iOS version) during the past year and about 83% during the last 6 months.

iOS version	**Release date**	**Estonian distribution[1] (February 2020)**	
13.4	03.2020	0.66%	
13.3 - 13.3.1	12.2019 - 01.2020	76.29%	
13.2 - 13.2.3	10.2019 - 11.2019	2.51%	
13.1 - 13.1.3	09.2019 - 10.2019	3.22%	
13.0	09.2019	0.68%	93.31%
12.4 - 12.4.5	07.2019 - 01.2020	7.7%	
12.3 - 12.3.2	05.2019 - 06.2019	1.56%	
12.2	03.2019	0.69%	
12.1 - 12.1.4	10.2018 - 02.2019	1.31%	
12.0 - 12.0.1	09.2018 - 10.2018	0.55%	
11.0 - 11.4.1	09.2017 - 07.2018	1.91%	
10.0 - 10.3.4	09.2016 - 07.2019	1.48%	6.67%
9.0 - 9.3.6	09.2015 - 07.2019	0.98%	
8.0 - 8.4.1	09.2014 - 08.2015	0.21%	
7.0 - 7.1.2	09.2013 - 06.2014	0.21%	
6.0 - 6.1.6	09.2012 - 02.2014	0.02%	

[1] https://gs.statcounter.com/os-version-market-share/ios/mobile-tablet/estonia

malware has, it may be able to use the attached electronic identity (e-ID) tool by faking the mobile OS UI calls. While individual verification can provide mitigation against malware that drops and changes votes before they are submitted, it is not possible to protect the privacy of the ballot in case the mobile device is infected. Thus, the official voting application should not be offered for devices running on out-of-date operating systems that may contain vulnerabilities.

However, it is not possible to prevent voting from a device that runs an out-of-date operating system. When an official voting application is not offered, it is still possible to create an independent voting application that follows the published Application programming interface (API) to communicate with the back-end server. In Android it is easy to install and use an unofficial voting application. Even if the voting API requires the version of the operating system to be included, the unofficial voting application can report any value that gets accepted by the back-end server.

Leaks through side channels. It is common for Android applications to silently request the list of other installed applications [SKMR20]. This is usually done by the advertisement libraries by using an official Android API[12,13]. However, when such information leaks, it can endanger the effectiveness of individual verification. As the percentage of voters who verify their vote is low, the hope is that an attacker is not able to profile and identify voters who will not verify their vote.

[12] https://developer.android.com/reference/android/content/pm/PackageManager#getInstalledApplications(int)

[13] https://developer.android.com/reference/android/content/pm/PackageManager#getInstalledPackages(int)

However, it is possible that the information about the list of installed applications is regularly collected and sent out to third parties. By having access to such information, it is possible to target these voters who do not have the vote verification application. This kind of an estimate may not work when there is a longer time period between two consequent elections. However, when two elections are close to each other, it is quite likely that the voters who verified their votes in the first election will keep the verification application installed for the next one as well.

Weaknesses in Android application signatures. Android applications have to be signed before they can be distributed through Google Play Store. There are multiple ways how the application files can be signed, but in order to provide backward compatibility, older and insecure signature schemes are still supported. Android has three signing schemes for signing Android application package (APK) files. The initial version, also known as Java ARchive (JAR) signing, has to be used for Android versions older than 7. This signature scheme contains known vulnerabilities and according to Google's own documentation offers a sizeable attack surface[14].

Another issue lies within the certificates and cryptographic keys which were used to sign the APK files. Android supports applications which are signed with 1024 bit RSA keys.[15] In order to allow updates throughout the life-cycle of the application, it is recommended that the corresponding certificate should be valid for at least 25 years. According to Google's documentation, the signing certificate has to be valid at least until 22 October 2033 in order to get an application accepted to Play Store[16]. Thus, there are many applications, which still use 1024 bit RSA keys for signing updates. We did not find a recent overview of the percentage of applications that are signed with 1024 bit keys, but each application's APK file can be manually checked as the information is present in the meta-inf folder. Researchers estimated already in 2003 that a state level actor should have the resources to brute force 1024 bit RSA keys [ST03]. In December 2019, a team of researchers announced successful factoring of a 795-bit number[17].

Cybernetica's Cryptographic Algorithms Lifecycle Report from 2016 [BHL$^+$] gives an estimate that a 1 million dollar investment would be needed to factor one 1024 bit RSA modulus. Thus, in theory it would be possible to brute force the update key of a selected application to deliver a malicious update to end users. While such an attack may be possible, it is probably easier for a state level actor to run a targeted exploit against the application developer to directly access the update key.

3.3 e-ID level risks

In Estonia, there are currently three main electronic identity solutions.

- **ID-card**, first launched in 2002, was historically the first one and is still in wide use. The latest generation of ID-cards also possesses Near Field Communication (NFC) functionality which provides an option of using it in the context of m-voting as well. For digital-only activities, it is also possible to apply for a digi-ID-card. In this report, we will not make a distinction between ID-card and digi-ID as they are equivalent from the remote electronic point of view.

[14]`https://source.android.com/security/apksigning#v1`

[15]RSA is a popular public key encryption and signature algorithm named by the original authors Ronald Rivest, Adi Shamir and Leonard Adleman [RSA78].

[16]`https://developer.android.com/studio/publish/app-signing#considerations`

[17]`https://techxplore.com/news/2019-12-encryption-keys.html`

- **Mobile-ID (mID)**, first launched in 2007, relies on the mobile phone Subscriber Identity Module (SIM) card as the key storage and cryptographic coprocessor.
- **Smart-ID (sID)**, first launched in 2016, is a software-only solution making use of a specific cryptographic scheme [BKLO17] where the signature key is split between the mobile device and server.

Right now, only ID-card and mID are used for i-voting, but it may happen that we need/want to also support sID in near future. To the best of knowledge of the authors of this report, at the time of this writing (spring 2020), the decision whether to support sID for voting or not is still open.

There are four main threats scenarios of unauthorised use of voter's e-ID:

- physical access,
- use of e-ID tool in untrusted terminal,
- compromise of e-ID environment,
- compromise of service provider.

Out of these, we consider the compromise of the user's e-ID environment to be the most serious one in the context of m-voting due to mobile OS level risks (see Section 3.2). A possible attack vector includes gaining control over the UI elements of the mobile OS. Both mID and sID use OS input-output mechanisms to display confirmation codes, enter personal identification numbers (PINs), etc. If an attacker is able to monitor PIN entry of some legitimate session, he will later be able to emulate tap events of the smart device screen to enter the same PINs in the session of his choosing.

The most serious implication of this threat is an attacker submitting a vote using a compromised e-ID environment without the voter noticing. This is a problem both in the scenarios when the attacker changes the originally submitted vote by re-voting, and also when the voter did not intend to vote at all (which is her legal right in Estonia).

As a possible mitigation measure, a notification side channel, say an e-mail to the official @eesti.ee address, can be introduced. This measure has actually been recommended before, and the main reason why it has not been implemented this far is the fear of making coercion attacks (e.g. vote buying/selling) easier. Indeed, this would be a substantial change into the current protocol. Thus, before taking a decision on whether to introduce such a measure or not, a wider analysis including also legal aspects should be conducted.

However, from the technical point of view we make the following observations.

- Even if the coercer observes a voter during the voting session and demands to see her mailbox during this session, the voter can still re-vote later.
- We assume that it is hard for the coercer to maintain physical access at many victims at the same time (most importantly, during the last minutes of the voting period). However, it is possible to demand virtual presence, say, in the form of e-mailbox passwords. If this is the case, the coercee can temporarily redirect her @eesti.ee email to a different address (even mailinator.com if she wishes). Note that @eesti.ee provides both a redirection service and a mailbox. In order to deduce the coercion risk, i-vote notification should not be saved in the mailbox.
- If the coercer is trying to get a control of all the digital channels of a voter, there must be sufficient evidence of this attempt so that the voter can turn to the law enforcement.

However, the main rationale behind making use of the @eesti.ee redirection service is to lower the coercer's incentive to control the voter's main mailbox, since this gives the coercer no guarantee of detecting a revote.

- If the coercer is willing to go as far as ceasing all the e-ID means from the voter in an attempt of blocking her option of logging onto the @eesti.ee redirection service, he can use the same approach to block the voter's revoting ability already with the present system. However, the voter is still able to cast a paper ballot in case she has access to a passport, driver's licence or any other valid ID. Thus, from this point of view, introducing the notification feedback channel does not open significant new attack vectors.

Historically, not too many Estonian citizens have used the @eesti.ee email redirection service, but in recent years this trend has changed and currently about 30% of citizens have their @eesti.ee forwarding set.[18].

Note that attacks against e-ID environment also apply to the regular i-voting even in the case no m-voting will be introduced. An attacker who is able to (remotely) take over the victim's mobile device so that he can perform authentication and digital signature operations without the voter's knowledge, can submit votes on her behalf. For that, the attacker would need to implement his own voting client. This is feasible as the protocol description is public, even though not always sufficiently detailed [KFW20].

We can argue that the threat of an attacker compromising the user's OS and e-ID environment, and as a result, submitting the votes on her behalf, is not a new problem. Indeed, in case the personal computer (PC) OS has been injected with sufficiently powerful malware, the attacker might be able to use the ID-card to (re)vote without the voter noticing this.

There are a few aspects of user behaviour that contribute to this problem.

- General low level of digital hygiene, e.g. installing software from untrustworthy locations, carelessly opening email attachments, failure to keep the OS updated, etc. Such failures are often required as presumptions for attackers to launch malware-related attacks. Raising digital hygiene awareness is one key measure in raising the security level of every kind of digital services, including i- and m-voting.
- Usage scenarios where ID-card is left attached to the working terminal for extended periods of time, e.g. as a login token. Even though short periods of legitimate ID-card usage might already be sufficient to implement an attack, the login token scenario has more problems. Namely, it is typically implemented at an OS level by leaving the card's authentication environment open. As a result, applications (including malicious ones) do not need to have access to PIN1 in order to perform authentication.

In general, one of the core problems seems to be that e-ID tokens (be it an ID-card or an mID SIM) are getting too intimately connected to the computing platforms and OSes. On one hand, this connection is convenient for the users, but at the same time it increases the attack surfaces and time windows. Whether the corresponding risk level still remains acceptable depends on the application scenario and threat actor we consider.

In case of electronic voting (both PC and mobile platform based), the integrity risks become significant when the attacks start to scale easily. We estimate that out of the threat actors listed

[18]`https://www.ria.ee/et/uudised/muutus-eestiee-postkasti-teavitus.html`

in Section 3.1, high-resource state level attackers have the capacity to attack mobile platforms in a sufficiently scalable manner.

To counter this threat, the following mitigation measures can be applied.

- Establish an informative feedback channel notifying a voter when a vote has been cast on her behalf.
- Promote individual verification. This measure acts both as a personal safeguard against a potentially vulnerable voting platform, and as a system-wide detection mechanism of scaling attacks. The higher a share of voters who verify their votes, the higher also the probability of detecting vote manipulations caused by, say, a malware attack. Another reason to promote individual verification is making the set of verifying voters less distinguishable from the general population of i-voters, making it harder for the potential malware to profile the people who do not verify their votes.
- Consider e-ID solutions where authentication factors are more separated. For example, new ID-cards have NFC interface which allows them to be used with mobile devices. One benefit of this would be decreasing the time window during which the e-ID device is exposed to the mobile device (unlike mID SIM card that is constantly attached). However, a very determined and resourceful attacker could still be able to create malware that could use even a short time window to cast a (re)vote on behalf of the user. To completely rule this option out, even more capable ID-cards are required, e.g. featuring integrated PIN-pads and also screens for trusted preview of verification codes. Smart cards with such capabilities are already marketed as payment cards[19]. Hopefully it is a matter of a few upcoming years when vendors are starting to offer them also as a basis for the next generation of ID-cards. Such requirements should be considered when procuring for the next generation of the ID-cards.
- In case the regular PC-i-voting, card readers with PIN pads should be used.
- Another way to make it harder for an attacker to submit votes on behalf of a person after having taken over her environment, is to require more authentication factors to be present, e.g. to demand vote signing by both ID-card and a mobile e-ID (mID or sID). This would somewhat harm usability, but this may be worth the trade-off in terms of increased security level.

3.4 Risks related to browsers and browser based voting

One simple solution to address mobile application development and deployment is not to have a standalone application at all, but rather rely on (mobile) browsers and JavaScript. As a matter of fact, the first voting events in Estonia made use of a browser-based interface (more precisely, ActiveX component working within Internet Explorer). However, this approach was abandoned due to browsers of mid-2000s having numerous issues.

In many respects, mobile platforms are today where PC platforms were 10 years ago. Browser vendors have invested a lot into improving their products security, but there are still a number of issues remaining. The amount and variety of these issues does not allow us to recommend browser-based voting for Estonia at the time of this writing (early 2020).

[19]https://www.gemalto.com/financial/cards/payments/display-interface

3.4.1 Ensuring authenticity of the webpage and voting application

It is more complicated on a mobile platform to become convinced in authenticity of the webpage than it is to do it on a PC platform. The authenticity is checked with the help of a certificate which is issued and digitally signed by a trusted third party also known as a certificate authority. However, viewing the site's full certificate is not an easily accessible option on mobile platforms and may not be available at all. We tested multiple browsers running on Android and iOS devices, and the user experience varies significantly across platforms and browsers. For example, in iOS we tested Google Chrome, Firefox, Safari and Brave browser, and out of these only Google Chrome allowed the user to view information about the certificate. Brief testing showed that Google Chrome for Android allows to view the certificate information similarly to the desktop version of the browser. Android's Samsung browser also allows to view the full certificate. However, Firefox for Android only displays the issuer of the certificate.

Another issue is related to checking the integrity of the browser based voting application. Currently, integrity of the desktop based voting application is guaranteed by digitally signing it under Windows and macOS, and by comparing SHA-256[20] checksum under Linux (see Section 4.1.7). However, for a JavaScript-based voting application, verifying integrity is non-trivial and may not work on all browsers. In that case the voter can not check whether the application that arrives to the voter's browser matches the one distributed by the Estonian State Electoral Office (ESEO). This issue is described in more detail in Section 4.4.3.7.

3.4.2 Man-in-the-middle attacks

It is quite common for corporations to use middleboxes that monitor Transport Layer Security (TLS) traffic. Similar behaviour can be observed in antivirus software that monitors web traffic in order to protect the user from malicious web content. Usually the middleboxes and antivirus software enable this functionality by installing an additional root certificate to the trust store of the end user's operating system or browser. Thus, in essence the monitoring is done via a local man-in-the-middle (MITM) attack, which affects the security guarantees provided by Hypertext Transfer Protocol Secure (HTTPS) [DMS+17]. The voter is also able to set up a local proxy that intercepts the traffic. This can be easily achieved with a MITM based testing tool like Burp Suite[21]. Cloudflare has built an online tool to monitor the usage of HTTPS interception and it shows that such practice is widespread[22,23]. Now, the question arises whether the man-in-the-middle attacks work with the voting application and how it would affect the voting system.

In case of a standalone voting application, the TLS certificate of the voting server can be pinned. This means that the voting application is preconfigured to accept only a specific set of server certificates or root certificates. Thus, a local man-in-the-middle attack would not work by just adding a new root certificate to the trust store. In case of a pinned certificate, the application itself would have to be modified to use a certificate that allows to execute a man-in-the-middle attack.

Mainstream browsers used to support HTTP public key pinning, but this approach is now depre-

[20]SHA-256 is a member of the Secure Hash Algorithm 2 family of hash functions. The family also includes functions with other output lengths, like SHA-384.

[21]`https://portswigger.net/burp`

[22]`https://malcolm.cloudflare.com/`

[23]`https://blog.cloudflare.com/monsters-in-the-middleboxes/`

cated[24,25]. One of the reasons for dropping the support was the difficulty of correctly implementing and using pinning. As a result, web sites could cause self-inflicted denial of service for their users. Therefore, in the case of a browser based voting application, it would not be possible to pin certificates. It is important to note that Certificate Transparency does not offer a solution as it does not have to apply to locally issued certificates as it is in the case of Google Chrome[26].

Thus, in case of a browser based voting application, all of the previously mentioned parties could run a local man-in-the-middle attack. This would give them the option to monitor and change the messages that are exchanged inside the TLS session. Therefore, when using a browser based voting application, it should be assumed that the messages sent over the TLS channel can be intercepted and the whole API is revealed. Thus, the API that is currently used should be re-evaluated in order to consider the threat of traffic interception. For example, the list of candidates queried by the voting application is not digitally signed. Thus, with a browser based voting application, it would be easy for the voter to intercept the response of the query and change the candidate list in order to cast an invalid vote. The same threat exists also for other cases of TLS interception.

3.4.3 URL bar

Checking the certificate is not a common practice as users rather expect to see a familiar Uniform Resource Locator (URL) if the service is well known. However, browsers on smartphones are often optimized to increase the available screen area and therefore hide the address bar. Testing showed that smartphone browsers do not have a common behaviour in this respect. On iOS, some browsers hide the URL bar while the user is scrolling the page, some browsers hide the URL when the device is in landscape mode, and some browsers like Google Chrome always display at least part of the URL. On Android, we tested Google Chrome, Firefox and Samsung browsers, and all of them behaved in a similar manner by hiding the URL bar while scrolling down the page.

Such behaviour can be exploited by running cleverly designed phishing campaigns which are targeted against specific browsers (which is feasible as the user agent specification is visible to the server). For example, a voter could be tricked into clicking a link which directs to a fake voting web site that visually looks exactly like the official voting web site with the only difference being in the URL. However, when a similar URL is registered by an attacker, the voter may not notice it, especially, if the voter has to scroll the page, so that the URL bar disappears. An attacker who successfully tricks voters into using a fake voting web site can see how these voters vote and thereby violate the ballot privacy. In a worse case, the attacker can change the vote that is going to be signed and return an invalid verification Quick Response (QR) code to the voter, which makes it impossible for the voter to verify the given vote. The attack against vote integrity could be executed dynamically depending on the probability estimate of the victim verifying the vote.

[24]https://www.chromestatus.com/feature/5903385005916160

[25]https://www.fxsitecompat.dev/en-CA/docs/2019/http-public-key-pinning-is-no-longer-supported/

[26]https://groups.google.com/a/chromium.org/forum/#!msg/ct-policy/wHILiYf31DE/iMFmpMEkAQAJ

3.4.4 Browser extensions

Browser's behaviour can be somewhat changed by installing extensions, which are small pieces of software written by third parties. It is very hard to control all kinds of (potentially malicious) extensions that the user may have installed. The current permission model for both Firefox and Google Chrome extensions allows them to ask for far-reaching access. The user has to either agree with the requested permissions while installing the extension, or retract from installing it altogether. There is no middle way which would allow fine-grained control over the permissions the extension has. A common permission asked by Google Chrome extensions is to "Read and change all your data on the websites you visit". Similarly, Firefox extensions can ask the permission to "Access your data for all websites" and with other permissions it is also possible to change the content on selected pages.

