Mobile Voting – Still Too Risky?

Sven Heiberg[1], Kristjan Krips[2,3], and Jan Willemson[3,4](✉)

[1] Smartmatic-Cybernetica Centre of Excellence for Internet Voting, Tartu, Estonia
[2] Institute of Computer Science, University of Tartu, Tartu, Estonia
[3] Cybernetica, Tartu, Estonia
[4] Software Technology and Applications Competence Center, Tartu, Estonia

Abstract. This paper studies the challenges of creating a mobile device based voting client. We discuss the issues related to standalone and mobile browser based voting applications. In both cases we discuss the problems of vote privacy, integrity and voting channel availability. We conclude that neither of the options can currently achieve the level of security PC-based voting clients can provide, with the attack surface being larger in the case of mobile browser based voting application.

1 Introduction

Voting is the core method of delegating public power in modern democratic societies. However, in the contemporary increasingly mobile world, relying only on polling stations to vote is less and less of an option.

Physical polling station based elections were put into a completely new light due to the SARS-CoV-2 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 [4]. A straightforward alternative is postal voting (that was heavily used e.g. in 2020 US presidential elections).

History of postal voting goes back a long time. Voters in the Swiss canton of St. Gallen are reported as having sent their ballots via mail as early as 1673; however, postal voting became more widely used only in late 19th/early 20th century [17]. Still, this method is not perfect. It is hard to remotely authenticate the voter, reliability of postal services varies a lot across the world, there are no good measures against vote selling, etc. Even the international postal service can have severe disruptions as illustrated by the US Postal service temporarily stopping its service to 102 countries during the pandemic in spring 2020[1]. Thus, it is important to study 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

[1] https://newjerseyglobe.com/campaigns/with-102-countries-not-receiving-mail-from-u-s-military-and-overseas-voters-might-not-get-primary-election-ballots/.

M. Bernhard et al. (Eds.): FC 2021 Workshops, LNCS 12676, pp. 263–278, 2021.
https://doi.org/10.1007/978-3-662-63958-0_23

out in several parts of the World (e.g. Estonia, Norway, Switzerland, Australia, etc.) using either a PC- or browser-based voting client software.

In addition, mobile technology has been used in several countries for the purposes of democratic inclusion (e.g. campaigning and polling) since early 2000s [12]. The rise in mobile platforms usage has put forward a natural question whether casting votes from mobile devices would be a viable option.

For example, in a number of African countries more people have mobile phones than access to continuous electric supply [27]. At the same time, long distances and poor transportation make remote voting a necessity in order to achieve universal suffrage. Thus, the options for mobile voting have been studied in Africa for a decade already [6,7,15]. However, the security issues of smartphone based voting clients haven't gotten much attention by the research community.

The paper is organised as follows. Section 2 gives an overview of the used methodology. Next, Sect. 3, describes the issues of browser based voting. It is followed by Sect. 4, which highlights the issues faced by a standalone voting application on the two most popular mobile platforms – iOS and Android. Finally, Sect. 5 offers some discussion and conclusions.

2 Methodology

Voting software has to adhere to the security principles of the corresponding election system and jurisdiction. As a result, the list of requirements set for the elections varies a bit between different authors and sources (see e.g. [3,10,21,25]).

In this paper we concentrate on the client side security of remote electronic voting, hence we will focus on the following subset of general requirements.

1. **Vote privacy** is a measure to guarantee the freedom of choice.
2. **Vote integrity** in our setting means that the vote should reach the (digital) ballot box in a way that corresponds to the voter's real preference.
3. **Availability** means that voters should have access to the voting methods and channels to guarantee both uniformity and generality of elections.

Some aspects remain outside of the scope of the current paper, most notably tally integrity and verifiability issues. Eligibility verification and guaranteeing uniformity are typically solved via voter authentication, and this is a vast area of research on its own. Due to space limitations, we also leave mobile authentication out of scope of the current paper.

To further focus our field of interest, we will concentrate on solutions where the mobile device is used to directly implement the standard voting workflow including proving eligibility, presenting the candidates, recording the voter preference, submitting it and presenting any information required for individual verification of the vote. Under the hood, the voting software must also perform security-related operations like establishing authenticated, secrecy-and-integrity-protected channel to the server, protecting the vote from manipulation etc.

We note that in our setting, the voting application participates in the cryptographic protocol as a voter's tool to cast her preference as a vote, and to submit it into the digital ballot-box. This is different from e.g. code voting [11], where the voting application is a mere transparent transportation channel of anonymous pre-encrypted data. As a consequence, the client application learns both voter's identity and her preferences, and is in the position to change the vote. This problem can be mitigated by the voting protocol providing verifiability.

