Reproducing Vote Verification Application Builds for Estonian I-Voting System

Report in Research Seminar in Cryptography (MTAT.07.022)

Annika Tammik
University of Tartu, Tartu, Estonia

December 20, 2017

1 Introduction

Internet voting has been available for Estonians already for more than 10 years – the first election which gave the possibility to vote online in Estonia was conducted in 2005. During this whole period, the basic concept behind the internet voting has stayed the same. It mimics the double envelope postal voting where the "inner" envelope is the vote encryption done with the server's public key and the signed "outer" envelope is the signing done with the national eID signing device (ID card, Mobile-ID or Digi-ID). [1]

In 2011 several potential attacks were published against this rather simple scheme. To counter such attacks, vote verification scheme was launched in 2013 that allows the voters to verify the cast vote using a mobile device [2]. Currently the vote verification application is provided for Android and iOS mobile operating systems, the distribution channels being Google Play Store[1] and iOS App Store[2]. The source code of these apps is published on Estonian National Electoral Committee's GitHub repository [3] to give a possibility for the public to audit the code and verify that the verification apps work as expected.

The main objective of this work is to check whether the vote verification applications distributed in the app stores can be compiled from the source code that has been made publicly available by Estonian National Electoral Committee. The experiments were performed using the Vote Verification application versions that were distributed in the I-voting period of the Estonian municipal council election held in October 2017 [4].

The report will go through the different steps that were done during this experiment – monitoring the binaries, building the app from the source code, comparing build result with the distributed version and trying to reproduce it based on the differences found. Last but not least, there are also some recommendations to the app developers and the Estonian National Electoral Committee on what could be changed in order to make the build process and the reproduction of the apps easier.

2 Monitoring the Apps in the App Stores

Apps are designed for mobile devices and due to that their only official distribution channel is the app store that is integrated into those supported devices. Thus, the main challenge in the

[1] `https://play.google.com/store/apps/details?id=ee.ivxv.ivotingverification`
[2] `https://itunes.apple.com/us/app/eh-kontrollrakendus/id1265172086`

monitoring part was to find a way how to download the apps from the app stores into a device that is neither a phone nor a tablet. It was especially challenging as downloading apps to a non-mobile device is neither expected nor officially supported by the app stores. This means that there is also not much relevant documentation nor any official guidelines available on how to do this.

2.1 Android

In case of Android there are several sites[3,4] from where it is possible to download the Google Play Store apps. Yet, as the objective of this work is to verify the binaries distributed from the official app store, then using the available unofficial web solutions was in the current context not an option.

There are also several GitHub projects[5,6] that provide an unofficial API for downloading the apps from Google Play Store. The problem there, though, is that most of those projects are 3 - 5 years old, sometimes poorly documented and most important – they do not work. Fortunately, after trying out several different projects the Google Play Unofficial Python API[7] was found to be actively developed and actually working. Thus, this project was used to build Google Play Store app monitoring.

2.2 iOS

In case of iOS the problem of app binary availability is even more severe. There again are a few websites[8,9] that promote a possibility to download iOS apps online. Yet, when trying to actually download the apps, all those sites do is redirect the user to the official iTunes preview page of the selected app. Also, searching for unofficial iOS App Store API projects from GitHub gives no results, similarly to all the Google searches for any relevant guidelines.

As another option, it used to be possible, using the App Store in iTunes desktop version, to at least download the binaries of the iOS apps from the connected iOS mobile device (for backup purposes). Earlier this year such possibility was removed when the backup policy of apps on iOS devices was revised and App Store integration removed from iTunes [5].

Thus, as there seems to be no straightforward way to download iOS apps through an API to a computer, then the iOS apps were left out of the scope of this work.

2.3 Vote Verification App in Google Play Store

2.3.1 Downloading binaries from Google Play App Store

The downloading of an app from Google Play App Store requires several requests with different complexity of payload.