Therefore, it is not straightforward to protect integrity and privacy of the ballot while a browser has been augmented with extensions. Before casting a vote, the voter would have to make sure that browser extensions would not have access to the content on the vote casting web site, but this is clearly infeasible for an average user.

In addition, it is not common for smartphone browsers to have the option to add extensions. For example, in iOS neither Firefox nor Google Chrome have extensions, but in Android, Firefox has support for extensions, while Google Chrome does not. Thus, an extension-based voting application would not be available for a significant fraction of smartphone users.

3.4.5 Exotic browsers

It is common for the manufacturers of smart devices to bundle their own browsers with the devices. For Android devices, there are tens of browsers which are only used by a small fraction of users. The question arises which browsers should be supported and tested to work correctly with a web site based voting application. It is also not well known how secure the less common browsers are. Therefore, creating a list of supported browsers and reviewing their security mechanisms may be necessary in case it is decided to implement a browser based voting application.

Note also that browser detection on the server side is not 100% reliable as a malicious extension might try to lie about the browser version. This way it is possible to masquerade an exotic and vulnerable browser as a more secure one, while still making use of the vulnerabilities to manipulate the votes.

3.4.6 Built in phishing and malware protection

Google Chrome, Firefox, Edge and Safari monitor which web sites the user is visiting to detect phishing attacks. They also attempt to detect malware from downloads. Although some of the filtering is done in the local machine in a private manner, there are still pieces of information that are sent out to be analysed by third parties.

Google Chrome[27], Firefox[28] and Safari[29] rely on Google's Safe Browsing service to identify phishing sites and malware. Visited URL-s are checked locally against a list of 32-bit hash pre-

[27]`https://safebrowsing.google.com/`

[28]`https://support.mozilla.org/en-US/kb/how-does-phishing-and-malware-protection-work`

[29]In the case of Safari for Chinese users, Tencent Safe Browsing may be used. `https://support.apple.com/guide/safari/security-ibrw1074/13.0/mac/10.15`

fixes. In case a match is found, the prefix is sent back to Google, who replies with a list of all URL-s that match the corresponding prefix. Thus, the visited URL can be matched locally. The information about which specific URL the user queried is protected by a k-anonymity mechanism as there are probably multiple other URL-s that share the same prefix. However, it may be possible to identify users by using additional information available to Google, see Matthew Green's essay [Gre19].

Microsoft's Edge browser uses Microsoft SmartScreen[30] to detect phishing attacks and malware. It uses local filtering for known phishing sites and top traffic sites. In case the visited URL does not match either, the URL is sent to be analysed by the SmartScreen service. This is described in SmartScreen's documentation[31] with the following sentence: "Microsoft Edge passes the URL, relevant info about the site, an identifier unique to your device, and general location to the SmartScreen service to determine the safety of the site". However, it is not clear which web sites are classified as top traffic sites and thus the end-user does not know when the browsing info is sent to the SmartScreen service.

A technical report was written in the beginning of 2020 focusing on the information that browsers send back to the vendor or third parties [Lei20]. The report focused on the following six browsers: Google Chrome, Mozilla Firefox, Safari, Brave Browser, Microsoft Edge, Yandex Browser. The comparison showed that Brave Browser did the best when considering user's privacy, while Microsoft Edge and Yandex Browser were in the opposite end of the scale. Chrome, Firefox and Safari had similar results, e.g., they all sent information to the respective vendors regarding the information that was typed to the URL bar.

In the context of voting, it may be problematic if third parties know the exact time when a certain user is visiting the vote casting web site as such information may enable some types of attacks. Such privacy issues are not only restricted to browsers as similar metadata is also visible to internet service providers and Domain Name System (DNS) providers. Theoretically, this kind of information could be used to enumerate voters who do not i-vote. If such information leaks, an attacker could attempt to infect the devices of voters who have not i-voted in order to prevent anomalies in the log analysis of re-votes.

The time of casting a vote may also give hints about the (re-)voting. For example, when there is a political scandal during the i-voting period and many i-voters rush to change their votes, a third party with access to the usage patterns to the vote casting web site may be able to draw some conclusions about voters' political preferences.

It is also not known how easily a web site can be classified to be malicious. As there are methods for end-users to report web pages, the question arises whether it is possible to create a denial of service attack by starting to massively report a specific web page. Thus, the privacy and usability effects from protective filters and browser telemetry should be further studied.

3.4.7 Security risks related to the web applications

Web-based voting application may introduce new risks depending on how the web site is built. Most likely the web site of the voting application would use JavaScript. The voting web site would face the same risks as other web sites. According the OWASP Top 10[32], these risks include injection, cross-site scripting, and using components with known vulnerabilities.

[30]https://docs.microsoft.com/en-us/microsoft-edge/privacy-whitepaper#smartscreen
[31]https://docs.microsoft.com/en-us/microsoft-edge/privacy-whitepaper#smartscreen
[32]Open Web Application Security Project, https://owasp.org/www-project-top-ten/

It is important to prevent cross-site scripting vulnerabilities as they could lead to session hijacking, website defacement and information leakage. It is important to understand that by relying on third party frameworks the vulnerabilities in these frameworks also affect the functionalities of a web based voting application.

3.4.8 Vulnerabilities in browsers

The architecture of mainstream browsers has been improved over the years to lower the attack surface, but it does not mean that there are no vulnerabilities to exploit. CVE databases like `cvedetails.com` reveal that remote code execution vulnerabilities are still regularly found in all mainstream browsers. These could lead to remote code execution either in the context of the browser or to getting operating system level access.

For example, while writing this report Google published an update for Google Chrome to patch the vulnerability CVE-2020-6418, which was already actively being exploited in the wild[33].

While some browser vulnerabilities can be used to execute arbitrary code on the device, the majority only affects the browser. Thus, the attack surface of a browser based voting application is larger than the attack surface of a stand-alone application. For the context of the stand-alone application, the issue is in the vulnerabilities that give remote access to voter's device. However, in the case of a browser-based voting application, the list of threats has to be extended to include browser vulnerabilities.

3.4.9 Phishing web sites

A significant threat related to web site based voting lies in phishing as people are not used to carefully check the URL and certificate of a web site they visit. It would be trivial to copy the visual appearance of the web site of the official voting application. Next question is whether URLs similar to the official voting web site could be set up to fool the voter into believing that the web site is official. It is difficult to prevent the registration of such domains as there may be many similar domains that are based on other top level domains. Also, the attacker may make use of the browsers' functionality to hide the URL while scrolling (see Section 3.4.3).

In case a voter becomes a victim of phishing, it may result in both the violation of privacy and integrity of the ballot. First, when a voter picks a candidate on a phishing web site, the attacker can see who the voter voted for and such information could also be published later. Second, when a voter votes on a phishing web site, it may not forward the vote to the vote collection server. Third, the phishing web site could execute malicious JavaScript that would not follow voter's choice when voter picks who to vote for. The malicious phishing site could replace the candidate number that is sent to be signed by the voter.

The phishing web site is controlled by the attacker and therefore the attacker could generate a bogus verification QR-code so that it could not be read by the verification application. As the attacker is able to authenticate the voter, the attacker can make the decision of whether to drop, modify or forward the vote based on the identity of the voter.

A proof-of-concept phishing web site was demonstrated in 2016 during the Republican Presidential Caucus in Utah [KKW18]. During the Caucus, an imposter site was set up at ivotingcenter.gop mainly with the aim of informing voters of the dangers related to i-voting; see also Section 4.4.2.

[33] `https://www.us-cert.gov/ncas/current-activity/2020/02/21/google-releases-security-updates-chrome`

3.5 Verification

One of the problems that arises when Estonia would introduce voting on mobile devices is losing mobile devices as an independent verification platform. Independence of the voting and verification platforms is important for the verification to fulfil its primary goal of detecting whether a vote has been manipulated by a potentially malicious (e.g. malware-infected) voting device [HW14b].

In principle, there are two possible solutions to this problem.

1. Retain verification from the mobile device as the only option, hoping that voters will be using different devices for voting and verification.
2. Allow verifying mobile votes from a PC-based verification app.

The second option assumes adjusting the verification protocol. The required adjustment together with a detailed risk analysis is provided in Appendix A. The main conclusion of the analysis is that security level of PC-based verification solution is largely comparable to the present one. However, the mobile voting device would become aware of whether the voter decided to verify her vote or not. This may give some advantage to malware that tries to delay the vote in an attempt to figure out whether the voter tries to verity or not.

The first option can, in principle, use the verification app as it is provided today. The QR-code can be displayed on the screen of one device and captured with another (which can be borrowed from, say, a family member). If this will be the main usage scenario of mobile voting and verification, we can still assume independence of devices to a certain extent.

However, risk lies within the voters who will try to trick the system by verifying the QR-code with the same device that generated it during voting. There are a number of ways this can be achieved, e.g. using physical mirrors or apps that may help capturing screen images to the camera input. Hence the answer to the question whether mobile verification fulfils its goal in case of mobile voting, really depends on the user behaviour.

Most of the users can be expected to behave in the simplest possible way, and if this way is finding a second mobile device for verification instead of physical mirrors or esoteric apps, mobile verification could still be used together with mobile voting as well. To confirm this hypothesis, user studies need to be conducted.

Additionally, it is not known how many voters have access to a second trusted mobile device, which they can use to verify their vote. It is also not known how many voters are reluctant to use someone else's mobile device for verification due to the risk of coercion or risk to the privacy of their vote. For these reasons, the number of voters who verify their votes may decrease when continuing to use only the current verification mechanism. Unfortunately, such estimates can not be easily validated.

However, the biggest problem with verification we currently have is that a relatively low fraction of voters actually verify their votes. Independent of whether mobile votes will be verified on mobile or desktop devices, in order to really gain the effect of verifying, the share of verifiers should increase. The primary way how this can be achieved is by a respective public campaign. It would help raising public awareness if the verification option would be more clearly promoted by the voting application.

In an attempt to raise the share of verifying voters, it is also possible to deploy mobile and PC-based verification solutions in parallel.

3.6 Mitigation measures

In general, possible mitigation mechanisms can be divided into prevention, detection and recovery measures. All three categories are important. The detection measures allow to identify interference and to react by executing the recovery measures. However, such measures on their own do not prevent attacks. Thus, without applying sufficient preventive measures a resourceful attacker can effectively stop i-voting. Therefore, the preventive measures have to be good enough to prevent a large scale attack that could affect the outcome of the elections.

Table 3 presents a list of mitigation measures and classifies them according to their aim.

Table 3. Classification of mitigation measures based on their effect to i-voting.

	Prevention	**Detection**	**Recovery**
Increase awareness	●	●	
Promote verification	◐[1]	●	
Introduce a feedback channel	◐[2]	●	
Do not support legacy mobile operating systems	◐[3]		
Other client side restrictions	◐[4]		
Add freshness notification to vote verification	◐[1]	●	
Prevent ID-card from being in the reader when not used	●		
Promote the usage of PIN-pad based ID-card readers	●		
Require both ID-card signature and mobile-ID/Smart-ID signature	●		
Analyse i-voting logs		●[5]	
Possibility to re-vote on paper after i-voting has ended			●[6]
Postpone i-voting			●[7]
Allow to re-vote	●[8]		●[6]
Fall back to paper voting after a large scale attack			●[7]

● = measure is efficient   ◐ = measure is partially efficient

[1] In case individual verification is widespread, the motivation for some types of attacks falls.
[2] A feedback channel may stop an attacker who wants to invisibly interfere.
[3] An attacker is able to run his own voting client on legacy operating systems.
[4] Client side restrictions can be bypassed if adversary has full control over the voting device.
[5] This is a system-wide measure to detect anomalies.
[6] This is an individual recovery measure for voters who were coerced.
[7] This is a system-wide measure to recover from a malfunction or from an attack.
[8] The option of the voter re-voting limits the coercer's capability to ensure that coercion was successful.

In the rest of this Section, we are going to elaborate further on the proposed mitigation measures.

Increase awareness. It is important to raise the general awareness level of digital hygiene. For example, it would have a significant positive impact if many citizens would regularly update their software to patch existing vulnerabilities. While such action is necessary, it won't be possible to educate every voter. In addition, state level actors have access to zero day vulnerabilities and are able to bypass antivirus software.

Promote verification. Currently, the rate of verifiers is about 4-5%[34], but the more there are, the more attacks we are able to detect [HW14b]. In case individual verification would be more widespread, it would also act more as a preventive measure. When an attacker wants to change the election outcome, the attack should be executed silently. Thus, widespread individual verification can reveal if votes get dropped or changed by malware and thereby deter such attacks from attackers who have to prevent detection. However, the current vote verification system is not able to detect malware that casts a re-vote which overwrites voter's original choice. The following mitigation measures also address the issue of detecting such malware and preventing it from succeeding.

Introduce a feedback channel A feedback channel can be used to notify the voters about their act of voting. This measure would be useful in multiple scenarios. For example, the voter would be able to detect re-voting malware or malware that drops votes based on a prediction on whether the voter is going to verify the vote. In the latter case, the voter could detect vote dropping attacks even without using the verification system, which would make it difficult for an attacker to avoid detection. This is relevant when considering the proposed verification scheme for m-voting discussed in Appendix A. Similarly to verification, the feedback channel is mainly a measure to detect interference. However, as a side effect, it can also deter an attacker in the fear that the attack to be revealed. Again, similar to individual verification, we hope that this deterrence will also act as an efficient prevention measure. This idea is discussed in more detail in Section 3.3.

Do not support legacy mobile operating systems. It is possible to try to restrict the official voting client so that it would run only on up-to-date operating systems. However, the effectiveness of this measure depends on the capabilities and attack goals of the attacker.

The problem is that a really determined and resourceful attacker can develop a voting client also for an old and vulnerable platform where he can potentially run it without the user knowledge. This is doable as the voting protocol is open even though not always documented the best way. If an e-ID utility is also accessible without the user knowledge then the attacker can mount an attack against vote integrity. Efficient measures against this threat include increasing the general level of digital hygiene and establishing a feedback channel as described above.

However, not supporting legacy OSes by the official voting client has a positive effect on vote privacy. If the voter only has access to the voting client on an up-to-date OS, it will be hard for an attacker to develop and deploy malware that would attempt to, say, read the user's screen during the voting session.

Other client side restrictions. Obfuscation and malware detection measures only work against some attackers. State level actors and researchers have the capability to reverse engineer the voting application to detect which measures are used. Once the measures are known, they can be bypassed, assuming that the attacker has root access to the device. An example of bypassing obfuscation and malware detection measures is given in Section 4.3.1.

[34]`https://www.valimised.ee/en/archive/statistics-about-internet-voting-estonia`

Add freshness notification to vote verification. Estonian i-voting system gives voters the option to use individual verification. This means that voters can check whether their vote reached the voting system. The existing implementation allows to verify the vote during a limited time window, which is configurable but has historically been set between half an hour and an hour. Thus, after casting a vote the voter has up to an hour to take a smartphone with a verification application and check whether her ballot reached the voting system. It is important to note that the voter is not able to check whether the ballot that reached the voting system will be counted in the tally as such an ability would also make vote selling very easy.

The current verification system is optimised for being coercion resistant and thus verification does not reveal if a re-vote has been cast. Now, imagine what could happen when a voting device would be infected and controlled by malware. It is known that malware can use voter's e-ID if it is directly connected with the infected device. In the case of ID-card this is trivial if a regular smart card reader is used. Thus, malware could wait until the voter casts her vote, then copy the PIN codes and re-vote right after the voter has successfully voted (although the re-vote could also be cast later when malware detects that the voter has connected the ID-card to the card reader another time). The voter is not quick enough to remove the ID-card from the card reader to prevent malware from accessing it (which can be done in a fraction of a second). Instant re-voting by the malware can be prevented by using a PIN-pad based card reader that has a PIN-firewall, or a PIN-pad enabled card. The downside lies in the fact that these kind of card readers are rarely used by voters, and PIN-pad enabled ID-cards are not even on the market yet. PIN-firewalled card readers are no longer locally available in the shops in Estonia and thus voters are not able to buy them even if they are aware of the risks of using regular smart card readers.

However, the existing individual vote verification can be easily extended so that it would also provide a partial integrity check. The verification system could notify the voter during verification whether the given vote was overwritten or not. If the voter performs this verification after she has removed the ID-card from the possibly malicious device and does not use it any longer during the i-voting period, the voter can be sure that malware did not abuse the access to the ID-card. The verification time window is short and is probably not suitable for re-voting in case the initial vote was given under coercion. The coerced voter can re-vote later after the verification time-window has passed as then the coercer can not check whether the coerced vote was overwritten. In case coercion would happen during the last hour of the i-voting period, the coerced voter could go to the polling station to cast a paper vote. That is the main reason why currently i-voting ends a few hours before paper voting during the pre-election period. Starting from 2021, voters will also have the option of voting during election Sunday, i.e., casting a paper vote that overwrites the i-vote.

Until ID-card's NFC interface is not used for other activities on a mobile device (nor over a regular smart card reader), the voter can be sure that malware does not have access to the ID-card. This measure only works when the voter is careful and when malware can not rely on other e-ID solutions like mobile-ID or Smart-ID to cast a (re)vote.

The problem of malware having constant access to a mobile e-ID is similar to the situation where the ID-card is constantly kept attached to the PC using a reader without an integrated PIN-pad. A such, this problem needs mitigation measures anyway, with possible solutions including a feedback channel for the fact of voting and limiting the options of casting a vote using just one e-ID solution that can potentially be taken over by an attacker.

Prevent ID-card from being in the reader when not used. Discourage the scenarios where it is required to leave the ID-card in the reader for extended periods of time, and practices where the card's authentication environment is left open on the OS level. In case voter's device is infected with malware and the voter is not using a PIN-pad based ID-card reader, the malware could re-vote and thus overwrite voter's initial choice.

Promote the usage of PIN-pad based ID-card readers. Target e-ID solutions with better separated authentication factors. E.g. on the regular PC platforms, make use of PIN-pad-equipped ID-card readers. Without a PIN-pad based card reader, malware could issue a re-vote right after the voter has voted as the ID-card is still in the reader. Currently individual verification would not detect such an attack. For usage with mobile devices as terminals, NFC cards with integrated displays and PIN-pads should be utilised. In case individual verification would provide some integrity guarantees as described above, the NFC based vote signing could be a step forward. While the majority of smart phone users rely on Smart-ID and mobile-ID for daily interactions, the NFC based vote casting could offer a way to prevent malware from re-voting.

Require both ID-card signature and mobile-ID/Smart-ID signature. The idea is to force the vote casting to depend on two independent devices. The vote should be accepted only if the timestamps of both signatures are within a certain time-limit. This measure would lower usability of electronic voting, but it may be an acceptable trade-off with increased resistance against malware attacks.