Given these limitations, we study two main alternative architectural solutions – implementing the voting client based on a browser platform, or as a standalone mobile application. In both cases, we will take a more detailed look at how the three main requirements identified above can be satisfied, and what are the residual risks given today's level of mobile platforms and their security features.

3 Issues with Browser Based Voting

We consider the voting application to be implemented as a dynamic webpage using HTML, JavaScript and CSS. The application is deployed to a web server and the URL is published. Voters use browsers of their choice to visit the webpage.

Browser is responsible for downloading the application files and creating a representation of the Document Object Model (DOM), which is made available to JavaScript via standardised APIs. In the voting workflow, the voter's actions activate JavaScript implementation of the protocol, which affect the DOM, changing what is rendered to the voter. The voting application is executed in the browser context and has limited access to the system level APIs.

While standalone applications can be analysed relative to the OS they are running on, web applications are not tied to a single platform. Thus, the security and privacy aspects of both mobile and desktop browsers have to be reviewed.

Usability. In the context of voting, the lack of screen size creates challenges for designing the user interface. For example, a desktop browser may display the full list of candidates, while a mobile version can not. Thus, a question arises which candidates should have the advantage of being displayed first.

The limitations on screen size have prevented mobile browsers from fully adapting some standard functionalities of their desktop counterparts. For example, it is common for mobile browsers to optimize the way how URL and URL bar are displayed. In addition, mobile browsers usually do not allow the users to view detailed information about the TLS connection. Such optimizations increase the risk of voters being tricked by a phishing website.

Mobile Browser Issues. Distributing the voting client as a standard web application has the potential to simplify deployment to both desktop and mobile platforms, unify user interface, and provide an option for easy hot-fixing. At the same time, this introduces a middleman (browser) between the voting application and the operating system, vastly increasing the attack surface.

Luo, Nikiforakis *et al.* published two longitudinal studies on security features of mobile browsers [19,20]. They showed that, in general, mobile browsers lag behind their desktop counterparts in the adoption of new security features.

Thus, web applications used via mobile browsers are more vulnerable to a variety of attacks (e.g. phishing, malicious scripts, etc.). In the context of mobile voting this may create issues related with both ballot privacy and ballot integrity.

3.1 Privacy

Sandboxing. The voting application has to be isolated from other web pages and third party software. Browsers isolate content from different origins with the help of Same-Origin Policy (SOP). However, vulnerabilities can be used to bypass SOP[2]. Thus, some desktop browsers provide additional sandboxing by launching different web sites in different processes. Google Chrome does this since version 67 by relying on Site Isolation, which provides protection against multiple classes of attacks [23]. This feature was added to Android based Google Chrome for devices that have at least 2 GB of RAM, but it only works on web sites where users enter passwords[3]. While Firefox does not yet have a similar functionality, a project to add the feature is ongoing[4].

Even the extra measures do not provide absolute isolation. For example, malicious browser extensions could violate ballot privacy by reading page contents, as described in Sect. 3.2.

Telemetry. A 2020 report by Leith gave an overview of the telemetry mainstream browsers send back to the vendor or third parties [18]. 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 all sent information about the query that was typed to the URL bar to their respective vendors.

Google Chrome, Firefox and Safari rely on Google's Safe Browsing service to identify phishing sites and malware, while Microsoft Edge relies on Microsoft Defender SmartScreen for the same features. While Google's Safe Browsing uses k-anonymity mechanism and local filtering, Microsoft's SmartScreen uses local filtering only for top traffic sites and known phishing sites. The rest of the URL-s are sent to be analysed by the SmartScreen service[5].

In the context of voting, even the leakage of indirect information about the voting habits of users may cause security issues. This kind of information could be used in coercion attacks to detect whether a voter has abstained or re-voted.

Metadata about voters' behaviour is also visible to Internet service and DNS providers. Such data could be used to list eligible voters who abstain from voting. The list could be used for targeted attacks in order to submit votes on behalf of

[2] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Same+Origin+Policy+bypass.
[3] https://www.chromium.org/Home/chromium-security/site-isolation.
[4] https://wiki.mozilla.org/Project_Fission.
[5] https://docs.microsoft.com/en-us/microsoft-edge/privacy-whitepaper.

the abstained voters. This threat could be mitigated if all eligible voters would get notifications when a vote has been submitted on their behalf. However, such notifications would have organisational and privacy issues of their own.

Man-in-the-Middle Attacks. Real world examples show that TLS is often the only defensive measure that protects submitted ballots [2]. In case of a standalone voting application, the TLS certificate of the voting server can be *pinned*, i.e. the application can be built to accept a limited set of certificates. To bypass pinning, either the application or the operating system would have to be modified.

Mainstream browsers no longer support HTTP Public Key Pinning (HPKP). As a replacement, it is recommended to rely on Certificate Transparency (CT)[6] and DNS Certificate Authority Authorization (CAA)[7]. However, CT is not mandatory for locally issued certificates.