First of all, in order to get access to the Play Store services, authorization is needed. For a new user this is a 3-step process. As the first step, a login request must be made using valid credentials from a Google account. A successful login request will return an authentication key and a token. As the second step, a device check-in is required where the device data[10] has to

[3] `https://apkpure.com`
[4] `https://apps.evozi.com/apk-downloader/`
[5] `https://github.com/egirault/googleplay-api`
[6] `https://github.com/CMUChimpsLab/googleplay-api`
[7] `https://github.com/NoMore201/googleplay-api`
[8] `https://en.softonic.com/iphone`
[9] `http://download.cnet.com/s/software/ios/`
[10] For the implementation being, the OnePlus One smartphone was emulated.

be sent together with the acquired authentication key to receive a Google device identifier. The last step requires another login request, using the token from the first login request, to acquire a reusable authentication key. Both, the device identifier and the second authentication key are required to be set as headers to all the subsequent requests.

As the downloads can only be made when the package name and the version code is known, a search query is needed to translate the app name to the package name (in the current case from "EH kontrollrakendus" to "ee.ivxv.ivotingverification"). When knowing the package name, it is possible to make a details query to determine the last available version (identified by the version code value) of the package. The package name and the version code are prerequisites for the downloading.

Downloading an app from Google Play Store is another 3-step process. First of all, a purchase request must be made with the package name and a version code. If the user is allowed to download the requested app, a download token is returned. The token can be used to obtain a delivery URL together with an authentication cookie that is also required for the delivery request. Finally, using the received URL and the authentication cookie, the actual binary `.apk` package can be downloaded.

2.3.2 Android App Monitoring System

Since for the app monitoring purpose only a small part of the Google Play Unofficial Python API is required, a separate project was created with the focus on the monitoring. As a result, there is now a Google Play Store App Monitor available in BitBucket[11] that can be used to automatically download new versions of free Android apps.

Steps needed to set up the Google Play App Store Monitor:

1. Clone the Google Play App Store Monitor:
   ```
   $ git clone https://bitbucket.org/emomeio/google_play_store_app_monitor.git
   ```
2. Move to the cloned folder:
   ```
   $ cd google_play_store_app_monitor
   ```
3. Follow the instructions from the `README.md` file.

2.3.3 Monitoring the Vote Verification App in Google Play Store

The internet voting period of the municipal council election was from the 5th until the 11th of October 2017 [4]. Before elections, on the 2nd of October a new version (version code 18) of the vote verification app was released in Google Play Store. The app[12] was called "EH kontrollrakendus" with the package name "ee.ivxv.ivotingverification".

During the voting period the monitoring was done daily. On the evening of the 10th of October this resulted in downloading a new binary version of the application (version code 22) from Google Play Store. Looking into the details available through the monitoring project's details API it was possible to see that the new version had been released due to "Fixes in connection with not updated ID-cards":

```
details {
     appDetails {
             developerName: "Cybernetica AS"
             versionCode: 22
```

[11] `https://bitbucket.org/emomeio/google_play_store_app_monitor`
[12] `https://play.google.com/store/apps/details?id=ee.ivxv.ivotingverification`

```
            versionString: "3.1.8"
            ...
            packageName: "ee.ivxv.ivotingverification"
            recentChangesHtml: "Parandused seoses uuendamata ID-kaartidega"
            uploadDate: "Oct 10, 2017"
            ...
        }
}
```

Thus, there were two versions of the Android Vote Verification app released that were used to verify the cast vote during those elections. This results in two samples available for the further analysis:

1. `ee.ivxv.ivotingverification_18.apk`[13] – downloaded on October 2, 2017.
2. `ee.ivxv.ivotingverification_22.apk`[14] – downloaded on October 10, 2017.

3 Building the Android Vote Verification App from Source Code

The source code of the Android Vote Verification App was published in GitHub [3] on the 5th of September with one commit. What is significant, though, is that the source code available in GitHub has not been updated since then (last checked: December 9th, 2017). This means that the source code of the updated app (version code 22) released during the election period has not been made publicly available. This makes already clear that it will not be possible to fully reproduce the second binary released during the voting period.