Analyse i-voting logs. Log analysis can reveal anomalies which can be used to identify attacks. For example, it is possible to monitor when and how many times people vote, which e-ID tools and OSes they use, whether and when they verify their votes, etc. [HPW15].

Possibility to re-vote on paper after i-voting has ended. In case the voter is coerced to vote according to someone else's wishes during the last hour of the i-voting period, the coerced voter should be able to go to the polling station to cast a paper vote. This is the main reason why i-voting has to end a few hours before paper voting. Re-voting as an anti-coercion measure could be bypassed when it would not be possible to re-vote on paper as the coercer would then have to force the voter to vote in the end of the i-voting period. According to the best knowledge of the authors of the current report, i-voters will have the option to revote on paper during the election Sunday starting from the elections in 2021.

Postpone i-voting. This is a legal measure that can be executed when a large scale attack is detected.

Allow to re-vote. Re-voting is a measure designed for preventing coercion. It allows the voter to overwrite her vote by casting a new i-vote. It is also possible to go to the polling station to vote on paper. Paper vote overwrites an electronic vote and can no longer be changed.

Fall back to paper voting after a large scale attack. This is a legal measure that allows to cancel i-voting in case a large scale attack is detected that can not be mitigated with other means. That way voters can be asked to vote on paper during the election Sunday. This is also one of the reasons why i-voting period should be limited to the pre-election phase.

3.7 Further research

This report can not give a complete overview of all of the security aspects that may influence mobile voting. Thus, in the following we list the areas that may require further research.

Telemetry and crash reports. It should be studied which kind of information may be sent to the vendor of the device or to the vendor of the operating system. Do iOS and Android differ in the amount of collected telemetry? Has it been studied if and how telemetry collection affects the current PC based voting application? Do the crash reports leak information related to the voting application?

Devices that are under corporate control. It should be studied if corporately controlled devices are suitable for voting. For example, it may be impossible to install additional applications on corporately controlled devices. How do iOS, Android and PC based platforms differ in this regard? Which third party applications are commonly used to monitor employees and do these applications violate the privacy of vote casting? How do the corporate devices influence vote verification?

Bloatware. Bloatware is often found in cheap Android devices. How common is this problem? Does it affect Estonia where the market share of no-name Android vendors is small? Does bloatware pose a threat to voter's privacy?

Integrity of JavaScript application. Is there a suitable method for verifying the integrity of a JavaScript application? The solutions proposed in Section 4.4.3.7 provide only a partial solution to the issue. This is one of the problems that has to be solved in case browser based voting is considered.

User studies related to m-voting. It is currently not known how voters will react to changes brought by m-voting. For example, do the voters continue to verify their votes with the same mobile device which they used to cast the vote? Are voters able and willing to use someone else's device to verify their vote? User studies should be conducted to find out how they would adapt to the changes. It is especially important to make sure that the user interface of the voting application is easily usable. For example, the user interface should make it clear for the voters why and how they can verify the vote.

4 Technical solutions for (mobile) voting application

In this chapter we analyse potential technical solutions for mobile voting application.

For the general context, we will first describe the key architectural requirements decisions taken in 2010 for the current Estonian online voting application. Then we will proceed to briefly discuss development alternatives for a PC-based voting application followed by a treatment of different aspects that need to be considered for the development of platform-specific mobile voting apps. The final section of the current chapter is devoted to the browser-based approach to mobile voting.

In order to limit the scope of the treatment, we make two restrictions in the current chapter.

- We do not consider the e-ID security issues. These are covered in Section 3.3.
- We do not consider the individual verifiability issues. These are treated in Section 3.5 and Appendix A.

4.1 Status Quo – standalone voting application

Standalone voting application is a native application developed for the latest versions of Windows, macOS and Linux. Voting application binaries for all platforms are distributed via the official website `https://www.valimised.ee/` directly under the control of ESEO. The binaries are self-contained, statically linked and digitally signed native executables for aforementioned platforms.

When the voting application is executed, it runs as a process in the rights of the current OS user (which may be a privileged user, if the voter decides so). The process benefits from all OS related features (such as integrity verification on macOS and Windows, user-based access restrictions to resources, etc). The process can be interfered by other processes via standard means – desktop platforms do not sandbox processes by default. A process capable of e.g. debugging the voting application could potentially overwrite its memory and change the control flow. This risk has to be accepted, the voting application itself cannot ensure safety of the environment it is executed in.

The voter uses a graphical user interface (GUI) to cast a ballot and, in the end, explicitly closes the application. As a response to closing the application, OS process is terminated and all associated resources, including memory, are released.

Currently the responsibility for ensuring the platform security lies with the voter. It is advised that the operating system is kept up-to-date and an anti-virus solution is used. Such user awareness based preventive measures and their effectiveness are described in Section 3.6.

In 2010, the voting application was developed without an individual verification tool available. In 2011, a proof-of-concept malware attack to both disenfranchise a voter and to tamper with one's vote was implemented with help of AutoIT scripting tool [HLW11]. This again raised the issue of trusting the voters' computers, and for the next elections in 2013, an individual vote verification tool was introduced [HW14b]. Additionally, a virtual mouse- and keyboard event filtering together with some visual hints about overlapping screens were implemented to harden the application against similar scripting hacks. These countermeasures are in conflict with some UI accessibility tools.

In the rest of this section we will list and discuss the main aspects to consider when developing a voting application.

4.1.1 Functional requirements

Voting application implements voting for all Estonian national elections

Voting application has to support voting in Parliamentary, Municipal and European Parliamentary elections, it also has to support referendums. It is possible to have multiple contests at the same time. Related logic must be implemented in a uniform manner on all platforms.

One voting application has to support one and only one particular election

Voting application is configured specifically for one event, the same application cannot be reused in any other elections. There is a theoretical possibility to use same base application for two concurrent elections, but it would still require two distinct sets of binaries to be configured. This means that the lifespan of a voting application is at most a month and there is no need for a business logic to handle different potentially simultaneous elections within the same application. From the server perspective, there is no need for backwards compatibility with older versions which simplifies the server development and reduces legacy code.

The look-and-feel of the voting application is completely controlled by the ESEO

Voting application is a voter's tool to cast her preference as a ballot in a given election. The original intention with the voting application was to be as close to the official paper-based process as possible. This means that all parties and candidates have to be presented in the correct order, but it also means that no filtering or reordering capabilities should be available for a voter as paper voters do not have these. An invalid ballot cannot be cast with the official voting application as such an option is not provided by the election law. Some of these restrictions (filtering) have later been loosened, some (no casting of invalid ballot) still remain.

Additionally, ESEO must have complete control over the texts displayed and colours used by the voting application. Tools to provide modifications and preview to the voting application look-and-feel are developed for the ESEO.

Voting application is aware of the distributed voting service

Since 2017, vote collection service is distributed for reliability, allowing voters to vote even when some nodes of the collection server are not available. Although the uniform distribution of voting application instances between different nodes could be achieved via DNS round-robin, there exists a logic to try out multiple known end-nodes in random order to avoid nodes that are not reachable at the time of voting.

4.1.2 Development toolchain

Voting application shares codebase on multiple platforms and architectures

The voting application is made available on Windows, macOS and some Linux distributions. Approximately 93% of voters uses Windows, 6% macOS and 1% Linux-based version [HPW15]. Only sufficiently recent OS versions with supported e-ID drivers are considered. Currently multi-platform support is achieved with a single C/C++ codebase compiled into native statically linked binaries (either x86 or x86-64) on each of the platforms.

C/C++ was selected as the programming language due to good support on all required platforms, availability of compatible third party libraries and GCC compiler suite.

Third party libraries are selected to be multi-platform, operating system level issues are handled mostly in the libraries – networking in OpenSSL[35] and GUI with FLTK[36]. Only e-ID specifics are visible at the application level. With this approach, all logic and protocol related issues are handled in a uniform manner. On the other hand, it is more difficult to get hold of platform specific capabilities, mostly concerning modern user interfaces or dependencies of system-wide configuration (such as proxy servers in the network).

Voting application limits the number of external dependencies

Since the voting application has to be supported on a variety of operating system versions on any given platform, it cannot make many assumptions about the available APIs. Therefore, the binaries are statically linked, using only OS level APIs dynamically. This improves stability of the application as most aspects of the system are under application control – such as specific parameters of TLS connections. This also means that the voting application can be executed without any installation. The application is self-contained and when the OS level pre-requisites are satisfied, it works.

4.1.3 Transport layer security

Voting application implements its own TLS-stack and completely controls its parameters

Transport protocol between the voting application and the vote collection service uses JavaScript Object Notation remote procedure call (JSON-RPC) over TLS. Specific cipher-suite and TLS protocol version are configured for the application. This allows for very restricted configuration on the server side as the TLS stack is statically compiled into the voting application.

Currently only TLS 1.2 or TLS 1.3 should be used to avoid well known and exploited vulnerabilities[37].

TLS Server Name Indication (SNI) extension is necessary to support the voting protocol.

OpenSSL is used as the TLS library. Previous, ActiveX based voting application relied on the Internet Explorer API on HTTPS communication. This meant dealing with all differences of available TLS implementations in all supported Windows versions, and was unstable compared to the present version.

Voting application uses mutual authentication when possible

In case of ID-card, mutually authenticated TLS is used. In case of mobile-ID, only the server side is authenticated on the TLS level.

[35]https://openssl.org

[36]https://fltk.org

[37]https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices

Voting application uses certificate pinning

In order to avoid local trust store poisoning and MITM attacks (e.g. in case of mobile-ID), TLS certificates are pinned into the voting application. Pinning can be done both on the root CA and server certificate level. Because the voting system is distributed, several certificates can be pinned.

4.1.4 Encryption and digital signature

Voting application requires cryptographically secure randomness

In order to ensure ballot secrecy, access to cryptographically secure random numbers is required by the voting application. On current platforms this is not an issue as native voting applications have access to operating system level interfaces via the OpenSSL library.

Voting application requires encryption APIs

Voting application must implement asymmetric and randomised ElGamal encryption (currently in $\mathbb{Z}_p$) and corresponding Abstract Syntax Notation One (ASN.1) data structures to support IVXV[38] voting protocol [HMVW16]. ElGamal protocol for vote encryption is implemented directly on top of the OpenSSL big-integer implementation.

Voting application gives and verifies digital signatures

Voting application has to calculate BDOC[39] compatible digital signatures. Currently, BDOC-TS (ASiC-E LT) profile is used. Voting application can simplify the BDOC implementation – it is not necessary to support all profiles / use-cases, or have an XML library. The vote can be both composed and verified with the help of templates. Nevertheless, support for X.509[40] certificates, Online Certificate Status Protocol (OCSP)[41] responses and PKIX[42] timestamps is necessary and is currently achieved with the help of OpenSSL library.

Additionally, ZIP compression must be provided by the voting application.

4.1.5 e-ID tool support

Voting application requires ID-card APIs

The voting application uses ID-card for digitally signing the vote and authentication in TLS handshake. On Linux and macOS platforms, ID-card is used via PKCS11[43] interface. On Windows, CNG[44] (Cryptography API: Next Generation) is used.

ID-card support is a crucial element of the voting application and it depends on the service provided by DigiDoc[45] software. Continuous integration / testing of voting application and ID-card driver co-operation is important. This testing must take into account differences in ID-cards issued at different times (and by different providers).

Voting application requires mobile-ID APIs

Voting application uses mobile-ID as an alternative authentication and signing tool. Voting

[38]IVXV (Internet Voting with eXtended Verifiability) is a code-name for the i-voting infrastructure and protocol suite in use in Estonia since 2017.

[39]`http://www.evs.ee/tooted/evs-821-2014`

[40]`https://tools.ietf.org/html/rfc5280`

[41]`https://tools.ietf.org/html/rfc6960`

[42]`https://tools.ietf.org/html/rfc3161`

[43]`https://www.cryptsoft.com/pkcs11doc/`

[44]`https://docs.microsoft.com/en-us/windows/win32/seccng/cng-portal`

[45]`https://installer.id.ee/`

application does not communicate with DigiDocService / mobile-ID REpresentational State Transfer (REST) API service directly, instead it relies on the vote collector to proxy this communication. This is potentially a problem, see Section A.6.3.2 for more details.

4.1.6 Usability and accessibility

Voting application provides a graphical user interface

Voting application relies on FLTK as a GUI framework supporting all the required platforms.

FLTK abstracts itself away from the operating system level GUI APIs. For example, a multi window FLTK application on Windows only opens two operating system window handles and has its own internal handle structure to create all windows visible to the end user. FLTK does not reuse widgets available natively, but draws its own widgets using low-level drawing APIs (such as GDI[46] on Windows).

The choice of the FLTK for GUI was made based on the multi-platform support, small memory footprint and suitability for static linking.

Voting application implements accessibility interfaces

Voting application provides limited accessibility on Windows by implementing MSAA[47] interface. Tools such as NVDA[48], JAWS[49] or Windows Narrator[50] can be used to read the application screens, but the control elements are not implemented. Voting application supports keyboard navigation.

Today, MSAA API is becoming deprecated, and instead UI Automation Specification[51] on Windows is becoming a new standard. Additionally, there are now accessibility interfaces and tools on macOS. Improving accessibility is not a matter of simple upgrades in the current architecture.

4.1.7 Deployment

Voting application has a small footprint

Voting application must have small footprint to reduce the download times and it must be usable in case of low network bandwidth. Currently, the deployed application is between 1MB and 2MB. To satisfy this requirement, the following work has been done.

- Careful consideration of the third party libraries (e.g. FLTK over Qt[52]).
- Reducing the size of the third party libraries (e.g. removing drag-and-drop from FLTK, excluding all unused protocols from OpenSSL, etc.).
- Dropping large frameworks if only a small subset is needed (e.g. Cairo[53] and gradients).
- Reducing the compiled size of the statically linked binary (-oS, strip).

[46]https://docs.microsoft.com/en-us/windows/win32/gdi/windows-gdi

[47]https://docs.microsoft.com/en-us/windows/win32/winauto/microsoft-active-accessibility

[48]https://www.nvaccess.org/

[49]https://www.freedomscientific.com/products/software/jaws/

[50]https://support.microsoft.com/en-us/help/22798/windows-10-complete-guide-to-narrator

[51]https://docs.microsoft.com/en-us/windows/win32/winauto/windows-automation-api-portal

[52]https://www.qt.io

[53]https://www.cairographics.org/

- Using executable packer before distribution.

Voting application is configurable by ESEO

Voting application binaries are compiled by the vendor and delivered to the ESEO. In addition, a configuration application is deployed to the ESEO. This application is used to specify the complete configuration – URLs, certificates, texts, fonts, colours etc. – and inject this configuration into the binaries before digitally signing and distributing them.

Authenticity and integrity of a voting application can be verified

Voting application binaries configured by the ESEO are digitally signed by the vendor with a respective code signing key (for macOS and Windows). Additionally, SHA-256 checksums of the binaries are digitally signed by the election organizer and published on the `https://www.valimised.ee` website.

On Windows, the SmartScreen[54] verifies the voting application and stops it from running unless verification is successful. On macOS, similar functionality is provided by the Gatekeeper[55]. However, such phishing and malware detection mechanisms may leak information to the the service provider as described in Section 3.4.6.

Critical updates for voting application can be deployed in a timely manner

Voting application is distributed as a single executable file not requiring any installation. The voting application is distributed over `https://www.valimised.ee` website, which is under the control of ESEO.

Under the regular conditions, hot-fixing should not happen during the elections. However, the need for hot-fixing may appear unexpectedly, and the possibility to have a hot-fixing option is a critical mechanism to counter possible future issues.

For hot-fixing, the length of the deployment cycle (compile, configure, sign, deploy) is of utmost importance. In the current scenario, a hot-fix can be deployed in matter of a few hours, given that the fix exists and a decision to deploy is taken by the election organizer.

4.2 Platform specific standalone voting applications

Current platform specific standalone voting application is in many ways in a good shape. However, there are potential areas for improvement – such as accessibility of the application and general look and feel on supported platforms.

Voting application GUI is implemented with FLTK framework. This framework makes it possible to have single codebase for all required platforms, abstracting the voting application development away from platform specific GUI development. Additionally, FLTK is a very slim framework that supports well both requirements for a small footprint and protecting the voting process from external interferences.

The downside of FLTK is that it does not take advantage of platform specific GUI widgets and therefore stands out from other applications on all platforms – this is mostly an issue with macOS and Windows. Additionally, FLTK feature-set does not include support for assistive technologies / accessibility. The MSAA interface to support accessibility on Windows is a patch to the original framework that must be maintained separately.

Next, we will briefly discuss three potential ways forward with native voting applications.

[54] `https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-smartscreen/windows-defender-smartscreen-overview`

[55] `https://support.apple.com/en-us/HT202491`

4.2.1 Current application

Despite all the issues, it is possible to continue to use the current application for future elections. This means that differences in the look and feel of the underlying platform and that of the voting application have to be accepted.

This also means that a significant amount of the work has to be dedicated to development of the FLTK framework itself in order to improve the support for assistive technologies. On Windows platform this means abandoning the legacy MSAA automation and moving forward with UI Automation API. On macOS this means supporting NSAccessibility[56] API introduced in macOS 10.9.

4.2.2 Separate application for each platform

The second option would be to abandon the current single code base and to develop voting applications for Windows, Linux and macOS using native technologies. In its extreme, the Windows application would be developed in C#, macOS application in Swift and Linux application in C.

It would be more sustainable to isolate GUI related concerns from the rest of the application logic and to have platform independent abstraction of voting application GUI elements, while making platform specific implementations based on the OS level GUI frameworks, taking advantage of native widgets and accessibility. In this case the core development language would have to be well supported on all platforms, such as C, C++, or Go. Similar approach is already existing in the current application – namely the ID-card interface is the same for the application logic, but Linux and macOS implementation uses PKCS11 API, whereas Windows uses CNG API.

4.2.3 Piggy-backing with the DigiDoc software

The third option would be to integrate the voting application with the DigiDoc software. This would enable the implementation of similar GUI experience with Qt framework, deployed together with DigiDoc. It would also make it easier to maintain a good level of integration between the voting application and the e-ID drivers.

Some aspects that have to be taken into consideration with this approach are as follows.

- Some voters use mostly mobile-ID and may not have the DigiDoc software installed. For these voters the process will become more complicated.
- DigiDoc software update cycles are not related to elections. It might be desirable to have a fast deployment option at the time of an election or right before the election.
- ESEO will have less control over the look and feel of the voting application. The configuration distribution issue would have to be reanalysed as the current injection-based method would not work.

We also note that in this case the voting application becomes something that is always available. In a wider perspective, one could take advantage of this by creating a steady online voting service running between the elections as a public test-service. It would also raise the issue of server versions having to consider more voting application versions than simply the one that was deployed for that election.