Thus, there is no straightforward way for pinning certificates for a browser based voting application. This makes it difficult to prevent man-in-the-middle (MITM) attacks by corporate middleboxes and anti-viruses that monitor the messages exchanged inside TLS sessions. A study published in 2017 found that such interception rate is in the range of 4–11% [5]. In 2019, Cloudflare reported that this rate has significantly increased since the 2017 study was published[8].

TLS traffic is often monitored via a local MITM approach, affecting the security guarantees provided by HTTPS [5]. Thus, in addition to the privacy risks, there are also risks to the integrity of the vote.

A voter is also able to set up a proxy to intercept and modify API calls. Thus, when designing API-s, it should be assumed that the queries may be intercepted.

3.2 Integrity

One of the main issues with online voting is the untrustworthiness of voter's device, which can affect the integrity of the ballot. The risks are even higher when votes are cast through a web application. Browsers introduce an extra layer that has to be protected in addition to the rest of the operating system.

Fortunately, security of mainstream browsers has improved over the years [19]. While zero-day vulnerabilities can not be ruled out, delivering patches to the end users is easy due to the browsers getting automatic updates either directly from the vendors or via the operating system. This holds both for mainstream desktop browsers and for mobile browsers running on iOS and Android.

Web Application Integrity. While the identity of a website can be verified by inspecting its certificate, there is no assurance that the web content distributed by the server is the one that was deployed by the election organizer. There are no straightforward ways for voters to verify integrity of the JavaScript

[6] https://tools.ietf.org/html/rfc6962.
[7] https://tools.ietf.org/html/rfc8659.
[8] https://blog.cloudflare.com/monsters-in-the-middleboxes/.

code. However, there are some mechanisms that provide partial solutions to the problem.

Scytl has proposed a tool (`wraudit`) to remotely verify that a proper web-application is distributed by the web server [24]. However, it does not prevent local tampering, and the voter can still access a tampered with web application.

W3C recommendation of Subresource Integrity (SRI)[9] defines a checksum-based mechanism by which user agents may verify that a fetched resource has been delivered without unexpected manipulation. However, SRI does not protect against tampering caused by a compromised web server or a MITM attack.

Cross-Origin Resource Sharing (CORS)[10] allows web servers to tell browsers which resources can be shared cross-origin. A voting server could inform browsers that it only accepts requests from the official voting web page. This would prevent a fake voting application from sending messages to the real back-end. The restriction could be circumvented by also using a fake back-end, either by proxying queries to the true back-end or disenfranchising voters completely.

Browser Extensions. Browser behaviour can be modified by extensions that may be able to interfere with web applications' control flow and visuals. Most mobile browsers do not support extensions, but there are a few exceptions[11].

Extensions can both read and modify the content of websites via DOM. There are some methods to ensure DOM integrity[12], but this far they are still experimental.

The current permission model for mainstream browsers allows extensions to ask for far-reaching access. When installing an extension, the user has to either accept the requested permissions, or retract from installing it altogether. While Firefox does not allow to turn off permissions, Google Chrome lets the user to configure on which web sites certain permissions may be used. However, the option may not be available for extensions installed using an enterprise policy.

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. A Google Chrome extension can ask for a permission to "Read and change all your data on the websites you visit".

The latter has significant side-effects for browser based voting applications. It is a non-trivial task to protect the integrity and privacy of the ballot while a browser has been augmented with extensions. To mitigate the implied risks, the voter would have to make sure that browser extensions would not have access to the content on the vote casting website. This is infeasible for an average user.

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

[10] https://fetch.spec.whatwg.org/.

[11] The Android version of Firefox supports extensions and there exist third-party mobile browsers (e.g. Kiwi) that support Google Chrome extensions.

[12] https://toreini.github.io/projects/domtegrity.html.

3.3 Availability

It is common for vendors to bundle browsers with their devices. This results in tens of browsers being used only by a small fraction of users.

Additionally, it was recently shown by Kondracki *et al.* that data saving browsers can affect both privacy and integrity of the transmitted content [16]. However, only some of these browsers intercepted TLS connections. This illustrates the need for separately validating browsers that can be used for voting.