Nevertheless, to build any software, it is useful to have software build instructions available. In case of the Android Vote Verification App no information neither in the form of a `README.md` file nor a set-up script was available. In such cases, it is useful to look into the technology stack used in the project to understand which dependencies are required. Thus, first of all, since it is an Android app, the Android SDK is needed. Also when looking at the source code, it is possible to see that the app is written in Java (application code is in `ivotingverification/app/src/main/java` folder and the files there have a `.java` extension) and uses Gradle as the build system (there is a `build.gradle` file in the project root folder). Thus, apparently, Java and Gradle are also required to build the app.

As in the current stage it is still unclear if and to what extent version numbers of the compilers and dependencies might affect the build result, the latest versions are used.

3.0.1 Steps to build the Vote Verification App (on Ubuntu 16.04)

1. Download Android SDK:

```
$ wget http://dl.google.com/android/android-sdk_r24.2-linux.tgz

$ tar -xvf android-sdk_r24.2-linux.tgz -C $HOME
$ cd $HOME/android-sdk-linux/tools
$ echo 'y' | $HOME/android-sdk-linux/tools/android update sdk --no-ui

$ export PATH=${PATH}:$HOME/android-sdk-linux/platform-tools
$ export PATH=${PATH}:$HOME/android-sdk-linux/tools
$ export PATH=${PATH}:$HOME/android-sdk-linux/build-tools/22.0.1/
$ export ANDROID_HOME=${HOME}/android-sdk-linux/

$ sudo apt-get install libc6:i386 libstdc++6:i386
$ sudo apt-get install zlib1g:i386
```

[13] SHA256: `35dac3859ffbe4d85acd20e51c117f17425b26e2db4520ce9aea7533e7583c94`
[14] SHA256: `cbb1f86cebfcd2c02715e6ca2999b5d609ab6aecb092115d25b205ddc00f221b`

2. Download Java JDK:

```
$ sudo apt-get install openjdk-8-jdk
$ apt-cache search jdk

$ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk
$ export PATH=$PATH:/usr/lib/jvm/java-8-openjdk/bin

$ javac -version
javac 1.8.0_151
```

3. Download Gradle:

```
$ sudo mkdir /opt/gradle
$ wget https://services.gradle.org/distributions/gradle-4.3.1-bin.zip
$ sudo unzip -d /opt/gradle gradle-4.3.1-bin.zip

$ export PATH=$PATH:/opt/gradle/gradle-4.3.1/bin
```

4. Download the source code of the app and checkout relevant revision:

```
$ git clone https://github.com/vvk-ehk/ivotingverification.git
$ cd ivotingverification
$ git checkout 20e64ed8726cba1b19e28b0ed00ef95ffcd36d3a
```

5. View the app's dependency tree for required local dependencies:

```
$ gradle androidDependencies
...
+--- __local_jars__:/home/at/ivotingverification/app/libs/xom-1.2.10.jar:unspecified@jar
\--- __local_jars__:/home/at/ivotingverification/app/libs/zxing_core-3.1.0.jar:unspecified@jar
```

6. Add missing libraries:

```
$ cd app
$ mkdir libs
$ cd libs

$ wget http://central.maven.org/maven2/com/io7m/xom/xom/1.2.10/xom-1.2.10.jar

$ wget http://central.maven.org/maven2/com/google/zxing/core/3.1.0/core-3.1.0.jar
$ mv core-3.1.0.jar zxing_core-3.1.0.jar
```

7. Build the app:

```
$ gradle assemble
```

8. Go to the build outputs folder and verify that the `.apk` was created there. The name of the file to look for is `app-release-unsigned.apk`.

```
$ cd [CORRECT_PATH]/ivotingverification/app/build/outputs/apk/
```

As a result of this build process an `app-release-unsigned.apk`[15] file is produce which is an unsigned version of the Vote Verification App. Therefore, it is expected that even if the source code and the build environment are exactly the same, the result will not directly match the corresponding binaries. In order to understand weather the hash difference comes only from the signing block or are there also other differences in the files, a binary comparison has to be conducted.