[56] https://developer.apple.com/documentation/appkit/nsaccessibility

4.3 Platform specific mobile voting applications

One possible route to take is to implement voting application as a native application for both Android (86% of the market[57]) and iOS (13% of the market) platform. Those native applications would be distributed via platform specific marketplaces and would be subject to an external review before acceptance, reducing the control of ESEO over the deployment process. The distributed binaries would be self-contained and digitally signed native software packages.

Both Android and iOS sandbox different applications on the operating system level by default, restricting the access to application resources by other applications. The applications are run in unprivileged user rights. These assumptions do not hold for rooted/jailbroken devices.

In case of native mobile applications, the associated OS process may be active for a long time after the user has seemingly ended it. On Android, the application can be in a cached state, on iOS in either background or suspended state. As the application can be restored from these states to the foreground states, it must be analysed, what data is allowed to be persistent in these states.

The responsibility for ensuring the platform security lies with the voter. It is advised that the operating system is kept up-to-date. However, anti-virus software is a controversial topic in this case. It can not be as efficient on mobile platforms as it does not have root access and therefore has limited options to detect malware. This is one of the reasons why Apple does not allow any fully functional anti-virus apps to be distributed for iOS devices. The security apps from anti-virus vendors that are available in App Store provide only limited functionalities.

General security level of mobile OSes is discussed in Section 3.2.1. A specific issue with Android is that the base OS upgrades created for vanilla version of Android may not propagate to the vendor and device specific branches in a timely manner; see Section 3.2.2 for more details.

The platform specific mobile voting application approach has been attempted by at least two providers of online voting systems for governmental elections – Voatz and Votem. Before analysing our local case, we give a short overview of the lessons learned by these companies.

4.3.1 Voatz and Votem

Voatz[58] implements an online voting system that provides voters with both Android and iOS applications to vote with. The online voting system is built on top of permissioned blockchain, biometric authentication and mixnet. Voatz notes that while it is desirable to support browser-based voting on PCs, it is important to realize that not all the security features available to smartphone users can be incorporated into browser-based voting applications[59].

There is limited amount of public information about the actual underlying voting protocol and assumptions it uses to achieve its security targets. Information provided by Voatz is marketing-oriented, and open-sourcing is rejected with a reference to competitive advantage with proprietary technology[60].

On the other hand, Voatz was used in 2018 midterm federal (!) elections in West Virginia, US[61].

[57]https://www.idc.com/promo/smartphone-market-share/os
[58]https://voatz.com/
[59]Voatz Security Whitepaper, https://voatz.com/voatz-security-whitepaper.pdf
[60]https://voatz.com/faq.html
[61]The West Virginia Mobile Voting Pilot, https://sos.wv.gov/FormSearch/Elections/Informational/

Recently a group of researchers from MIT reverse engineered the Voatz Android application (acquired via Google Play Store as of 01-01-2020) and gave extensive overview of its architecture and shortcomings [SKW20]. The app and the protocol turned out to be unsuitable for secure elections, nevertheless the app-specific security features are still of interest to us.

One of the strongest claims of Voatz was that the Voatz app does not permit a voter to vote if the operating system has been compromised. The following security measures were detected by researchers [SKW20].

- The app's long-term storage was encrypted with a key derived from the PIN or fingerprint. This feature made brute-force attack against ballot-secrecy possible, partially because the Android Keystore was used the wrong way.
- The app contained a third-party anti-virus and had hooks to it to report to the central system in case a threat is detected in the app's running environment. It turned out that there was an easy way to turn off this feature dynamically by simply overriding the hooks.
- The app was partially obfuscated (both on the APK level and app level) causing some of the decompilation tools (APKTool, IDA) to crash, while JADX and Ghidra remained usable. The app contained zip-bombs as some of its resources. The APK was downloaded on 2020-01-01 and the research was published on 2020-02-13, showing that although the obfuscation was a barrier, it did not win too much additional time for the app.

Voatz Android application was deployed for phones running Android OS version 6+. Only phones with specific distributions of Android were permissioned to participate by app store preferences, leaving out voters with devices by ZTE and Huawei. Researchers point out that this enables attackers to trick the users of unsupported devices into installing an app containing malware by establishing a legitimate-looking website with information about how to vote and directing the reader to install a malicious version of Voatz app. Similar attack already took place when the game of Fortnite was distributed outside Google Play Store for cost saving reasons [SKW20].

On March 13th, 2020, a full report[62] on the Voatz mobile voting platform was published by Trail of Bits[63], a security testing company from US. Their security review based on white-box approach resulted in 79 findings (some of those mobile application specific) and confirmed all findings from the MIT black-box analysis.

Votem[64] is another US-based online voting provider, working on Android and iOS platforms and trying to take advantage of the blockchain technology. In contrast to Voatz, there is a plenty of public information about the voting protocol[65]. However, they haven't been successful in securing any governmental elections. There is currently no app for voting by them on Google or Apple marketplaces.

Votem seems to share the mindset with Voatz. Both iOS and Android client applications must detect the presence of a debugger if the application is not being run in a permissible build configuration. Potential modification, tampering with, or debugging of a live application should result in the creation of an audit entry that may be reported back to the web service. Once that re-

West-Virginia-Mobile-Voting-White-Paper-NASS-Submission.pdf

[62]https://blog.trailofbits.com/2020/03/13/our-full-report-on-the-voatz-mobile-voting-platform/

[63]https://www.trailofbits.com

[64]https://www.votem.com/

[65]Proof of Vote, https://github.com/votem/proof-of-vote

quest has been sent, the application should prevent any further action from being taken in the application and alert the voter that a potential attack has been detected.

We can conclude that Voatz should have invested less into the obfuscation of the app and more into development of secure online voting protocol. Both companies consider mobile devices as an insecure environment and propose anti-virus-like counter-mechanisms. Although appealing at first hand, the (cost-)effectiveness of such approach is not fully clear.

4.3.2 Architectural impact

In this Section we reconsider voting application requirements and architectural decisions in the light of platform specific mobile voting applications.

4.3.2.1 Functional requirements for a standalone mobile voting app

Functional requirements of a standalone PC-based voting application were as follows (see Section 4.1.1).

- Voting application implements voting for all Estonian national elections.
- One voting application has to support one and only one particular election.
- The look-and-feel of the voting application is completely controlled by the ESEO.
- Voting application is aware of the distributed voting service.

We would expect the functional requirements not to change drastically simply because the voting platform has changed. However, native mobile applications for Android and iOS are distributed over vendors' application markets (Google Play[66], Apple App Store[67]) and usually stay available for long time periods. It is possible to change the app behaviour based on the election being active or not, but it also introduces a potential for having continuous online voting service, requiring readiness to serve several different election events potentially at the same time.

It is also very likely that the ESEO control over the look-and-feel changes, implying that the configuration service will have to be hosted independently of the applications and it can change dynamically.

Together with the concern that the timeliness of the updates of the applications in the market is not completely under the control of ESEO any-more, this changes lifecycle of a voting application and induces changes to the server side (e.g. support for apps with older versions).

4.3.2.2 Development toolchain

Development toolchain related aspects of a standalone PC-based voting application were as follows (see Section 4.1.2).

- Voting application shares codebase on multiple platforms and architectures.
- Voting application limits the number of external dependencies.

On Android, the most popular programming language is Java, but the official support by Google has increased the usage of Kotlin as well. On iOS, Swift is preferred over Objective-C for new

[66]https://play.google.com/store
[67]https://www.apple.com/ios/app-store/

projects. On these default routes, there is no potential for shared codebase between different platforms meaning that both development and maintenance efforts would be doubled.

There are also cross-platform development opportunities that should be evaluated.

- Qt[68] is a cross-platform application and UI framework. The development takes place in C++, Qt Platform Abstraction makes it possible to support both desktop and mobile platforms with the same codebase. There is an opportunity to support all major platforms with Qt – Windows, Linux, macOS, Android and iOS. Note that one of the most important libraries for current voting application – OpenSSL – is also available for all these platforms.
- Xamarin[69] is an open-source mobile platform on .NET framework (development in C#). It enables the developer to create native apps that run across multiple platforms such as Android, iOS, and Windows.
- ReactNative[70] is an open-source mobile application framework. It enables the developer to create native apps for Android and iOS. In its usual form this would mean sharing the GUI codebase. Still there are ways to implement common modules in e.g. C and use them via ReactNative.

We note that application packages on Android and iOS are self-contained by default and the user experience of downloading and installing the application through the given market is uniform. On both platforms the supported API level is important as it defines the available functionality. On Android, many providers have their own tweaked versions of the base OS, thus the API promise does not always hold and source code checking for certain API versions and having multiple ways of getting the same outcome are common. For example all devices on Android 4.1+ should support TLS 1.2, but in reality this is not the case[71]. This means that it might be useful to lock certain functionalities (e.g. TLS, cryptography) into the application itself in a form of e.g. C library that can be used over Java Native Interface (JNI).

4.3.2.3 Transport layer security

Transport layer security related aspects of a standalone PC-based voting application were as follows (see Section 4.1.3).

- Voting application implements its own TLS-stack and completely controls its parameters.
- Voting application uses mutual authentication.
- Voting application uses certificate pinning.

On both platforms there will be no problems supporting TLS in a manner required by the voting protocol (e.g. TLS 1.2, SNI).

On Android, SNI support was added in 4.2 and TLS 1.2 support in 4.1. There is also an opportunity to use OpenSSL over JNI.

On iOS, SNI support was added in version 4 and TLS 1.2 support in version 5. Again, there is an opportunity to use OpenSSL via Objective-C.

[68]`https://www.qt.io/`
[69]`https://dotnet.microsoft.com/apps/xamarin`
[70]`https://reactnative.dev/`
[71]`https://ankushg.com/posts/tls-1.2-on-android/`

4.3.2.4 Encryption and digital signature

Encryption and digital signature related aspects of a standalone PC-based voting application were as follows (see Section 4.1.4).

- Voting application requires cryptographically secure randomness.
- Voting application requires encryption APIs.
- Voting application gives and verifies digital signatures.

On Android, cryptographically strong random number generator is provided by java.security.SecureRandom[72]. On iOS, SecRandomCopyBytes[73] fulfils the same task.

There are also no issues in implementing the required encryption and digital signature methods with surrounding objects (such as X.509 certificates). However, since Android ships with cut-down and out-of-date BouncyCastle cryptographic library, an application specific SpongyCastle cryptographic library has to be set up[74]. It would also be possible to use OpenSSL via JNI. On iOS, the use of OpenSSL as crypto provider is possible. The required functionality has already been implemented for vote verification application on both Android and iOS.

4.3.2.5 e-ID tool support

e-ID tool support related aspects of a standalone PC-based voting application were as follows (see Section 4.1.5).

- Voting application requires ID-card APIs.
- Voting application requires mobile-ID APIs.

Both e-ID tools are currently supported on both platforms by RIA DigiDoc app[75] (available since 2017/2018). ID-card is connected to the smartphone with USB Type-B or Type-C card-reader. The driver code is part of the application (in both cases encapsulated into a library that should be reusable). It is interesting to note that both applications take platform native development path and no code cross-use is taking place.

It is also known that the entire electronic functionality of new Estonian ID-card can be used also over the contactless NFC interface[76]. This would enable readerless operation on smartphones with NFC support.

As mobile-ID does not require OS level interactions with hardware, it is the simpler e-ID tool to support. However, in the IVXV protocol suite the vote collector service is used as a mediator between the voting application and DigiDocService / mobile-ID REST API service, and no direct communication between the two is taking place. See Section A.6.3.2 for a discussion of the potential implications.

4.3.2.6 Usability and accessibility

Usability and accessibility related aspects of a standalone PC-based voting application were as follows (see Section 4.1.6).

[72]`https://developer.android.com/reference/java/security/SecureRandom`

[73]`https://developer.apple.com/documentation/security/1399291-secrandomcopybytes`

[74]`https://rtyley.github.io/spongycastle/`

[75]`https://www.id.ee/index.php?id=39150`

[76]`https://github.com/martinpaljak/esteidhacker/wiki/NFC`

- Voting application provides graphical user interface.
- Voting application implements accessibility interfaces.

Both platforms require dedicated user experience development.

Android provides android.view.accessibility[77] API and TalkBack[78] screen reader. iOS provides UIAccessibility[79] API and VoiceOver[80] screen reader. Depending on the choice of development tools, good knowledge of World Wide Web Consortium (W3C) Accessible Rich Internet Applications recommendation (WAI-ARIA)[81] might be needed.

The impact on voter experience is discussed in Chapter 5. However, we do not foresee major technical difficulties in achieving the goal of accessible and usable voter interface on Android or iOS.

4.3.2.7 Deployment

Deployment related aspects of a standalone PC-based voting application were as follows (see Section 4.1.7).

- Voting application has a small footprint.
- Voting application is configurable by ESEO.
- Authenticity and integrity of a voting application can be verified.
- Critical updates for voting application can be deployed in timely manner.

With the standalone PC voting application, the deployment lifecycle is as follows.

- Vendor compiles the release-candidate binaries.
- ESEO creates the JSON configuration file for the application.
- ESEO configures the release-candidate binaries to get the configured application.
- Vendor signs the configured application with its codesigning key to get the signed application.
- ESEO representative signs the checksums of the signed application.
- ESEO publishes the signed application and its own signatures on its website.

The process is under the complete control of ESEO, assuming that a proper service-level agreement is in place between the vendor and ESEO.

With Android voting application, someone – private person, institution, ESEO, vendor – would have to have an account with the Google Play Market. The app is published via the market and has to adhere to the requirements of the market (e.g. have privacy policy, self-rating of app content, etc.). Although unlikely, it might mean that the voting application is removed from the store for e.g. not specifying its target audience.

[77]https://developer.android.com/reference/android/view/accessibility/package-summary
[78]https://support.google.com/accessibility/android/answer/6283677
[79]https://developer.apple.com/documentation/uikit/accessibility/uiaccessibility
[80]https://support.apple.com/en-gb/guide/iphone/iph3e2e415f/ios
[81]https://www.w3.org/TR/wai-aria/

Once ESEO has a digitally signed APK that can be uploaded to Google Play, the specific point in time when the new version shall become visible for the voters is not known. For the verification app (EH Kontrollrakendus), the updates have appeared in matter of hours, but Google warns that review times may be 7 days or longer in exceptional cases[82]. When there is a critical update for an ongoing/upcoming election, the 7 day review time might be too long.

On iOS, the app review times have been historically lengthy, but have shortened significantly[83] over past years averaging to 48 hours for 90% of the apps[84]. One may assume that the worst case scenarios are similar to Google's. Apple makes it possible to request expedited reviews for time-critical issues.

In case of Android, the APK could be additionally distributed by the ESEO itself. With iOS, there is no such workaround on a scalable level.

Both Android APK and iOS iOS App Store Package (IPA) are digitally signed by the vendor. On Android, there is an option of using keys managed by Google. However, for the purpose of voting application, the signing keys should be under control of ESEO. Both platforms ensure integrity of the app installed into the device, this is similar to the current standalone voting application.

On Android, additional layer of integrity verification can take place – reproducible building. In case of the current verification app, there is a procedure[85] allowing someone with development tools carry out building to determine that the signed APK published in the Google Play store is indeed built from the source code published on GitHub. This procedure is enabled by the fact that it is possible to extract APK from the phone and the Java applications can have deterministic builds.

On iOS, reproducible building is possible (see e.g. the Telegram app[86]), but would require a jailbroken device.

4.4 Voting application as a web-application in browser context

Instead of implementing mobile platform specific native applications, it is possible to implement voting application as a JavaScript/TypeScript based web application that can be run with a supported browser (on certain occasions, some OS related restrictions may apply). The side effect of this approach is that the resulting application is usable on any platform (including desktop) where there is a supported browser.

The voting application for browsers would have to be distributed via some web server that could be under the direct control of ESEO. The application would be a bundle of JavaScript, HTML and CSS files. There is no standard way to digitally sign a JavaScript web-application, it would only be possible to verify the authenticity of the web-server with help of a TLS certificate.

The web-application would be interpreted and executed by the browser and the isolation of the voting application from other, potentially malicious web-applications, add-ons and extensions would depend entirely on the browser. For example, some browsers (Chrome, Safari, Firefox) properly isolate different applications into different OS-processes, whereas some browsers (e.g. Puffin) may use completely unacceptable models for online voting (e.g. cloud based rendering,

[82]`https://support.google.com/googleplay/android-developer/answer/6334282`

[83]`https://appreviewtimes.com/`

[84]`https://developer.apple.com/app-store/review/`

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

[86]`https://core.telegram.org/reproducible-builds`

effectively removing all privacy). Although there are dozens of different browsers with JavaScript support available, the actual market share is not so mottled. Browser market share in Estonia, in December, 2019 was the following[87]:

- Chrome, 66%
- Safari, 15%
- Firefox, 9%

On desktop, both Edge and IE were represented, on mobile devices Samsung Internet was also relevant.

The browser based approach makes the landscape for voting application more complex. In addition to the security of underlying OS environment, the security of specific browser environment has to be considered. An additional layer of potentially malicious or vulnerable software – browser extensions – comes into play. These topics are discussed in Sections 3.4.4 and 3.4.8.

The voting-application-in-browser approach has been attempted by at least two providers of online voting systems for governmental elections – Scytl and Smartmatic-Cybernetica Centre of Excellence for Internet Voting. We give short overview of the lessons learned by these companies.

4.4.1 Scytl

Scytl[88] is an international election technology company based in Barcelona, Spain. Among their products, there are online voting solutions for both governmental and organizational landscapes. In 2019, Scytl had online voting projects in New South Wales (Australia), France and Neuchâtel (Switzerland) among others. They also suffered a very controversial year, having issues with convincing the community and customers in the security of their systems. Their solution got negative feedback from public hacker test in Switzerland[89], negative feedback from the research community in Australia[90], and election in Åland, Finland, was cancelled in the last minute due to security concerns[91].

Until 2013, Scytl used Java Applet technology to vote in the browser context. The evolution of HTML5 and JavaScript together with browsers abandoning the support for Java, led Scytl to transition to a JavaScript based voting application [CGG16].

Scytl elaborates on the following aspects on their transition:

- improved usability due to modern light-weight technologies,
- availability of required cryptographic primitives,
- availability of secure random numbers and entropy,
- code authenticity and integrity verification,
- proper use of third party libraries.

[87] https://gs.statcounter.com/browser-market-share/all/estonia/2019
[88] https://www.scytl.com/
[89] https://portswigger.net/daily-swig/swiss-post-puts-e-voting-on-hold-after-researchers-uncover-critical-security-errors
[90] https://www.itnews.com.au/news/nsw-electoral-commission-confirms-ivote-contains-critical-scytl-crypto-defect-520460
[91] https://www.val.ax/nyheter/ingen-rostning-internet

The transition from Java to JavaScript wasn't painless. In order to mitigate the lack of secure random numbers on some platforms at that time, Scytl had implemented their own entropy gathering process and a pseudo-random generator. In 2013 parliamentary pilot election in Norway it turned out that due to faulty initialisation procedure, the pseudo-random number generator would generate the same output in more than 50% of the cases. This meant that voters' choices were effectively unencrypted [BGS+16].