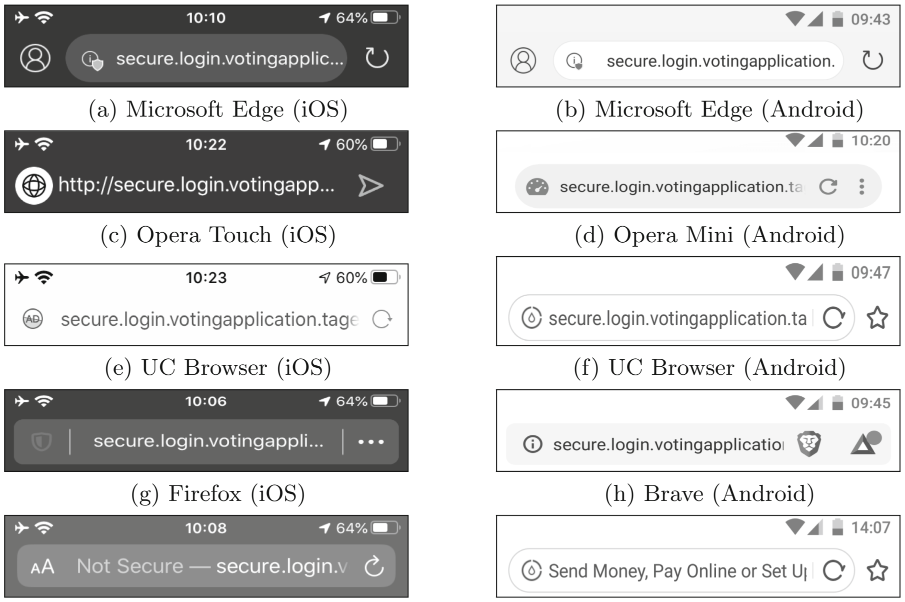
On the other hand, monitoring the security of all marginally used browsers is not realistic. Thus, in the context of voting, supporting only a limited number of browsers seems unavoidable. At the same time, correctly detecting a browser on server-side is not 100% reliable as browsers may lie about their identity.

Authenticity of the Voting Application. A voter would have to check the TLS certificate of the web site to validate its authenticity. However, it may not be possible to view the full certificate on mobile browsers. By testing multiple browsers on Android and iOS (Google Chrome, Firefox, Brave, Microsoft Edge, UC Browser, Samsung Browser, Opera Mini/Touch, Safari), we discovered that only Google Chrome, Brave and Samsung Browser running on Android display detailed information about the certificate. Google Chrome on iOS and Firefox on Android allowed the user to only view the issuer of the certificate. The rest of the tested browsers only displayed a padlock icon on the URL bar. As it is trivial to get a valid TLS certificate for one's own domain, relying on the padlock icon can create a false sense of security about the website's trustworthiness [8].

URL Bar Visibility. Browsers on smartphones are optimised to increase the available screen area and thus limit the information displayed on the URL bar. We tested multiple browsers on Android and iOS (Google Chrome, Firefox, Brave, Microsoft Edge, UC Browser, Samsung Browser, Opera Mini/Touch, Safari) to see how their URL bar behaves.

It turned out that there is no common behaviour in this respect. Most of the tested browsers hide the URL bar while scrolling or changing page orientation. Only the iOS version of Chrome always displayed at least part of the URL.

It has been observed that some mobile browsers have issues with displaying long URL-s [20], which can be abused by phishing attacks. To prevent confusion, browsers should always display the domain name of the visited web site. Our testing revealed that Safari, Firefox, Edge, Opera Touch and UC Browser on iOS prioritize subdomains when using vertical layout as illustrated in Fig. 1. Testing the same layout in Android showed that Google Chrome, Firefox and Samsung Browser display the domain name of the visited website, while Microsoft's Edge, Opera Mini, UC Browser and Brave prioritize the left side of the URL. Surprisingly, UC Browser on Android showed the title of the web page instead of the URL, thereby making phishing attacks trivial, see Fig. 1.


10:10
64%
secure.login.votingapplic...
09:43
secure.login.votingapplication.
10:22
60%
http://secure.login.votingapp...
10:20
secure.login.votingapplication.ta
10:23
60%
secure.login.votingapplication.tage
09:47
secure.login.votingapplication.ta
10:06
64%
secure.login.votingappli...
09:45
secure.login.votingapplicatio
10:08
64%
AA Not Secure — secure.login.v
14:07
Send Money, Pay Online or Set U

(a) Microsoft Edge (iOS)
(b) Microsoft Edge (Android)
(c) Opera Touch (iOS)
(d) Opera Mini (Android)
(e) UC Browser (iOS)
(f) UC Browser (Android)
(g) Firefox (iOS)
(h) Brave (Android)
(i) Safari (iOS)
(j) UC Browser (Android) – screenshot is taken while visiting paypal.com

Fig. 1. URL bars of mobile browsers that do not display the domain of the website. The screenshots were taken in June 2020.

Such behaviour can be exploited by running targeted phishing campaigns to trick voters into using a fake voting website. A successful attack could reveal how these voters vote. It may also allow an attacker to block or change the vote that is going to be cast, or interfere with the vote verification processes.