In case of 2015 state elections in New South Wales, Australia, researchers showed that due to improper dynamic inclusion of third party code, the voting system was subject to both then recent FREAK and Logjam attacks. The latter would have allowed a man-in-the-middle to inject arbitrary malicious code into the voting application. The proof-of-concept attack was implemented for test-voting site [HT15].

4.4.2 Smartmatic-Cybernetica Centre of Excellence for Internet Voting – TIVI

Smartmatic-Cybernetica Centre of Excellence for Internet Voting[92] is a subsidiary of Smartmatic[93], a global provider of electronic voting technologies, and Cybernetica AS[94]. The joint venture develops online voting protocol suite and platform TIVI[95], which has been used in municipalities and organizational elections in Chile, USA, Norway and Estonia[96].

TIVI came to life in 2014, its voting application has always been in JavaScript. In addition, TIVI comes with native voting application for Windows, Linux and macOS platforms. However, the customer expectations have preferred browser-based solutions so far because of their simplicity for the voters.

TIVI has similar approach to solve the entropy problem for JavaScript as Scytl. In addition to entropy collected via the browser, TIVI takes advantage of a server-side entropy endpoint and combines the two sources to get the required random numbers for encryption.

In 2016, TIVI was used in Republican Presidential Caucus in Utah [KKW18]. The voting application was deployed at ivotingcenter.us. At the time of the Caucus, an imposter site was set up at ivotingcenter.gop[97]. This site took the source code from the original site and added a functionality to show a warning if someone tried to vote there.

TIVI voting application has been tested on major browsers – Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, Internet Explorer. In all elections there have been several other browsers voting successfully – Pale Moon, Maxthon, UC Browser, Vivaldi, etc. In those occasions where an election event has had a Facebook communication channel, also Facebook's built-in browser has been used. This is an explosion of platforms to be supported in comparison to standard desktop platforms.

4.4.3 Architectural impact

We will now reconsider voting application requirements and architectural decisions in the context of a browser-based voting web application.

[92]`https://www.ivotingcentre.ee`
[93]`https://www.smartmatic.com`
[94]`https://www.cyber.ee`
[95]`https://tivi.io`
[96]`https://www.valimised.ut.ee`
[97]`https://bradblog.com/?p=11624`

4.4.3.1 Functional requirements

Functional requirements of a stand-alone voting application were as follows (see Section 4.1.1).

- Voting application implements voting for all Estonian national elections.
- One voting application has to support one and only one particular election.
- The look-and-feel of the voting application is completely controlled by the ESEO.
- Voting application is aware of the distributed voting service.

From the perspective of functional requirements, browser based voting application is similar to a standalone voting application.

4.4.3.2 Development toolchain

Development toolchain related aspects of a standalone voting application were as follows (see Section 4.1.2).

- Voting application shares codebase on multiple platforms and architectures.
- Voting application limits the number of external dependencies.

Browser-based voting application would bring the principle of shared codebase into its extreme – a single JavaScript application would be needed to support both mobile and desktop devices on a variety of operating systems and browsers.

Although, from the browser perspective, we speak about JavaScript, for development of the voting application a superset of JavaScript – TypeScript[98] – should be considered. TypeScript is a language that introduces types and static verification to JavaScript. It is backwards compatible and compiles into JavaScript for execution purposes. Static types and verification are preferable for a voting application to prevent type-related errors from occurring.

Additionally, it is likely that a JavaScript framework such as Angular[99], React[100] or Vue.js[101] has to be selected to streamline the user experience development. As this selection has strong impact on the overall composition of the application, it has to be considered in the light of other architectural aspects – expected loading times, accessibility mechanisms, security, etc.

With native solutions, even if the application consists of several subcomponents, it is deployed as a single installation bundle that originates from the single source. With web-applications, it is quite natural that some external dependencies are linked in dynamically at the time of use, potentially from external domains. We would advise against cross-origin resource sharing (see Section 4.4.1 above for the Scytl experience).

4.4.3.3 Transport layer security

Transport layer security related aspects of a stand-alone voting application were as follows (see Section 4.1.3).

- Voting application implements its own TLS-stack and completely controls its parameters.

[98] `https://www.typescriptlang.org/`
[99] `https://angular.io/`
[100] `https://reactjs.org/`
[101] `https://vuejs.org/`

- Voting application uses mutual authentication.
- Voting application uses certificate pinning.

In case of a browser-based voting application, approach to TLS changes completely. First and foremost – the voting application shall rely on the TLS implementation of the browser. Although TLS implementations in JavaScript exist (e.g. Forge[102]), it is not possible to get hold of a raw TCP socket in the browser context. This means that certificate pinning is not possible as browsers shall be relying on system certstore for verifying the other end of the communication. There used to be a HTTP public key pinning feature, but this has been deprecated[103].

Enforcement of TLS parameters and mutual authentication can be considered a purely server side issue. However, differences in TLS stack implementations in browsers potentially increase the workload both in development and deployment of a voting application. Desire to increase backwards compatibility / usability might introduce not-so-secure setups.

The lack of certificate pinning means that anybody capable of poisoning the system certstore can open a route for MITM attacks. This might be also the voter herself who introduces untrustworthy CA certificates into the certstore. These issues are discussed in Section 3.4.2.

4.4.3.4 Encryption and digital signature

Encryption and digital signature related aspects of a standalone voting application were as follows (see Section 4.1.4).

- Voting application requires cryptographically secure randomness.
- Voting application requires encryption APIs.
- Voting application gives and verifies digital signatures.

JavaScript native method (Math.random) for random number generation is not suitable for use in cryptographic protocols. This forced Scytl to implement an entropy collection mechanism together with a pseudo-random number generator (see Section 4.4.1). Entropy is gathered from e.g. mouse events, keyboard events, device accelerometer, AJAX calls, etc. [CGG16]. TIVI has a similar entropy gathering mechanism, additionally including entropy provided by the server.

There exists a W3C recommendation for Web Cryptography API[104], describing the method `getRandomValues` to obtain cryptographic random values in the browser. This method is today supported by all major browsers on all major platforms. Most likely, the best way forward is to continue using multiple entropy sources, while combining them with well known methods such as described in [BDPA10].

JavaScript does not itself provide cryptographic functionalities necessary for a voting client. We are aware of several third-party libraries that can be used for different purposes and in some combination could solve all cryptographic needs of a voting application. There is no OpenSSL-like library that would solve all problems, though.

- jsrsasign[105] contains necessary primitives for digital signature provision and verification.

[102]`https://digitalbazaar.github.io/forge/`

[103]`https://developer.mozilla.org/en-US/docs/Web/HTTP/Public_Key_Pinning`

[104]`https://www.w3.org/TR/WebCryptoAPI/`

[105]`http://kjur.github.io/jsrsasign/`

- Stanford Javascript Crypto Library[106] contains all major low level primitives for both encryption and signing, but lacks RSA and Public Key Infrastructure (PKI) related implementations.
- Forge by Digital Bazaar[107] provides a variety of PKI related tools, which makes it interesting from the digital signature provision and verification point of view.
- Verificatum JavaScript Cryptography Library (VJSC)[108] is developed by Douglas Wikström and is rooted in his work on the Verificatum mix-net. It contains all the necessary routines for implementing ElGamal based vote-protection protocols. However, Verificatum has unorthodox views in terms of standards such as ASN.1, and implements its own binary representations which requires changes also on the collection and tabulation end of the processing.

4.4.3.5 e-ID tool support

e-ID related aspects of a stand-alone voting application were as follows (see Section 4.1.5).

- Voting application requires ID-card APIs.
- Voting application requires mobile-ID APIs.

Both e-ID tools are in principle supported by browsers. We note that for ID-card support, hwcrypto suite[109] could be used. With Estonian ID-card, signing is supported on Chrome, Firefox, IE and Edge. However, we cannot confirm if the hwcrypto is currently functional with mobile browsers. There is only a limited number of browsers supporting extensions that would be required by hwcrypto approach, and we consider this as an extremely experimental path to take with high likelihood for a failure. Browser-based voting application for mobile devices would have to rely on mobile-ID.

4.4.3.6 Usability and accessibility

Usability and accessibility related aspects of a stand-alone voting application were as follows (see Section 4.1.6).

- Voting application provides graphical user interface.
- Voting application implements accessibility interfaces.

WAI-ARIA provides ontology of roles, states and properties that define an accessible user interface. This recommendation is supported by all major browsers on all major platforms, providing accessibility tools with necessary information about the web-application via native interfaces. Additionally, Web Content Accessibility Guidelines (WCAG)[110] assist web application developers in creating accessible interfaces. Community-driven accessibility efforts such as A11Y Project[111] and WebAIM[112] exist.

[106] `http://bitwiseshiftleft.github.io/sjcl/`
[107] `https://github.com/digitalbazaar/forge`
[108] `https://github.com/verificatum/verificatum-vjsc`
[109] `https://hwcrypto.github.io/`
[110] `https://www.w3.org/TR/WCAG20/`
[111] `https://a11yproject.com/`
[112] `https://webaim.org/`

All aforementioned JavaScript frameworks (Angular, Vue.js and React) are accessibility-aware and provide means to implement an accessible user experience. However, it is a separate design and testing process with potential impact to the visual user experience as well.

The impact on voter experience is discussed in Chapter 5. However, we do not foresee major technical difficulties in achieving the goal of accessible and usable voter interface with browser-based technologies.

4.4.3.7 Deployment

Deployment related aspects of a stand-alone voting application were as follows (see Section 4.1.7).

- Voting application has a small footprint.
- Voting application is configurable by ESEO.
- Authenticity and integrity of a voting application can be verified.
- Critical updates for voting application can be deployed in timely manner.

Voting application as a web application is similar to the standalone voting application in terms of the ESEO control over the deployment process. The downside in comparison with the standalone application is the need to refresh the webapp in one's browser to get hold of critical updates.

Application footprint is defined mostly by the underlying framework and the size can be reduced by minification and compression.

The major difference with other approaches is verification of authenticity and integrity of a voting application. On desktop platforms, the voting application could in principle be distributed even via an unreliable channel and then the integrity and authenticity of the application could still be verified post-transit, given that the information about how to verify is distributed over reliable channels. On some platforms (Windows, macOS), integrity would additionally be verified by OS at the load time. In case of web applications these safeguards are missing, which makes this approach questionable for the implementation of an online voting application.

With web applications, it is of utmost importance that voter starts her journey at the authentic web-site, and even then there is no assurance that the web content distributed by the server is the web content that was deployed by the ESEO. Neither are there any means for the voter to verify the integrity of the JavaScript code. Scytl proposed a tool – wraudit – to verify remotely that proper web-application is distributed by the web server [SCJ18]. This kind of a tool mitigates the risk that web server is delivering tampered content, but it alone does not mitigate the risk that voter accesses a tampered web application or that the web application is tampered with on the voter's device. The risks related to phishing web sites that may trick voters into using a non-authentic voting application are discussed in Section 3.4.9.

There are some mechanisms that slightly improve the situation – W3C recommendation Subresource Integrity[113] defines a mechanism by which user agents may verify that a fetched resource has been delivered without unexpected manipulation. An HTML-file would contain a SHA-384 hash for scripts included and the browser would only execute the script if the integrity verification succeeds. This feature is not fully supported on several browsers (IE, Edge, Mobile Safari)[114].

[113] https://www.w3.org/TR/SRI/

[114] https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity

This feature makes it possible to publish a minimal index.html with verification data also in a human-readable form instructing people to quickly review the source of the website they are going to use.

Cross-Origin Resource Sharing (CORS)[115] protocol provides web servers with the capability to inform browsers if certain resources can be shared cross-origin. A voting backend could inform browsers that only requests from `https://www.valimised.ee/` should be received. This would prevent a browser from enabling communication between a fake voting app and the real back-end. CORS, however, could be overcome by a fake voting app with a fake back-end, proxying queries to the true back-end. Even without communicating with the true back-end, a fake voting application could disenfranchise voters.

To further complicate the protection of voting application integrity, there exist browser extensions that execute in the background and in a separate execution environment, inject (potentially malicious) JavaScript directly into webpages, and modify the Document Object Model (DOM)[116]. Browser extensions can severely interfere with web applications control flow and visuals. We note that experimental methods to ensure DOM integrity exist[117]. Security issues related to the browser extensions are discussed in more detail in Section 3.4.4.

4.5 Discussion

We have identified the following candidate solutions for mobile voting application:

- platform specific mobile app developed with native tools;
- platform specific mobile app developed with cross-platform tools;
- browser based web-app.

It is technically feasible to implement the voting application for smartphones by any of these approaches. However, there are some caveats.

With platform specific mobile apps there exist necessary tools to support majority of architectural requirements of voting application – encryption libraries, ID-card support, flexible TLS support. The biggest change would be in the deployment model where platform specific marketplaces have the final say about if and when the voting application (or its critical updates) will be published. For Android, the risk can be mitigated by sharing the APK via some external channel. For iOS, no such scalable workaround exists.

With a browser based web-application, ESEO can maintain full control over both initial deployment and updates, however, there are other areas that raise flags. The browser itself is an additional abstraction layer added to voter's environment, becoming part of an attack surface. Some features hidden from the voting application by the browser force voting app to rely on browser TLS implementation and central certificate store. Web-application is inherently open-source, and all major browsers come with developer tools that make it possible to change the control flow and data of the application. This could lead to increased numbers of invalid votes by honest but curious hackers.

A significant downside of the browser based solution is the incapability of the voter to verify the integrity of the voting application. We note that with platform specific mobile apps there is no

[115]`https://fetch.spec.whatwg.org/`
[116]`https://blog.jscrambler.com/case-study-mitigating-browser-extension-attacks/`
[117]`https://toreini.github.io/projects/domtegrity.html`

direct way for a voter to verify the app; however, we do know that those apps are signed and these signatures are verified on respective platforms.

To conclude, we say that it is possible to mitigate the risk associated with platform specific marketplaces in development roadmaps and with public test elections. We are not aware of any good mitigations for the web-application integrity problem.

5 Voting application from the voter perspective

In this chapter we review the usage and complexity of the potential mobile voting apps from the voter perspective. We focus on the core use-case – casting a preference as a vote in an election – and briefly discuss installation, user experience and accessibility. We describe two candidate architectures – platform specific app (we identified 2 potential technical solutions) and browser based web-app (we identified 1 potential technical solution).

5.1 Standalone PC voting application

First, we will describe the current situation in order to compare how it will change with one or the other of the candidate architectures.

5.1.1 User interface

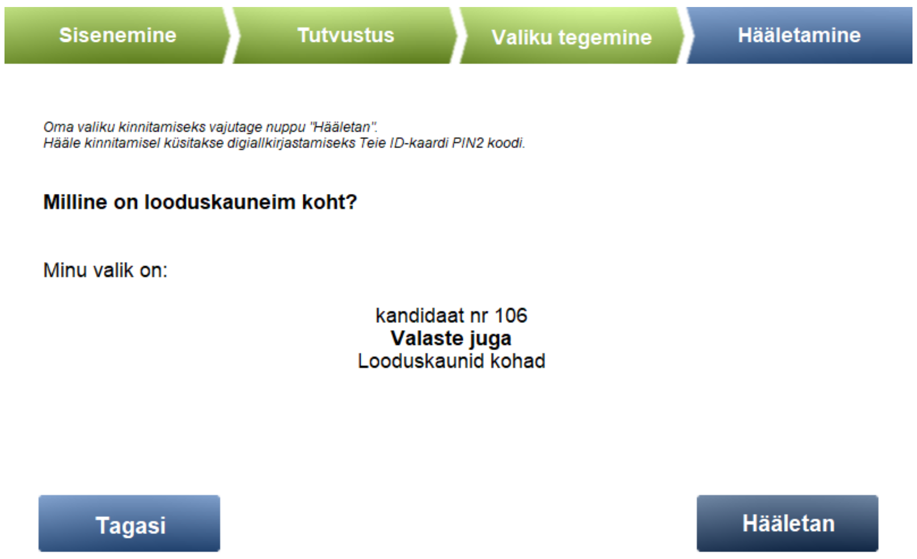
Current standalone PC voting application is a wizard that guides the voter through the process of voting – authentication, selecting and encrypting a candidate, confirming the selection with digital signature, casting the vote and acquiring the cryptographic material required for individual verifiability. The process can be cancelled by the voter any time before the actual confirmation; it is important that the voter does not accidentally vote for an undesired candidate. It is also important that there is no single screen where both the voter's identity and selected candidate are displayed together. The ballot presentation has to be aligned with the electoral law, most importantly that the presentation order of the candidates is fixed.

The user interface (Figure 2) has a fixed size of 800×500 pixels. It consists of 9 views for main steps in the process and about a dozen views for supportive features (such as error screens, PIN dialogues, modals).

The user interface can be navigated solely with mouse (except for the PIN-entry). It is also possible to carry the whole process out only with the keyboard (TAB, SPACE, arrows, ENTER, digit keys).

Main views consist of a roadmap widget, rich-text area and two buttons. Additionally, there is a tree-view widget to present hierarchical list of the parties and candidates. This tree-view widget has recently been augmented with a filter to quickly find the desired candidate. Verification QR-code is displayed as an image on one of the views.

Platform specific widgets are only used for PIN-entry dialogues on Windows. All other widgets are drawn by the application so that they look identical on all desktop platforms.



Oma valiku kinnitamiseks vajutage nuppu "Hääletan".
Hääle kinnitamisel küsitakse digiallkirjastamiseks Teie ID-kaardi PIN2 koodi.

Milline on looduskauneim koht?

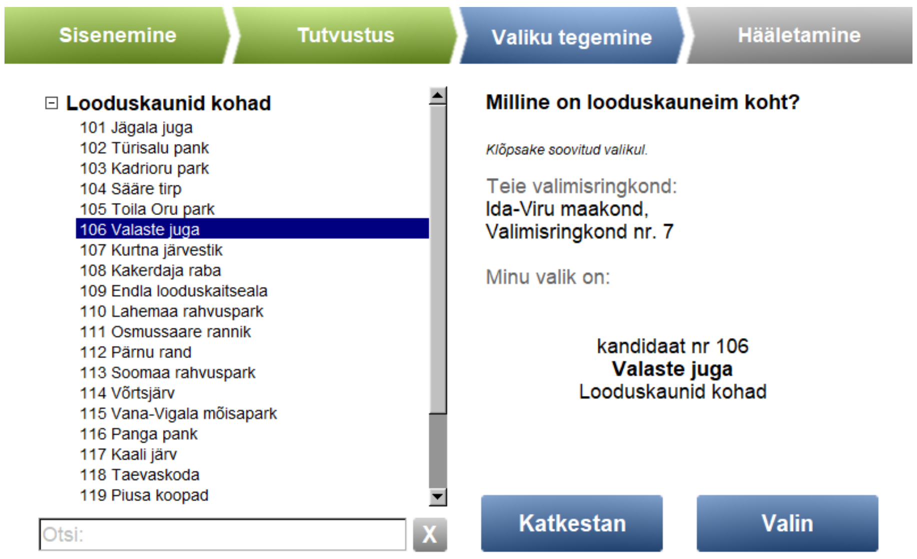
Minu valik on:

kandidaat nr 106
Valaste juga
Looduskaunid kohad

Figure 2. Voting application confirmation view.

The amount of text per view can be as high as 500 symbols and six paragraphs per view (usually the final view contains the largest amount of information).

The most complex view is where the candidates are presented grouped by parties in the official order (Figure 3) – there it is possible to make a selection and the current selection must be visible. There is a principle that the current selection has to be explicitly deselected before a new selection can be made. Additionally, if the current filter does not match the current selection, the current selection still must be displayed to the user.



⊟ **Looduskaunid kohad**
101 Jägala juga
102 Türisalu pank
103 Kadrioru park
104 Sääre tirp
105 Toila Oru park
106 Valaste juga
107 Kurtna järvestik
108 Kakerdaja raba
109 Endla looduskaitseala
110 Lahemaa rahvuspark
111 Osmussaare rannik
112 Pärnu rand
113 Soomaa rahvuspark
114 Võrtsjärv
115 Vana-Vigala mõisapark
116 Panga pank
117 Kaali järv
118 Taevaskoda
119 Piusa koopad

Otsi:

Milline on looduskauneim koht?

Klõpsake soovitud valikul.

Teie valimisringkond:
Ida-Viru maakond,
Valimisringkond nr. 7

Minu valik on:

kandidaat nr 106
Valaste juga
Looduskaunid kohad

Figure 3. Voting application candidate selection view.

The current standalone PC voting application has limited support for assistive technologies to allow visually impaired people to vote. The voting application supports keyboard navigation and the voting process has been tested with JAWS, NVDA and Narrator screen readers. The voting application provides feedback to the assistive technologies, but does not accept any input from them as a part of countermeasures to scripting attacks. This, however, may prevent access for people with e.g. motor disabilities.

5.1.2 Voting process

The voting process slightly differs based on the e-ID tool used by the voter. The voter must carry out the following steps in order to be prepared to vote.

- The voter MUST have a computer with Internet connectivity and a supported operating system.
- The voter SHOULD have installed all critical updates for her operating system.
- The voter MUST download the voting application to her computer.
- The voter SHOULD install verification app into her smartphone.
- The voter SHOULD verify the authenticity and integrity of the voting application based on the generally available trustworthy information.

A voter with mobile-ID is now ready to execute the application and vote.

A voter with an ID-card must additionally carry out the following steps.

- The voter MUST have ID-card drivers installed.
- The voter SHOULD ensure that latest ID-card drivers have been installed.
- The voter MUST have a functional smart-card reader connected to the computer.

A voter with ID-card is now ready to execute the application and vote.

Voting itself follows in the wizard-like manner.

- The voter selects an e-ID tool for voting and initiates authentication.
- The voter enters PIN1.
- The voter selects the desired candidate from the list.
- The voter reviews the selection and signs the vote by entering PIN2.
- The voter captures the QR-code for individual verification and quits the application.

It is then safe to delete the binary from the computer. For critical updates, new voting application binaries must be deployed and downloaded by the voters affected.

We note that a standalone PC voting application does not require installation in the technical sense of the word. The self-contained binary is downloaded, it can be executed, used and removed by deletion without leaving any trace in the system. There is no complex installation process – no files in system directories, no modifications in the registry, etc. However, the layman perception is that since the application has to be downloaded, it is also installed, and this is a complex experience.

The optional sub-process of integrity verification is enforced on Windows and macOS platforms with the help of code-signing. Since the binaries are signed with an extended validation code signing certificate, they are executed without any visible interference by either the Gatekeeper or

SmartScreen filter. However, the voter should check that the binary to be executed comes from the expected source.

Additionally the voters could verify the SHA-256 hashes published by ESEO in the digitally signed form (BDOC/ASICE). It is not known how many voters follow this scenario.

The voting application should be downloaded from the ESEO website, the URL of the website should be entered into the address-bar by the voter directly.

In several elections, voters have reported the voting application being quarantined by anti-virus solutions.

5.2 Platform specific mobile voting application

5.2.1 User interface

The current standalone PC voting application is small and simple in terms of views and interactions. However, it is clear that it cannot be replicated for the smartphone voting experience. We would like to point out the following aspects.

Smartphones come in variety of screen sizes and can be used in either portrait or landscape orientation. The variation of screen sizes increases if we include tablets. It follows that adaptive user interface design must be used. Safe areas on different devices would have to be taken into account. The screen real estate on smartphones is more expensive than on PC platforms, and the amount of information conveyed is small. Complex interactions with multiple UI elements presented in a single view on PC would have to be restructured into multiple views for smartphones.

We note that PC is mostly used in landscape orientation and applications are presented as windows, whereas on smartphone the preferred orientation is portrait, and single application usually occupies the whole viewport.

The mainstream way to interact with smartphones and tablets is via tapping and gestures (as opposed to keyboard or mouse navigation on PC). The low precision of tapping with a finger requires larger controls, the use of gestures has to be in line with the gesture language of the underlying platform.

The smartphones are used in a more mobile way than (laptop) PCs. We can expect people to vote while sitting in the crowded bus or crossing a street.

Both Android and iOS provide their own guidelines[118] [119] for the design of applications, pointing out the need for apps to look and behave in a way that is consistent with the platform.

Both platforms provide assistive technology and hooks to support it by the apps.

5.2.2 Voting process

In case of a platform specific mobile voting application, the core of the voting process remains essentially same. However, there are differences in the installation, authenticity verification and ID-card usage.

[118]https://developer.android.com/design
[119]https://developer.apple.com/design/human-interface-guidelines/

Platform specific mobile voting application is installed into the voter's device via platform's marketplace. This process is uniform for all applications and should be familiar to most of the voters. Although this time the application is truly installed, it is likely perceived as a natural process by the voters with shortcut to application showing up in an expected place for the voter. The process of guiding voters to the correct application becomes more complicated. For those voters who click the link on `https://www.valimised.ee/`, the process is clear. However, some voters might use the search tools of the platform marketplace which may in addition to the correct application show also unrelated, potentially adversarial applications. The risk is even higher when devices unsupported by the voting application are concerned.

Critical application updates are distributed via platforms' marketplaces and shall reach majority of voters automatically (even though there may be a significant delay that is hard to control by the application vendor).

With platform specific mobile applications, the voter has limited means of verifying the authenticity of the application. In case of a Windows executable there is a possibility to use Windows Explorer for examining the application signature. Linux users can be expected to be able to compute the SHA-256 checksum by command line tools and match it to the one found on `https://www.valimised.ee/`. Unfortunately, there are no such easily accessible tools on e.g. Android. Thus it is essential that a correct application is downloaded, and then the platform has to be trusted for the integrity. Both Android APKs and iOS IPAs are digitally signed, but at least in case of Android, these signatures are not necessarily as strong as we would like them to be (see Section 3.2.3 for more details).

With a platform specific application, supporting ID-card is possible. The PC application requires the installation of ID-card drivers, but in case of a mobile app these drivers would be part of the voting application code. We note that the whole ID-card scenario would be very experimental in first couple of elections. It is unlikely that there are many people with compatible card readers, the user experience with USB readers would be clumsy, and the more natural NFC based access is not yet officially supported. We would expect mobile-ID (or Smart-ID in case it gets approved) to be more popular e-ID tool(s) on smartphones.

5.3 Voting application as a web application in browser context

5.3.1 User interface

Voting application as a web application has to take into account all properties of smartphones in a similar manner to the platform specific mobile application. There is a slight difference in terms of the actual design language of the app – the same app would be accessed from Android, iOS and PCs, so it would not be so tightly coupled with any of these platforms.

An additional consideration is the need to also support the PC platforms. As there is no efficient way to prevent people with PCs voting from their desktop browsers, the need for responsive design has to be addressed from early on.

In addition to the app development, the support for assistive technologies also depends on the underlying browser, its implementation of relevant guidelines and compatibility with OS-level interfaces. With the mainstream browsers, sufficient levels of accessibility can be achieved.

5.3.2 Voting process

In case of voting application as a web application in browser context, there are differences in installation, usage and authenticity/integrity verification. ID-card shall not be an option on the smartphones, support for it may be present for the PC platforms. In this case, the drivers must be installed before the voting process can commence.

In terms of installation and usage, the web-based voting app might be perceived as the most light-weight and accessible solution by the voters. It requires a device with Internet connectivity and a supported browser. A mobile-ID user could vote on e.g. OpenBSD[120] as long as there is a browser with the necessary level of JavaScript support. Note that browser is almost a mandatory element of the voting process with other approaches as well. Additionally to the OS security, security of the browser becomes relevant for the voting environment. Assuring a sufficient security level is the responsibility of the voter, guidance and assistance is going to be expected from ESEO.

Critical updates are deployed as updates to the web application. Users who have used older versions might have trouble with locally cached versions.

Voter accesses the voting web application via entering its URL into address bar or navigating to it via `https://www.valimised.ee/`. From the voter's perspective it is important to verify the certificate of the TLS connection to ensure the authenticity of the origin. There are no integrity assurances for web content, the implied risks could be mitigated by wider use of individual verification tools. For a more detailed treatment of browser risks, see Section 3.4.

5.4 Discussion

Technologically, PC voting application and platform specific mobile application are more similar to each other than to the web-based voting application. The latter must take into account both PC and smartphone requirements. Even if originally developed with smartphone users in mind, it would attract voters on PC platforms as well. In mid-term, it could render the PC-based voting application obsolete.

From the voter perspective, the web-application would likely be perceived as the most uniform and light-weight solution that requires no installation. The technological consequences – such as lack of integrity assurances – of this approach are not going to be visible for the voter. If other considerations would direct towards platform specific apps, these considerations would have to be clearly communicated.

Regardless of the technological solution, the core voting process would remain essentially the same. However, since the communication channels for human-computer interaction differ significantly when switching from PC to smartphone, the user experience would have to be designed with those differences in mind while maintaining the same core principles – lawfulness, separation of the identity and the choice, conscious and confirmed decision-making. This should be seen as an opportunity to:

- revisit user-experience design for all voting applications on all platforms, including PC, and
- redefine the required level of support for assistive technologies on all platforms, and design assistive user experience to support specific needs of the respective focus groups.

[120] `https://openbsd.org/`

We note that regardless of the technical solution picked, guidelines for assuring the authenticity and integrity of the voting app have to be provided by the ESEO.

A Description and risk analysis of a possible alternative verification solution

A.1 General description of i-voting with verifiable delivery

Figure 4 depicts a general scheme of i-voting with verifiable delivery.

A.1.1 Parties

Voting Service is the technical environment that receives the votes cast and counts them to determine the results of voting.

Voting Device is the device that authenticates the voter, lets the voter make her choice, creates the vote, encrypts and signs her vote. Voting device has:

- *Key container* that holds voter's private signature key. Key container may be implemented either in software (encrypted file) or hardware (ID-card).
- *Voting application* is a software application that displays the voting options, encrypts the vote with the public key of the voting service, uses the key container to sign the vote, and sends the vote to the voting service.

Verifying Device is the device in which the voter verifies that the vote was received by the voting service. Verifying device has:

- *Verifying application*, i.e. software that downloads the cast vote from the voting service, obtains the verification code from the voting device, and displays user the vote.
- *Key container* (optional) that holds voter's cryptographic keys that might be needed to securely transmit the verification code from the voting application to the verifying application.

A.1.2 Protocols

Vote Casting is the protocol during which the voter

1. makes her choice,
2. encrypts the vote with the public key of the voting service, and
3. signs the vote using the private key.

Vote Download is the protocol during which the voter downloads her vote from the voting service to the verification device.

Verification Code Transfer is the protocol for transmitting the verification code from the voting device to the verifying device.

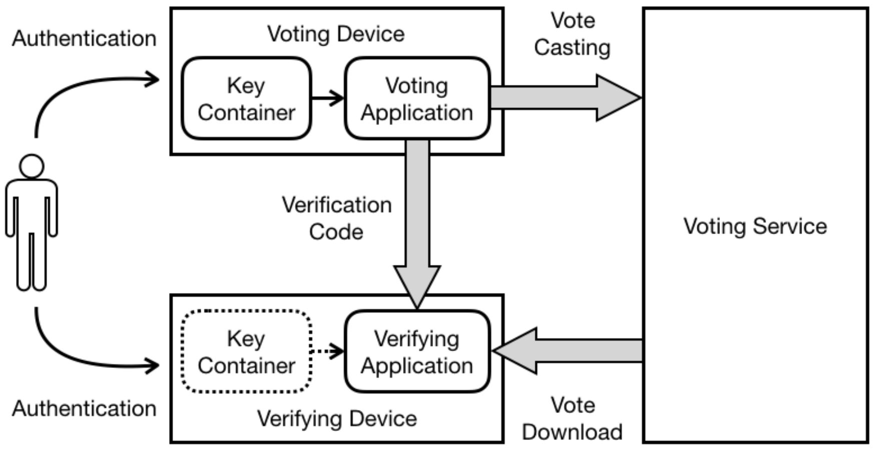


Figure 4. General scheme of i-voting with vote delivery verification.

A.2 Current solution

A.2.1 Voting

Voting Device is an ordinary desktop computer with Internet connection and optionally with ID-card interface. Voting applications are available for Windows, macOS and Linux operating systems. The authenticity of the application is checked at the desktop computer in all three cases.

Voter can use ID-card or mobile-ID for voting. The voter is asked to enter her PIN1 code. Mobile-ID users are asked to enter their mobile phone number and personal identification code. The list of candidates is then displayed and the voter makes her choice. The vote is encrypted by the voting application using the public encryption key of the voting service. The voter then enters PIN2 and signs the encrypted vote.

A.2.2 Verifying

Verifying Device is a mobile device such as a smartphone. If voters wish to check their votes, they must install the verification application on their smart device. The mobile device must have:

- a camera,
- a network connection,
- Android or iOS mobile operating system.

Verification application is downloaded from Google Play or App Store environment.

After voting, the verification code is displayed as a QR-code on the screen of the desktop computer. The verification application is then opened on the mobile device, and the QR code on the screen is scanned into the mobile device. During checking, the verification application uses the information in the QR-code to download the signed ballot from the voting server. It then takes the ElGamal randomness, which it got from the QR code, decrypts the vote and displays the candidate's name the voter voted for[121].

[121]https://github.com/vvk-ehk/ivxv/blob/master/Documentation/en/protocols/07-kontrollimine.rst

A.3 A possible PC application based verification scheme

If the voter would be using a mobile device to cast votes, she can not make use of the same mobile device for verification purposes, since we need the verification to happen in a reasonably independent device. What we could do to enforce this is to adjust the verification protocol so that a vote given on a mobile device could (only) be verified on a PC.

If we would like to stick to the present verification protocol, we need to get encryption randomness from the mobile device to the PC. If the PC (say, a laptop) has a camera, we can make use of the exactly same protocol. But if it does not, we need to introduce something new.

One option is to generate a one-time public-private key pair on the PC, transfer the public key to the mobile device (say, using a familiar QR-code trick) and let the mobile device to encrypt the encryption randomness using the public key obtained from the QR-code.

A modification of this scheme may use symmetric encryption key for confidentiality and a message authentication code (MAC) key for integrity. To combine the two, we can use one key with an authenticated symmetric encryption scheme, say, Advanced Encryption Standard block cipher in Galois Counter Mode (AES-GCM).

A.3.1 Devices and applications

Verification device is an ordinary desktop computer.

Voting application is a special app on the mobile device. It uses either ID-card or mobile-ID to sign the votes.[122]

Verification application is a special piece of software that can be downloaded and installed to the desktop computer.

A.3.2 Voting

After assisting the voter to cast her vote, the voting application keeps the verification code until either the verification time period expires or until the voting application is closed.

A.3.3 Verifying

1. Voter runs the verifying application on the desktop computer.
2. Verifying application generates temporary symmetric encryption and authentication keys and displays the key together the necessary additional information in the form of a QR-code.
3. Voter opens the voting application on the mobile device and scans the QR-code on the screen of the desktop computer.
4. Voting application encrypts the saved verification code, creates a message authentication code and sends the encrypted-and-MAC-ed verification code to the verifying application.
5. Verifying application, after receiving the message, decrypts it and checks the message authentication code.

[122]At the time of this writing (spring 2020), a decision is in the process of taking whether Smart-ID could also be used. In case the decision turns out to be positive, the current report also needs to be augmented with the treatment of Smart-ID.

6. Verifying application, based on the received the verification code, determines and displays the candidate's name the voter voted for.

A.4 A possible browser-based verification scheme

The scheme is the same as above but the verification application consists of JavaScript downloaded from a secure webpage of the voting service. However, the browser based approach may make it impossible to verify the integrity of the verification application. These issues are described in Sections 3.4.4 and 4.4.3.7.

A.5 General approach to risk analysis

In this Section, we describe the general approach of the risk analysis, followed by the analysis of m-voting specific risks (Sec. A.6).

There are several general requirements set for the elections (see, e.g. [HW14a]).

- Authenticity – only eligible voters are allowed to vote.
- Freedom – each voter can vote according to her free will without being coerced.
- Directness – voter votes in person.
- Generality – there are opportunities for the entire eligible citizenry to participate in voting.
- Uniformity – each eligible voter has one and only one vote.
- Availability – voting methods are available and accessible.

There is also an important meta-requirement of independent auditability in order to gain public trust in the electoral system and the results it produces.

Achieving all these targets at the same time is a non-trivial task, and particular implementations may bring along other requirements. For example, voter freedom is often implemented via secret vote casting. Generality and uniformity, in turn, translate into the ballot box and tally integrity requirements (the votes should not be undetectably added, deleted or modified). Thus, we get the classical information security CIA triad of

- *Confidentiality*, i.e. preventing the loss of voter privacy;
- *Integrity*, i.e. measures should be taken against vote deletion, altering or falsification;
- *Availability*, i.e. preventing selective or overall inaccessibility of the voting system.

The risks threatening these goals may be related to the voter, voting device, verification device, electronic identity system (ID-card or mobile-ID), or voting service, i.e. vote collecting servers or the vote counting system (Figure 4).

The voting and the verification device are asymmetric from the view-point of risks.

- Attacks against the voting device threaten the integrity, confidentiality, availability, and voting freedom.
- Attacks that only target the verification device threaten only confidentiality and public trust.