4 Issues with a Standalone Voting Application

The main alternative to the browser-based approach is to have a standalone mobile application as a voting client. In this Section we will identify the main issues that can be caused by running a voting client in either iOS or Android.

In the following we assume that a standalone voting application is implemented as a native executable application, packaged for the given platform and distributed via the mainstream appstore. Thus, the applications are distributed as digitally signed packages (APK in Android, IPA in iOS).

Both Android and iOS applications are executed as isolated processes on the OS level; tampering with these processes requires system level access. The OS level APIs give fine-grained control over the voting application development and enable e.g. certificate pinning for TLS connections.

Usability. A standalone application can use slightly more screen area compared to a browser, but still has the issue of being unable to display the full list of candidates. In addition, smartphone applications are used in portrait mode by default, whereas PC based applications are mostly used in landscape mode.

Users interact with smartphone applications via tapping and gestures, which are not precise actions. Thus, the relative size of the UI elements has to be bigger compared to desktop applications. Therefore, complex interactions presented in a single view on a PC need to be restructured into multiple views for smartphones.

4.1 Privacy

Sandboxing. To protect the integrity and secrecy of the ballot, the voting application should be isolated from non-system software. Such behaviour can be provided by sandboxes isolating applications from each other. Unfortunately, relying just on a sandbox is insufficient as malware with root access can influence it.

Android and iOS sandbox all user installed applications while setting restrictions to getting root access. As a result, non-rooted mobile applications can not significantly influence or monitor execution of the voting application.

Android isolates applications in the kernel level by assigning a unique user ID to each application and running them in separate processes[13]. In addition, SELinux is used to enforce mandatory access control over Android processes.

iOS application sandbox is built on top of a mandatory access control system [29]. Contrarily to Android, regular applications do not run as separate users, instead they are executed as a user named `mobile`, while system processes can run as root. iOS applications are isolated into `chroot` style jails, such that each application is only able to access its own directory and has limited access to system resources. While jailbreaking does not automatically disable the application sandbox, it breaks the security model by allowing applications to get root access and thereby influence other applications [29].

Third-Party Applications. Privacy issues can also be caused by side-channel leakages from legitimate applications. For example, it is common for Android applications to silently request the list of other installed applications [26]. This is usually done by the advertisement libraries by using an official Android API.

Information about the voting or vote verification app can be abused by an attacker. When a user has not installed a voting application, an attacker may attempt to cast a vote on voter's behalf with a lower risk of the voter getting a potential "You have already voted" message. In case an attacker can predict which voters do not verify their votes, it is possible to run targeted vote manipulation attacks without the risk of being caught by verification.

Another Android specific risk is non-removable bloatware pre-installed into the device by the vendor. The risks of pre-installed applications were recently

[13] https://source.android.com/security/features.

studied by Gamba *et al.* by gathering 424,584 unique firmware samples [9]. They found that 74% of the non-public pre-installed applications did not get security updates, and 41% were not patched for at least 5 years. They also stated that bloatware poses a threat to user's privacy as it often contains third-party libraries and custom permissions, which are not explained to the users.

Man-in-the-Middle Attacks. We mentioned in Sect. 3.1 that MITM attacks pose a threat to the voting systems that rely on TLS to protect ballots [2]. However, in iOS and Android such attacks can be mitigated by certificate pinning.

In order to bypass certificate pinning in iOS, the device has to be jailbroken [29]. The common way to bypass pinning is to use the tool `SSL Kill Switch` 2, which patches the API that is used for certificate validation[14].

Before Android 7, third-party VPN applications often violated user's privacy [13]. Once an Android application had requested the `BIND_VPN_SERVICE` permission, it was able to break sandboxing and redirect the traffic of other applications. There were also examples of VPN applications doing TLS-interception.

However, since the release of Android 7, root access is required to add new root certificates that are trusted system-wide[15]. After that change, malicious TLS interception become significantly more difficult on Android devices.

4.2 Integrity

Risks somewhat similar to installing third party extensions to browsers (see Sect. 3.2) also occur on the OS level where third party apps can be installed. However, special attention is needed for applications that run in root privileges.

Rooting and Malicious Applications. Applications running with root access are a potential threat for the voting application. First, such malware can influence how other programs behave and thereby either drop or modify the ballot. Second, by getting access to the voter's credentials, it could cast a vote on behalf of her. Third, by intercepting information, ballot privacy could be violated.

iOS has strict rules for installing applications, allowing installation only from Apple's App Store. This restriction can be bypassed by acquiring superuser credentials on the device (jailbreaking). However, jailbreaking is non-trivial and may not be available for all iOS versions.

In Android, getting root access is easier. However, rooting is not even necessary in order to install third party application stores and applications on an Android device. Third party application stores, in turn, have fewer restrictions for uploading applications and provide a good distribution channel for malware.