The goal of this work is to study whether m-voting with verifiable delivery can be implemented with security comparable to the security of the current solution (Sec. A.2). Thus, in this work, we only analyse the risks that differentiate the current solution from the possible m-voting solutions described in the current Chapter.

For example, we do not consider the attacks related to the vote counting system as they are exactly the same in the considered solutions (Section A.2).

A.6 M-voting specific risks

A.6.1 Classification of m-voting specific risks

There are nine types of risks that may differentiate m-voting solutions from the current solution (Figure 5).

1. *Voting device* risks, i.e. the mobile device specific risks related to the voting device (hardware, operating system, network connection, etc.).
2. *Voting application download* risks, i.e. the mobile device specific risks of downloading a faulty or fraudulent voting application.
3. *Vote transfer* risks, i.e. the mobile device specific integrity and confidentiality risks related to the vote casting protocol.
4. *Voting service* risks, i.e. complete compromise of the server side.[123]
5. *Voter* risks, i.e. the mobile device specific risks of influencing the voter considering that the voter may be easier to influence because voting is possible from any location.[124]
6. *Verifying device* risks, i.e. the risks considering a specific verification device, if it is not a mobile device, like in the current solution (Sec. A.2).
7. *Verifying application download* risks, i.e. the risks of downloading a faulty or fraudulent verification application, considering that the verification device is not a mobile device.
8. *Verification code transfer* risks.
9. *Vote download* risks, considering that the verifying device is not a mobile device.

A.6.2 Possible adversarial variations

In this analysis, we concentrate on the attacks against the technical system components (i.e. we leave out the voter risks from the list of Section A.6.1). This is motivated by the observation that the two verification solutions we compare provide a very similar user experience giving rise to the same attacks that the attacker can implement with a physical access to the voter (like looking over the shoulder, threatening, etc.). This also concerns the setups with the possible addition of a feedback channel.

[123]From the election result integrity point of view, the server side of the whole IVXV voting system is the key component. Because of this, a lot of effort has been put into building a number of safeguards into the server architecture. Most notably, a separate Registration Service has been introduced as a part of the server side system in order to account for possible misbehaviour of the vote collector component [HMVW16]. As a result, we estimate full server side compromise without it being caught by the auditing procedures to be an event of very low probability. However, for the purposes of the comparison between the current and proposed verification solution we will still consider this event, too.

[124]It can also be argued that greater choice for voting location may actually help the voter to escape coercion attempts more easily.

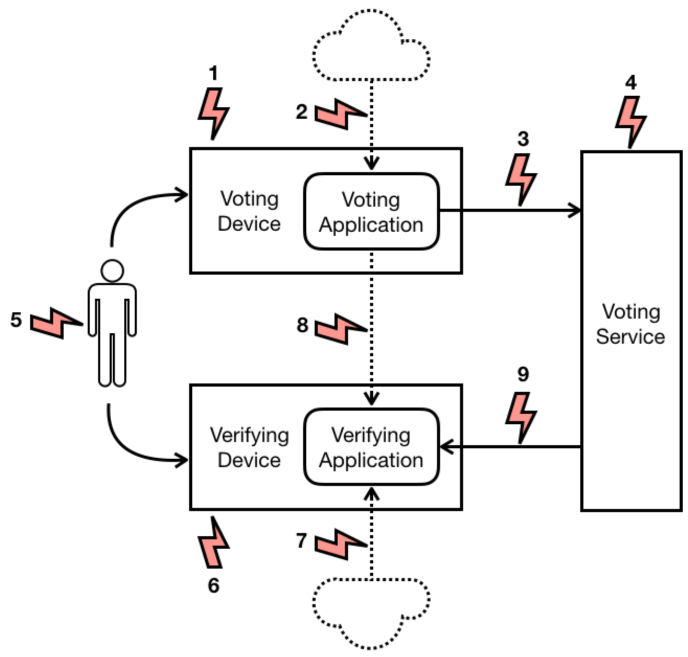


Figure 5. M-voting specific risks.

Thus, in this document, we consider eight possible adversarial situations depending on which components of the system are compromised (see Table 4).

Table 4. Possible adversarial configurations

Compromised components	Voting Device	Verifying Device	Server
Variation 0 (Figure 6 upper left)			
Variation 1 (Figure 6 upper right)			●
Variation 2 (Figure 6 lower left)		●	
Variation 3 (Figure 6 lower right)	●		
Variation 4 (Figure 7 upper left)		●	●
Variation 5 (Figure 7 upper right)	●		●
Variation 6 (Figure 7 lower left)	●	●	
Variation 7 (Figure 7 lower right)	●	●	●

We consider the following types of technical attack goals against the voting system:

- *Vote leakage*, where a vote will be revealed to unauthorised parties.
- *Vote deletion*, where a vote, though correctly created by the voter, will not reach the vote counting process, although the deletion may be detected by the voter via the verification mechanism.
- *Vote deletion without detection*, where vote can be deleted undetectably to the verification mechanism.
- *Vote forgery*, where the vote that reaches the vote counting process is not the intended vote, although the forgery may be detected by the voter via the verification mechanism.

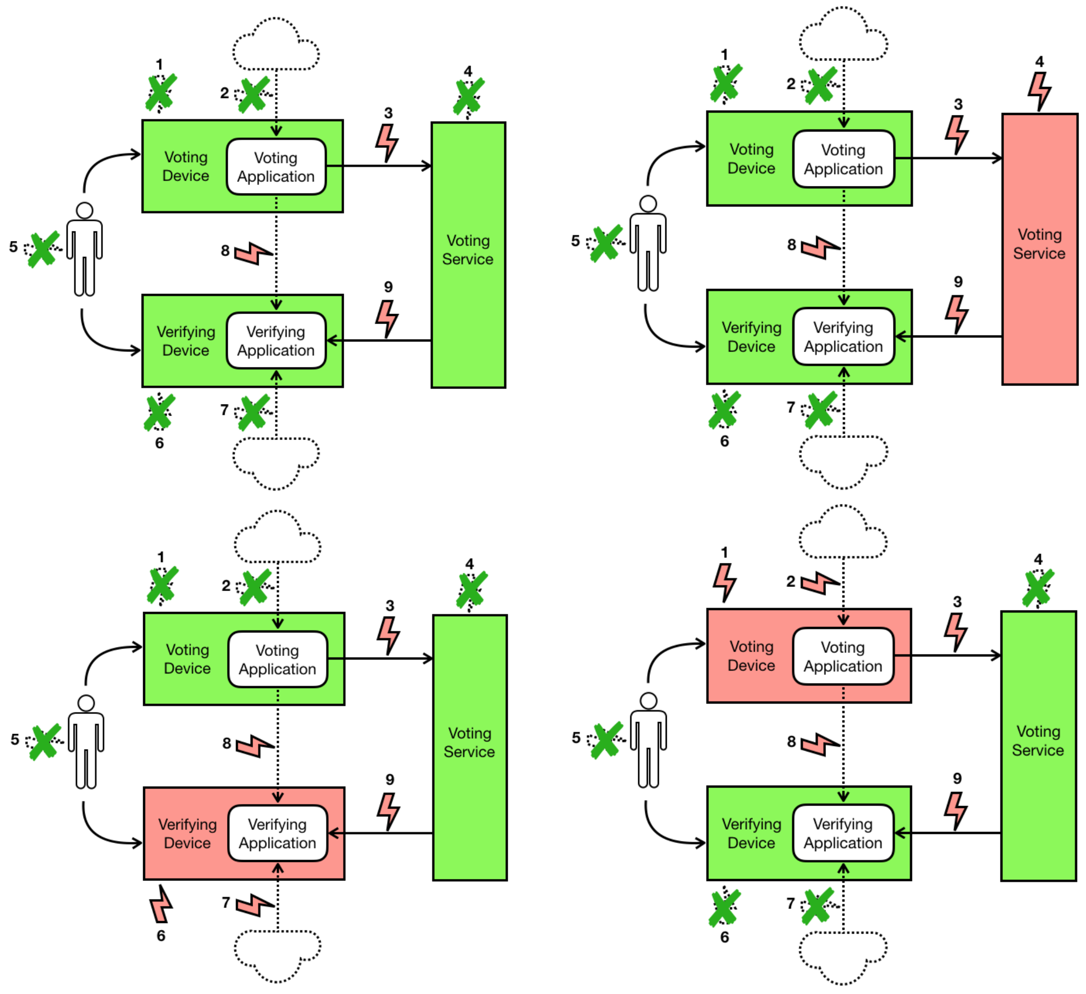


Figure 6. Variations 0 (upper left), 1 (upper right), 2 (lower left), and 3 (lower right).

- *Vote forgery without detection*, where vote can be forged undetectably to the verification mechanism. Malware that can cast a re-vote goes under this category.

In certain variations, in which too many components are compromised, it is impossible to prevent attacker reaching all these goals. It turns out that it only makes sense to study the variations 0-4, because in variations 5-7 there is no way of completely preventing any of the five attacking goals.

A.6.3 Comparison of the current solution with the proposed verification solution for m-voting

In this subsection, the current solution and the proposed m-voting verification solution are analysed based on the general scheme described on Figure 4. All possible and meaningful variations (0-4) are analysed and determined whether the attacks are prevented. Table 5 summarises the results of the analysis.

The conceptual security analysis under this framework does not depend on whether the voting and verification are browser-based or standalone application based.

Compared to the current solution with mobile-ID, the main difference of the proposed m-voting solution lies in the verification code transfer protocol. The proposed verification scheme is active,

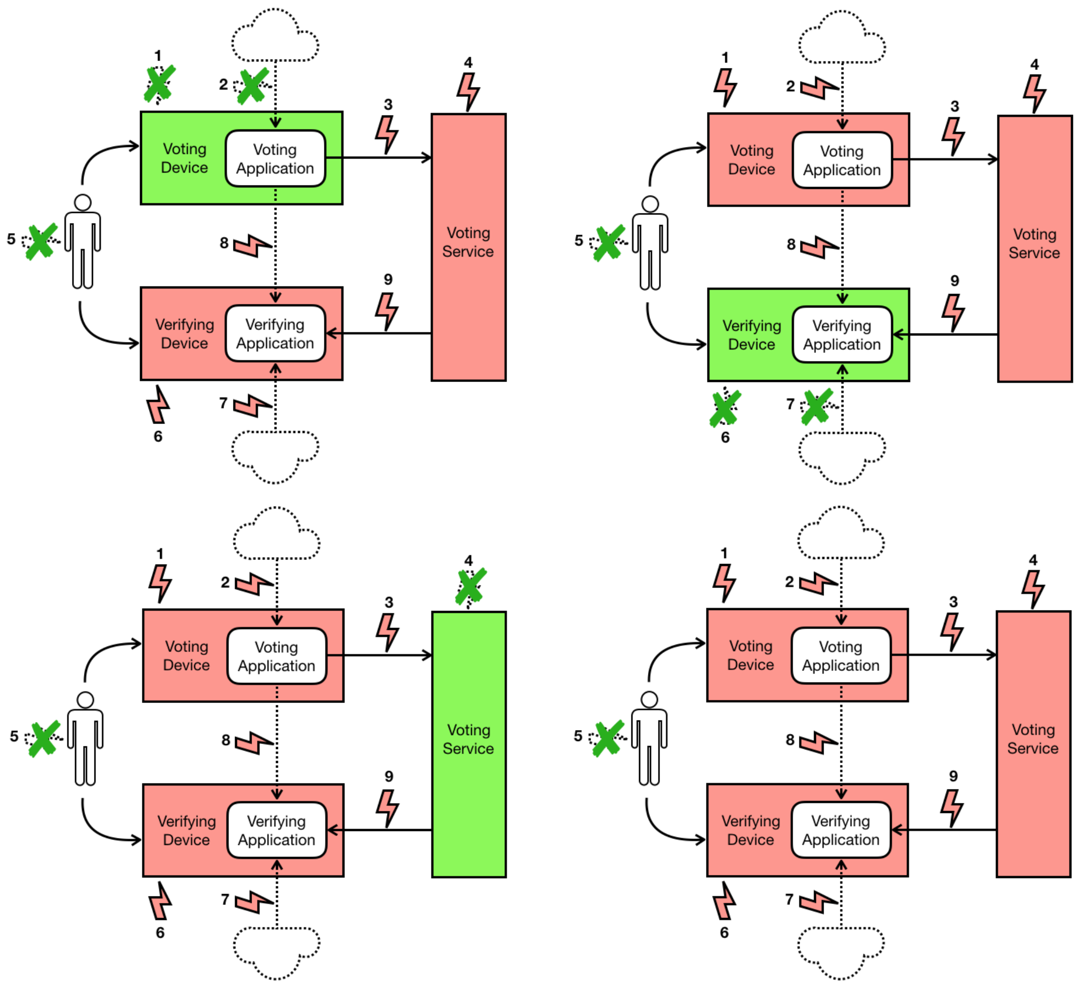


Figure 7. Variations 4 (upper left), 5 (upper right), 6 (lower left), and 7 (lower right).

meaning that the attacker who has control over the mobile device is able to see whether the voter verifies the vote. This may give an advantage to the attacker when the attacker wants to reduce the number of re-votes and thereby bypass the detection measure that is based on log analysis. In principle, the attacker who has infected the mobile device could prevent the signed vote from being sent to the server until it detects whether the voter tries to verify the vote. Of course, countermeasures may be found when specifying the verification protocol, but currently we have to consider such an attack.

In the following, the variations described by Table 4 and Figures 6,7 are reviewed according to the previously mentioned five types of technical attack goals.

A.6.3.1 Variation 0: None of the components are compromised

For this variation, the analysis result is the same for both the current system and the proposed m-voting verification system.

Vote deletion and forgery are prevented because once the vote reaches the vote collecting server, it will be processed correctly.

Vote leakage is prevented during transmission and processing because votes are encrypted (in a randomised way) with the vote encryption public key. In the current system, leakage during the verification code transfer is prevented because the transmission channel between the voting device and the verification device is close-range and optical (QR-code scan). In the proposed system, leakage during the verification code transfer is prevented because of the following reasons.

- Similarly to the current system, the transmission channel between the voting device and the verification device is close-range and optical (QR-code scan), and this guarantees authentic transmission of the encryption/MAC keys from the verifying device to the voting device.
- The verification code is encrypted/MACed with a fresh unique unpredictable encryption key.

A.6.3.2 Variation 1: Only service (vote collecting server) is compromised

For this variation, the analysis result is the same for both the current system and the proposed m-voting verification system.

Vote leakage is prevented for the same reasons as in Section A.6.3.1.

Vote deletion is not prevented in case the server side is compromised. However, it is possible to detect the vote deletion if at least one of the two key components (the vote collector or registration service) of the server side is honest. The auditors can verify the audit trail to check if any votes were removed on the server side.

Vote forgery without detection (and hence also *vote forgery*) is not prevented if mobile-ID is used for signing because a compromised vote collecting server can change the hash of the encrypted vote before signing and trick the user (pretending an error) to sign twice: one signature for the modified hash, another one for the unmodified hash. During verification, the fraudulent server presents the unmodified vote while the vote cast was the modified vote. Even if the voting application itself computes and displays the mobile-ID (four-digit) verification code, the fraudulent server may replace the vote hash in a way that the verification code of the forged vote remains the same as that of the original intended vote. For doing so, the fraudulent server has to perform about 10000 public key encryptions with different random strings, which may be feasible during a single voting protocol session. A possible mitigation for this problem would be sending the mobile-ID signature request directly to the DigiDoc Service without the mediation of the vote collecting server. However, the DigiDoc Service will be discontinued in October 2020, and replaced by mobile-ID REST API[125]. According to the new API description, the e-service provider (also known as relying party) calculates the verification code[126]. The voting system has to adapt the new API and redesign the way how the mobile-ID requests are handled. However, it is important to separate the roles of the vote collecting server and the mobile-ID relying party as otherwise the previously described attack remains possible.

A.6.3.3 Variation 2: Only the verifying device is compromised

For this variation, the analysis result is the same for both the current system and the proposed m-voting verification system.

[125]https://www.id.ee/index.php?id=30607&read=39500

[126]https://github.com/SK-EID/MID/blob/master/DDS-to-MID-migration/README.md

Vote leakage can not be prevented in case the verifying device is compromised as it can see the value of the vote.

Vote deletion and forgery are prevented because once the vote reaches the vote collecting server, it will be processed correctly.

A.6.3.4 Variation 3: Only the voting device is compromised

For this variation, the analysis result differs when considering the current system and the proposed m-voting verification system. The reason is that in the proposed verification solution the verification process is detectable by the malicious voting device.

Vote leakage can not be prevented in case the voting device is compromised as it can see the value of the vote. This is the same for both the current system and for the proposed m-voting system.

Vote deletion without detection can be prevented in the current system using the individual verification mechanism. In general, when a sufficiently large random subset of the voters verifies their votes, an attacker is not able to predict how the voter will behave, and a large-scale vote manipulation attack will be detected with high probability. However, malware working on a compromised voting device could prevent the voter from verifying, e.g., by not displaying a valid QR-code or making the voting application crash at a suitable time. In that case the voter must report the failure to verify. When the voter is prevented from verifying the vote, it can not be checked if the vote was changed or sent to the vote collection server. When comparing the current verification system with the proposed verification mechanism for m-voting, the latter is active, meaning that malware on the mobile device is able to detect if a voter is going to verify the vote. Thus, such malware could dynamically decide which voters to attack depending on whether they start with the verification process. For example, malware can delay the delivery of the ballot and wait to see if the voting application is closed right after the vote has been cast in the user interface of the voting application. In that case it is unlikely that the voter verified the vote and thus malware can drop the vote without the voter noticing it. This kind of an attack could be prevented by introducing a feedback mechanism which notifies the voter once a vote has been successfully cast. This mitigation measure is discussed in Sections 3.3 and 3.6.

Vote forgery can be detected only in some cases when the voting device in compromised. For example, the vote verification application can detect if the voter's intended vote does not reach the vote collection server. Additionally, by modifying data during the verification code transfer, a fraudulent voting device cannot make a forged vote verify as the intended vote, neither can it conceal the fact that the vote was deleted and hence not cast as this is detected during verification. However, the verification application does not reveal if a re-vote was cast after the initial vote by a malicious voting device. With the current configuration of the verification system, such an attack is not detected even if the re-vote is cast instantly after the original vote. The possibility for malware to re-vote depends on how the voter handles her e-ID and which e-IDs can be used in the voting process. Thus, the current verification system can not completely mitigate the risk of malicious re-voting.

When comparing the current verification system to the one currently proposed for m-voting, the issue of active verification also affects vote forgery. In the proposed verification system, the voting device has to scan a QR-code in order to access transport keys required by the verification system. As such behaviour can be detected by malware, it could launch an attack based on voters behaviour. Compared to vote deletion, it is more difficult to hide vote forgery as the choice whether to replace the candidate who the voter wanted to vote for has to be done before the malware can detect whether the voter is going to use verification. Thus, the malware would have to always prevent the voter from verifying the vote, which is likely to be detected.