On desktop platforms, anti-virus software can detect some types of malware. Anti-virus-like solutions also exist for Android and iOS, but ironically, their capabilities are limited as both of the OSes sandbox applications quite strictly.

[14] https://github.com/nabla-c0d3/ssl-kill-switch2.

[15] https://android-developers.googleblog.com/2016/07/changes-to-trusted-certificate.html.

A safer execution environment could be provided for the voting application by not supporting rooted devices. While various heuristics can be used to detect rooting, they are generally easy to circumvent [14]. Due to the threat of malware, absolute ballot privacy can not be guaranteed unless code voting is used [11]. However, that would significantly complicate the voting process for the voters.

Update Policy and Legacy Operating Systems. Android's update policy differentiates it from the other mainstream OSes. There is a problem of issuing software updates due to multiple dependencies between the software vendor, device manufacturers and the chip producer. Additionally, the business model of many device manufacturers is built around short product lifetime. As a result, updates and security patches are delayed or not available at all for many devices.

The situation with Android update policy has slightly improved during the last few years due to the separation of hardware specific code from the rest of Android[16] and introduction of the Android Enterprise Recommended program[17]. The former makes it easier for the vendors to create updates, but it does not force devices to be updated. The latter gives vendors the option to assign a special label to the devices which are guaranteed to get regular security patches.

A significant percentage of Android devices run old Android versions that contain severe vulnerabilities. While it would be preferable not to offer voting software for such devices, in practice this is not possible (iOS devices without jailbreaks being available being an exception). When an official voting application is not offered, it is still possible to create an independent voting client that follows the official API to communicate with the back-end server. The API can be found either from the documentation or by reverse engineering the official voting application as demonstrated e.g. for Voatz [28].

Authenticity and Integrity of the Voting Application. On desktop platforms, the voting application can in principle be distributed via an unreliable channel as the integrity and authenticity of the application can be verified post-transit. It can be done by relying on checksums or signatures, given that the information about how to verify is distributed over reliable channels. Windows and macOS automatically verify integrity of the signed binaries at the load time.

Android applications have to be signed before being distributed through Google Play Store. While the signature binds an application along with its updated versions to a certain key, it does not give assurance about the identity of the application developer as the signature can be issued with a self-signed certificate[18]. In addition, Android supports legacy signature schemes to provide backward compatibility. A recent study by Yoshida *et al.* showed that apps signed using MD5 and 512-bit RSA can still be found on Google Play Store [30].

[16] https://source.android.com/devices/architecture#hidl.
[17] https://www.android.com/enterprise/recommended/.
[18] https://source.android.com/security/apksigning.

Android allows new applications signed with APK Signature scheme v2 (or v3) to use RSA1024, although stronger alternatives are supported[19]. However, as RSA1024 provides only 80-bit security level, it is no longer recommended for even mid-term applications; at least RSA3072 should be used instead [1]. Unfortunately, the issues with weak signatures do not seem to disappear as applications uploaded to Google Play Store must be signed with a key that is valid at least until October 22nd 2033.[20] While key rotation functionality was added to APK Signature scheme v3 in August 2018,[21] it does not resolve issues with older signature schemes. There is no straightforward way to replace the signing key of an application that is signed with an older signature scheme[22].

iOS applications have to be signed with a developer key certified by Apple. The digital identity used for signing must be available through keychain. The macOS certificate assistant tool allows for requesting certificates with RSA (bitlengths 2048 to 8192) and ECC (bitlengths 256 to 521) keys.

4.3 Availability

On desktop platforms, the easiest solution to distribute an application is to allow it to be downloaded directly from its web page. In case of mobile platforms, installing apps from non-official sources is either impossible or not recommended due to security concerns. Thus, an official voting application would have to be distributed via the application store, but this can cause unexpected side-effects.

There is a risk of installing a bogus voting application as the app stores are not able to identify and remove all copycat applications. Counterfeit applications exist both in Google Play and in Apple's App Store[23]. A recent study analysed 1.2 million Google Play applications and detected 49,608 that were similar to one of the top 10,000 most popular applications in Google Play Store [22]. The potential counterfeits were scanned by VirusTotal and out of these, 2040 were classified to be malicious by at least 5 independent antivirus tools.

It should not be possible for third parties to issue updates for the voting application. However, Google is promoting an app signing functionality that allows application developers to out-source handling of the signing key to Google.[24] Although the developer can also use an upload key, this can be reset by Google. Thus, when releasing an official voting application, the signing key must not be disclosed to third parties and such functionality must not be used.

Additional risk lies in the limited control over the distribution channel. The application stores decide which applications to host and thus it is not guaranteed that a voting application is accepted in time. Similar problems may arise in case a hot-fix is needed due to a reported bug or vulnerability.

[19] https://source.android.com/security/apksigning/v2.
[20] https://developer.android.com/studio/publish/app-signing.
[21] https://source.android.com/security/apksigning/v3.
[22] http://support.google.com/googleplay/android-developer/answer/9842756.
[23] https://www.nytimes.com/2016/11/07/technology/more-iphone-fake-retail-apps-before-holidays.html.
[24] https://developer.android.com/studio/publish/app-signing.

Another aspect that may need to be considered is the leakage of app store profiles of voters who have downloaded the voting application. When a malicious actor gets access to such information, it may be possible to conduct targeted attacks against eligible voters who have not installed a voting application similarly to the attacks described in Sects. 3.1 and 4.1.

5 Discussion, Conclusions and Future Work

Several recent developments like increased migration and SARS-CoV-2 virus spreading have increased the motivation to introduce options for remote voting. There are two main alternatives for absentee ballot delivery – postal system and Internet – with both having their benefits and shortcomings.

In case remote voting is implemented via Internet, the main candidates for the voting client platform are PCs and mobile devices. This paper focused on the latter option, giving an overview of the arising issues in two main scenarios – browser based and standalone voting application on mobile platforms. Although mobile platforms are becoming more mature, they still have several shortcomings when considering them for remote electronic voting.

Firstly, there are a lot of legacy OS versions around that the vendors have no incentive to update, particularly in case of Android. However, in order to provide the voting option for a significant part of the electorate, there would be pressure to also support out-of date mobile OSes.

Second, smaller screen size makes it necessary to display only part of the content. This is a challenge from both the voting application UI and security point of view (as e.g. certificates and URLs can be displayed only partly).

When considering browser-based voting client as an alternative to a standalone app, one has to take into account that browsers extend the attack surface significantly. Mobile browsers also lag behind their PC counterparts in adoption of new security features. There is no standard way for users to check the integrity of web applications, and browser behaviour can be easily changed by third-party extensions that the users install without giving it much consideration.

As many of the shortcomings do not affect only voting, but have impact on a much wider variety of use cases, they can be expected to be fixed over time. It will be interesting to re-assess the situation in a few years to see if the situation will have been improved.

There are also several areas that remained outside of the scope of current paper and are left as subject for further research.

Voter authentication is likely to differ between desktop and mobile platforms. Smartphones tend to have relatively high quality cameras and the integration of fingerprint readers is becoming more widespread. As a negative side, it is difficult to interface a smartphone with a smart card based electronic identity solution. In addition, the mobile device itself is often used as a second authentication factor, which may exclude some authentication methods.

When running cryptographic protocols, access to high-quality randomness is important. While cryptographically strong random number generators are increasingly available on mobile platforms, their usability by mobile browsers still requires future research.

Acknowledgements. This paper has been supported by the Estonian Research Council under the grant number PRG920 and European Regional Development Fund through the grant number EU48684 and EXCITE centre of excellence. The authors are grateful to the Estonian Information System Authority and State Electoral Office for their support to the research process.

References

1. Abdalla, M., et al.: Algorithms, key size and protocols report. Tech. rep. ECRYPT CSA (2018). https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf
2. Cardillo, A., Essex, A.: The threat of SSL/TLS stripping to online voting. Proc. E-Vote-ID **2018**, 35–50 (2018)
3. Cetinkaya, O.: Analysis of security requirements for cryptographic voting protocols (extended abstract). In: Proceedings ARES 2008, pp. 1451–1456. IEEE Computer Society (2008)
4. Corbet, S.: France Holds Local Elections Despite COVID-19 Outbreak Fears. Time, March 2020. https://time.com/5803469/france-local-elections-coronavirus/
5. Durumeric, Z., et al.: The security impact of HTTPS interception. In: Proceedings of NDSS 2017. The Internet Society (2017)
6. Eilu, E., Baguma, R.: Designing reality Fit M-voting. In: Proceedings of the 7th International Conference on Theory and Practice of Electronic Governance. ICE-GOV 2013, pp. 326–329. ACM (2013)
7. Ekong, U.O., Ekong, V.: M-voting: a panacea for enhanced e-participation. Asian J. Inf. Technol. **9**(2), 111–116 (2010)
8. Felt, A.P., et al.: Rethinking connection security indicators. In: Twelfth Symposium on Usable Privacy and Security (SOUPS 2016), Denver, CO, pp. 1–14. USENIX Association, June 2016. https://www.usenix.org/conference/soups2016/technical-sessions/presentation/porter-felt
9. Gamba, J., Rashed, M., Razaghpanah, A., Tapiador, J., Vallina-Rodriguez, N.: An analysis of pre-installed android software. In: 2020 IEEE Symposium on Security and Privacy (SP), pp. 197–213, May 2020
10. Heiberg, S., Willemson, J.: Modeling threats of a voting method. In: Design, Development, and Use of Secure Electronic Voting Systems, pp. 128–148. IGI Global (2014)
11. Helbach, J., Schwenk, J.: Secure internet voting with code sheets. In: Alkassar, A., Volkamer, M. (eds.) Vote-ID 2007. LNCS, vol. 4896, pp. 166–177. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77493-8_15
12. Hermanns, H.: Mobile democracy: mobile phones as democratic tools. Politics **28**(2), 74–82 (2008)
13. Ikram, M., Vallina-Rodriguez, N., Seneviratne, S., Kâafar, M.A., Paxson, V.: An analysis of the privacy and security risks of android VPN permission-enabled apps. Proc. IMC **2016**, 349–364 (2016)