One way to prevent malicious re-voting is to only use smart card readers with PIN-pads and PIN-firewalls. Unfortunately, the number of voters who use PIN-pad based card readers is not significant. In addition, it could be possible to require multiple digital signatures for casting a vote to make it more difficult for malware to access multiple e-IDs. To detect stealthy re-voting malware, it is possible to implement a feedback channel for the voter. Also the semantics of verification can be augmented to include freshness information. These and other mitigation ideas are described in Section 3.6.

A.6.3.5 Variation 4: Verifying device and the service are compromised

For this variation, the analysis result is the same for both the current system and the proposed m-voting verification system.

Vote leakage can not be prevented in case the verifying device is compromised as it can see the value of the vote.

Vote deletion is not prevented in case the vote collector service component on the server side is compromised.

Vote deletion without detection is possible only if all of the independent key components on the server side are compromised.

In such a case the individual verification mechanism may not work correctly as it can collude with the malicious server. However, if not all of the components on the server side are malicious, the auditors can verify the audit trail to check if any votes were removed.

Vote forgery without detection (and hence also *vote forgery*) is not prevented if mobile-ID is used for signing because a compromised vote collecting server can change the hash of the encrypted vote before signing as described in Section A.6.3.2. This threat does not apply when ID-card is used for signing the vote.

Table 5. Summary of the analysis. The table shows the adversarial activities prevented in the current system and in the proposed m-voting solution.

Preventable Adversarial Activity	**Var 0**	**Var 1**	**Var 2**	**Var 3**	**Var 4**	Var 5	Var 6	Var 7
Vote leakage	●	●						
Vote deletion	●		●					
Vote deletion without detection	●	◐	●	◑	◐			
Vote forgery	●	◐	●		◐			
Vote forgery without detection	●	◐	●	◐	◐			

● = is prevented ◐ = is partially prevented ◑ = prevented only in current system

A.6.4 Conclusions of comparison

The comparison between the currently used verification solution and the solution described in Section A.3 brings out only one new risk which is related to Variation 3. As the new verification solution is active, an attacker who has control over the voting device may be able to detect whether the voter verifies the vote. This may allow the attacker to dynamically drop votes for those voters who are likely not to use verification. A partial mitigation for this issue is to provide both the current verification solution along with the new verification method. The most a malware application could attempt in this case is to use the camera on the mobile device attempting to detect if verification QR-code is being scanned by another device.

Additionally, vote dropping attacks that are either based on preventing voters from verifying or detecting whether the voters skip verification can be mitigated by introducing a feedback channel for the voters. In case the feedback channel notifies the voter of a successfully cast vote, the voter can detect vote dropping attacks even without using the verification system. The mitigation measure that is based on a feedback channel is discussed in Sections 3.3 and 3.6.

Other analysed threats are the same for both solutions and thus the proposed m-voting verification solution does not introduce other significant attack vectors compared to the current one. Although the attack vectors are largely the same, the difference between the current solution and the proposed solution lies is in the probabilities of successfully using the attack vectors. Finding these probabilities is a non-trivial task which comes down to analysing the security aspects of the used voting devices.

It is important to note that currently the mobile-ID signature request is sent to the vote collection server, which is not optimal as it makes the vote collection server a single point of failure. A single component on the server side should not be able to influence the integrity of the votes. By mediating mobile-ID requests via the collector service, the verification system is weakened as described in Section A.6.3.2. As a mitigation, the mobile-ID requests should be sent to an independent entity (also known as relying party according to the new mobile-ID API[127]), which is not under the control of the vote collection server. This would prevent the vote collection server from easily colluding with the mobile-ID request mediator and thereby prevent weakening of the verification system.

[127] `https://github.com/SK-EID/MID#1-introduction`

B Check-list for security audit

We propose a set of properties that should be verified during a security audit[128] for any voting application implementation. We additionally give a short self-assessment for all the proposed items for standalone PC, native mobile and browser based voting applications.

The randomness used for vote encryption has high entropy and is unpredictable.

PC: Voting application conforms. OpenSSL library is used for acquiring randomness via cryptographically suitable operating system interfaces.

Native mobile: Voting application conforms. Both Android and iOS platforms provide API for cryptographically secure randomness.

Browser app: Voting application conforms under the condition that a browser compliant with W3C Web Cryptography API is used.

The candidate and party name that gets encrypted corresponds to the voter's selection.

Voting application conforms on all platforms. Voters can additionally convince themselves via the verification tool.

The sensitive data (QR-code and its values, voter's choice, PIN codes) is not stored in the persistent memory.

PC: Voting application conforms. However, this requires self audit and tool help (such as ifdef-statements) in the development, as development versions of the application might use filesystem to save data for debugging.

Native mobile: Voting application conforms. However, in addition to the standard care with development versions of the app, one has to take into account different application lifecycle on Android and iOS.

Browser app: Voting application conforms. However, self audit is required to exclude the usage of local storage.

The sensitive data is not leaked outside the device and the best practices are applied to clear the memory as soon as possible.

PC: Voting application conforms. With C/C++ memory model it is possible to clean and release memory.

Native mobile: This depends on the choice of tools. Java uses garbage collection, Swift reference counting. For the sensitive data it is best to have handlers that overwrite it, once it is not needed any more, meaning that immutable objects such as String (Java) should not be used for e.g. plain-text ballots.

Browser app: JavaScript uses garbage collection and immutable objects that cannot be used for e.g. plain-text ballots. The best option is to close the browser window used for voting right in the end.

[128] We acknowledge Arnis Paršovs for giving valuable input to the process of developing these assumptions.

The app is compiled applying the best practices available for the target OS (enabling Address Space Layout Randomisation (ASLR) and Data Execution Prevention (DEP), stripping debugging symbols).
Such a list of best practices would have to be compiled, maintained and applied for each supported voting application platform.

The app has vote collector's TLS public key hardcoded and the app verifies it when connecting to the vote collector.
PC: Voting application conforms. TLS pinning is used, system cert-stores are not used for TLS. Certificates are configurable by election organizer.

Native mobile: Voting application conforms. On both major platforms it is possible to use certificate pinning and bypass central truststores.

Browser app: Voting application does not conform, this feature is not available for browser based apps.

The app enforces the hardcoded TLS cipher suites when connecting to the vote collector.
PC and native mobile: Voting application conforms. TLS cipher suite is configurable by election organizer.

Browser app: Voting application cannot control the TLS implementation. Server-side enforcing of TLS cipher suites is possible, but it may introduce compatibility issues.

The signed vote returned from the vote collector is correctly verified (signature, certificate chain, OCSP response, vote registration signature).
PC, native mobile and browser app: Voting application conforms. Audit of the correctness of implementation of cryptographic primitives and protocols must be carried out.

The private keys used to sign binaries / credentials used to publish packages are well protected.
This check is not to be carried out by the voting application, but the surrounding environment.

PC and native mobile: The private keys should be stored in a hardware token. An auditable procedure to ensure their authorized use of both keys and credentials should be defined.

Browser app: The check can not be implemented – there is no way to sign a browser based voting application.

The binaries published are compiled from the audited source code.
PC: In order to verify, the source code must be available to the auditor and the a compilation would have to take place in an audited environment or there would have to be a reproducible build scenario, which for C/C++ currently does not exist.

Native mobile: In case reproducible build scenarios are created, the voting application conforms given that the auditor has access to the source code. Note that the procedure is easier to define for Android platform.

Browser app: Derivation of the application bundle from JavaScript sources is reproducible.

The signed hashes of the voting app published by ESEO matches the binaries distributed on `https://www.valimised.ee`.
PC: This can be verified at each election event.

Native mobile: Signing of the hashes does not carry any meaning for iOS apps, as voters have no way to verify these against IPA files. In case of Android, there is theoretical possibility to verify APK files against checksums.

Browser app: This may be the only way to actually verify the integrity of a JavaScript application.

Only the official app is distributed on the app stores and the information to identify it is distributed via authentic channels.

PC: Not applicable for current voting application, other than monitoring the potential occurrence of bogus applications.

Native mobile: There is a risk of bogus applications on both Google Play and Apple App Store. A recent study was able to find 2,040 potential counterfeits that contain malware in a set of 49,608 apps that showed high similarity to one of the top-10,000 popular apps in Google Play Store [RKG+19]. Counterfeit apps are also found in Apple App Store.

Browser app: There is a risk of bogus websites mimicking the look of an official voting application.

C Resources required for mobile voting application development

We have identified the following candidate solutions for mobile voting application:

- platform specific mobile app developed with native tools;
- platform specific mobile app developed with cross-platform tools;
- browser based web-app.

We shall estimate the required development resources in person-hours.

We assume that iterative-incremental approach to software development is taken. We define following major increments.

- Minimum Viable Product (MVP) – accessible and user-friendly voting application usable on mobile device(s), supporting mobile-ID.
- Extended Product – Minimum Viable Product extended with graphical configuration tools for ESEO.
- Full Product – Extended Product with ID-card support.

The structure of the project may vary, but the following roles shall be present:

- Project Manager – responsible for project communication, iteration plans and management;
- Software Analyst – responsible for gathering and documenting the requirements, defining acceptance testing and creating documentation;
- Software Architect – responsible for the technical solution, lead developer with expert knowledge in the platform, good knowledge in e-ID and TLS, participates in code reviews;
- Software Developer – responsible for implementing specific parts of the application, good knowledge in the platform, participates in code reviews;
- Security Engineer – responsible for correct implementation of cryptographic algorithms, participates in code reviews;
- Quality Assurance Engineer – responsible for implementing fully automatic integration tests;
- Usability Expert – responsible for the user experience design;
- Accessibility Expert – responsible for the knowledge transfer and accessibility testing;
- DevOps Engineer – responsible for maintaining development infrastructure and publishing the applications.

In our calculations, we do not take potential role overlaps into account, although DevOps role could be fulfilled by e.g. a Quality Assurance Engineer, etc. We assume that all the code is subject to automated test coverage of 80%, all code must be reviewed, up-to-date specification and documentation has to be provided at the end of the development project.

The development of a single native mobile voting application is divided into the following work packages.

- WP: Project Management (MVP)
  - Iteration Plan and Management
- WP: Requirements and Architecture (MVP)
  - Requirements Model
  - Architecture Notebook
  - Documentation
- WP: Accessible User Experience (MVP)
  - User Experience Prototyping
  - User Interface Development
  - Accessibility
- WP: Election Organizer Tools (Extended Product)
  - App Configuration Tool
  - App Configuration Interface
- WP: App Development (MVP)
  - TLS with Collection
  - Communication with Collection
  - App Framework
- WP: Digital Signature and Encryption (MVP)
  - Create digital signature
  - Verify digital signature
  - Encrypt ballot
  - Verifiability
- WP: e-ID, ID-card (Full Product)
  - ID-card signature
  - ID-card authentication
- WP: e-ID, mobile-ID (MVP)
  - mobile-ID signature
  - mobile-ID authentication
- WP: Quality Assurance and Deployment (MVP)

We have estimated all features and work-packages. For development features the following work breakdown structure has been used.

- Specify feature (by Software Architect)

- Design and implement feature (by Developer)
- Create unit tests (by Developer)
- Perform code review (by second developer)
- Create integration tests (by Quality Assurance Engineer)
- Update technical documentation (by Developer)

We estimate that a native voting application for one platform can be developed within the following timeline (not taking into account vacations).

- Project start – month 1.
- Architecture Outline exists – end of month 1.
- UX Prototype exists – mid-month 3.
- Minimum Viable Product exists – end of month 5.
- Extended Product exists – end of month 8.
- Full Product exists – end of month 11.

We estimate that the effort to develop a native voting application for one platform is the following.

- Minimum Viable Product – 5000 person-hours, deadline mid-month 7.
- Extended Product – 6000 person-hours, deadline mid-month 9.
- Full Product – 7000 person-hours, deadline end of month 11.

We estimate that the cost of developing native applications for both iOS and Android at the same time, using the same project team and framework, does not double the time and efforts. Hereby we assume that there is expert level of knowledge about both platforms in the team and there exists one additional person for the developer role. The required efforts are estimated to be the following.

- Minimum Viable Product – 8000 person-hours, deadline mid-month 10.
- Extended Product – 10000 person-hours, deadline mid-month 12.
- Full Product – 11000 person-hours, deadline mid-month 14.

We note that the Full Product working on two platforms requires additional 4000 person-hours to develop. If we take into account those aspects where cross-platform development could help, we can estimate the Full Product for both platforms at 9000 person-hours by deadline mid-month 12. This means cost saving of 2000 person-hours. However, we suspect that this saving could efficiently be cancelled out by added complexity of cross-platform development.

Note also that it may turn out to be necessary to develop other components besides the voting application. E.g. several of the potential new mitigation measures listed in Section 3.6 presume a significant amount of development. The scope and extent of the required development are too unclear at this point to give reliable estimates. The respective analysis needs to be carried out separately.

Index of acronyms

Bibliography

[BDPA10] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge-Based Pseudo-Random Number Generators. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2010.

[BGS+16] Christian Bull, Kristian Gjøsteen, Katrine Stemland Skjelsvik, Ida Sofie, and Gebhardt Stenerud. The imperfect storm. *E-Vote-ID 2016 Proceedings*, pages 116–120, 2016.

[BHL+] Ahto Buldas, Kristo Heero, Peeter Laud, Riivo Talviste, and Jan Willemson. Cryptographic algorithms lifecycle report 2016. Cybernetica research report A-101-3, `https://www.ria.ee/sites/default/files/content-editors/publikatsioonid/cryptographic_algorithms_lifecycle_report_2016.pdf`.

[BKLO17] Ahto Buldas, Aivo Kalu, Peeter Laud, and Mart Oruaas. Server-Supported RSA Signatures for Mobile Devices. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*, volume 10492 of *Lecture Notes in Computer Science*, pages 315–333. Springer, 2017.

[CGG16] Jordi Cucurull, Sandra Guasch, and David Galindo. Transitioning to a Javascript Voting Client for Remote Online Voting. In Christian Callegari, Marten van Sinderen, Panagiotis G. Sarigiannidis, Pierangela Samarati, Enrique Cabello, Pascal Lorenz, and Mohammad S. Obaidat, editors, *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 4: SECRYPT, Lisbon, Portugal, July 26-28, 2016*, pages 121–132. SciTePress, 2016.

[Cor20] Sylvie Corbet. France Holds Local Elections Despite COVID-19 Outbreak Fears. *Time*, March 2020. `https://time.com/5803469/france-local-elections-coronavirus/`.

[DMS+17] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J. Halderman, and Vern Paxson. The security impact of https interception. 01 2017.

[Gre19] Matthew Green. How safe is Apple's Safe Browsing?, 2019. `https://blog.cryptographyengineering.com/2019/10/13/dear-apple-safe-browsing-might-not-be-that-safe/`.

[HLW11] Sven Heiberg, Peeter Laud, and Jan Willemson. The Application of I-Voting for Estonian Parliamentary Elections of 2011. In Aggelos Kiayias and Helger Lipmaa, editors, *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers*, volume 7187 of *Lecture Notes in Computer Science*, pages 208–223. Springer, 2011.

[HMVW16] 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, Bregenz, Austria, October 18-21, 2016, Proceedings*, volume 10141 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 2016.

[HPW15] Sven Heiberg, Arnis Parsovs, and Jan Willemson. Log Analysis of Estonian Internet Voting 2013-2015, 2015. IACR Cryptology ePrint Archive, `http://eprint.iacr.org/2015/1211`.

[HT15] J. Alex Halderman and Vanessa Teague. The New South Wales iVote System: Security Failures and Verification Flaws in a Live Online Election. In Rolf Haenni, Reto E. Koenig, and Douglas Wikström, editors, *E-Voting and Identity - 5th International Conference, VoteID 2015, Bern, Switzerland, September 2-4, 2015, Proceedings*, volume 9269 of *Lecture Notes in Computer Science*, pages 35–53. Springer, 2015.

[HW14a] Sven Heiberg and Jan Willemson. Modeling Threats of a Voting Method. In Dimitrios Zissis and Dimitrios Lekkas, editors, *Design, Development, and Use of Secure Electronic Voting Systems*, pages 128–148. IGI Global, 2014.

[HW14b] Sven Heiberg and Jan Willemson. Verifiable internet voting in Estonia. In Robert Krimmer and Melanie Volkamer, editors, *6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014, Lochau / Bregenz, Austria, October 29-31, 2014*, pages 1–8. IEEE, 2014.

[KFW20] Krisjan Krips, Valeh Farzaliyev, and Jan Willemson. Developing a Personal Voting Machine for the Estonian Internet Voting System, 2020. Submitted to ARES 2020.

[KKW18] Kristjan Krips, Ivo Kubjas, and Jan Willemson. An Internet Voting Protocol with Distributed Verification Receipt Generation. *E-Vote-ID 2018*, page 128, 2018.

[Lei20] Douglas J. Leith. Web browser privacy: What do browsers say when they phone home? SCSS Technical Report, 24th Feb 2020, 2020.

[RKG$^+$19] Jathushan Rajasegaran, Naveen Karunanayake, Ashanie Gunathillake, Suranga Seneviratne, and Guillaume Jourjon. A Multi-modal Neural Embeddings Approach for Detecting Mobile Counterfeit Apps. In Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia, editors, *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 3165–3171. ACM, 2019.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[SAL09] Marian Sawer, Norman Abjorensen, and Philip Larkin. *Australia: The state of democracy*. Federation Press, 2009.

[SCJ18] David Salvador, Jordi Cucurull, and Pau Julià. wraudit: A Tool to Transparently Monitor Web Resources' Integrity. In Adrian Groza and Rajendra Prasath, editors, *Mining Intelligence and Knowledge Exploration - 6th International Conference, MIKE 2018, Cluj-Napoca, Romania, December 20-22, 2018, Proceedings*, volume 11308 of *Lecture Notes in Computer Science*, pages 239–247. Springer, 2018.

[SKMR20] Gian Luca Scoccia, Ibrahim Kanj, Ivano Malavolta, and Kaveh Razavi. Leave my apps alone! a study on how android developers access installed apps on user's device. In *Proceedings of the 7th IEEE/ACM International Conference on Mobile Software Engineering and Systems*, 2020.

[SKW20] Michael A Specter, James Koppel, and Daniel Weitzner. The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in US Federal Elections. 2020.

[ST03] Adi Shamir and Eran Tromer. On the Cost of Factoring RSA-1024. *RSA CryptoBytes*, 6:10–19, 2003.

[UNM16] International migration report 2015, highlights, 2016. The Department of Economic and Social Affairs of the United Nations Secretariat, `https://www.un.org/en/development/desa/population/migration/publications/migrationreport/docs/MigrationReport2015_Highlights.pdf`.