14. Kellner, A., Horlboge, M., Rieck, K., Wressnegger, C.: False sense of security: a study on the effectivity of jailbreak detection in banking apps. In: Proceedings of IEEE EuroS&P 2019, pp. 1–14. IEEE (2019)
15. Kogeda, O.P., Mpekoa, N.: Model for a mobile phone voting system for South Africa. In: Proceedings of 15th Annual Conference on World Wide Web Applications, Cape Town, South Africa (2013)
16. Kondracki, B., Aliyeva, A., Egele, M., Polakis, J., Nikiforakis, N.: Meddling middlemen: empirical analysis of the risks of data-saving mobile browsers. In: 2020 IEEE S&P, pp. 1678–1692. IEEE, May 2020
17. Krimmer, R.: The evolution of e-voting: why voting technology is used and how it affects democracy. Ph.D. thesis, Tallinn University of Technology (2012)
18. Leith, D.J.: Web browser privacy: what do browsers say when they phone home? (2020), SCSS Technical Report, 24th February 2020
19. Luo, M., Laperdrix, P., Honarmand, N., Nikiforakis, N.: Time does not heal all wounds: a longitudinal analysis of security-mechanism support in mobile browsers. In: Proceedings of NDSS 2019. The Internet Society (2019)
20. Luo, M., Starov, O., Honarmand, N., Nikiforakis, N.: Hindsight: understanding the evolution of UI vulnerabilities in mobile browsers. In: Proceedings of the 2017 ACM CCS, CCS 2017, pp. 149–162. ACM (2017)
21. Mitrou, L., Gritzalis, D., Katsikas, S.K.: Revisiting legal and regulatory requirements for secure e-voting. In: Ghonaimy, A., El-Hadidi, M.T., Aslan, H.K. (eds.) Security in the Information Society: Visions and Perspectives, IFIP TC11 17th International Conference on Information Security (SEC2002), 7–9 May 2002, Cairo, Egypt. IFIP Conference Proceedings, vol. 214, pp. 469–480. Kluwer (2002)
22. Rajasegaran, J., Karunanayake, N., Gunathillake, A., Seneviratne, S., Jourjon, G.: A multi-modal neural embeddings approach for detecting mobile counterfeit apps. In: The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, 13–17 May 2019, pp. 3165–3171. ACM (2019)
23. Reis, C., Moshchuk, A., Oskov, N.: Site isolation: process separation for web sites within the browser. In: 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, pp. 1661–1678. USENIX Association, August 2019. https://www.usenix.org/conference/usenixsecurity19/presentation/reis
24. Salvador, D., Cucurull, J., Julià, P.: wraudit: a tool to transparently monitor web resources' integrity. In: Groza, A., Prasath, R. (eds.) MIKE 2018. LNCS (LNAI), vol. 11308, pp. 239–247. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05918-7_21
25. Schryen, G.: Security aspects of internet voting. In: 37th Hawaii International Conference on System Sciences (HICSS-37 2004), CD-ROM/Abstracts Proceedings, 5–8 January 2004, Big Island, HI, USA. IEEE Computer Society (2004)
26. Scoccia, G.L., Kanj, I., Malavolta, I., Razavi, K.: 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). http://www.ivanomalavolta.com/files/papers/MOBILESoft_iam_2020.pdf
27. Shapshak, T.: Africa not just a mobile-first continent - it's mobile only (2012), CNN Business. https://edition.cnn.com/2012/10/04/tech/mobile/africa-mobile-opinion/index.html
28. Specter, M.A., Koppel, J., Weitzner, D.: The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in US Federal Elections (2020)

29. Thiel, D.: iOS Application Security: The Definitive Guide for Hackers and Developers. No Starch Press (2016). https://nostarch.com/iossecurity
30. Yoshida, K., Imai, H., Serizawa, N., Mori, T., Kanaoka, A.: Understanding the origins of weak cryptographic algorithms used for signing android apps. J. Inf. Process. **27**, 593–602 (2019)