[15] SHA256: 2f5c4139ec9920cae14c67df58b4c0dc84606d32466487311af999ab915fbe56

4 Reproducing the Android Vote Verification App

Reproducible builds are a set of software development practices that create a verifiable path from human readable source code to the binary code used by computers. Meaning, that if two different people compile the project from the same source code, their outputs are bitwise identical to each other and to the binary distributed by the owner of the code. This allows people to verify that binaries downloadable from the Internet come from the corresponding sources and have not, for example, had malware added to them.[16]

If the build process is not set up by knowingly following the principles of reproducible builds, then it is very likely that the build reproduction will not work just out of the box. In those cases, when still aiming to achieve a matching binary, the first task is to find the actual differences through binary comparison. When the differences are found, it is important to understand what has caused those differences in order to, if possible, reproduce an environment that would also include those causes.

4.1 The APK file type

Before actually comparing anything, it is important to generally understand what will be compared. Thus, Android apps are in more precise terms Android Package Kits[17] or just APKs. In essence APKs are just one type of archive files that are used by the Android operating system. Their file extension, similarly to the name, is `.apk`. After a program that is meant for the Android operating system is compiled, all of its parts are packaged into the APK file. The easiest way to explore the contents of an `.apk` file is by renaming the filename extension it to `.zip` and then opening it with any of the ZIP decompression tools.

4.1.1 Structure of an APK file

1. `META-INF/`: A folder holding the signature info and containing[18]:
   - `MANIFEST.MF`: A file that holds a list of all the files in the APK (except the items in the `META-INF` folder itself) and hashes of their contents. The default algorithm used is SHA-1 and the digest is represented in a base64-encoded form.
   - `CERT.SF`: A file that contains SHA-1 hash of file `MANIFEST.MF` file and all the items in it.
   - `CERT.RSA`: A file that contains the developer's signature of the `CERT.SF` file and a certificate or a certificate chain to verify the key that was used for this signature.
2. `lib/`: A folder for native device specific libraries, if such are used by the app.
3. `classes.dex`: An executable file that contains compiled Java classes.
4. `resources.arsc`: A binary file that contains compiled resources such as images, strings, or other data used by the program.
5. `res/`: A folder containing resources (such as images, layouts, animations etc.) that are not compiled into `resources.arsc`.

[16] `https://reproducible-builds.org/`
[17] `https://developer.android.com/guide/components/fundamentals.html`
[18] `https://github.com/dweinstein/android_notes/wiki/AndroidPackageSignatures`

4.2 Potential Differences in the APKs

For this comparison the first released version (version code 18) was chosen, as this has potentially less differences compared to the published source code and thus, also to the self-compiled version. Before the actual binary analysis, let's recall the reasons why those two binaries will likely differ. First of all, there is the difference coming from the signature block (`META-INF` folder), that only the official version fully has. Secondly, it is highly likely that the released source code was not used to compile the released version. The reason for this suspicion comes from the fact that only one single commit was made to the repository and it was made almost one month before the actual release of the app. Also, there is a file `ivotingverification/app/build.gradle` in the project that should theoretically include the build configuration used to compile the package. However, the version code there is set to `16` and version string to `3.1.3` (see Listing 1), while in the distributed version of the app these numbers were `18` and `3.1.5`, respectively.

```
defaultConfig {
      applicationId: "ee.ivxv.ivotingverification"
      minSdkVersion: 16
      targetSdkVersion: 26
      versionCode: 16
      versionName: "3.1.3"
}
```

Listing 1: Extract from the `ivotingverification/app/build.gradle` file

Last but not least, another possible source of differences can be in the version numbers of the dependencies and the environment used for compiling.

4.3 Comparing the APKs

There are quite a few tools available for comparing binaries. The first tool that provided clear results was `diffoscope` – a program that is able to do in-depth comparison of files, archives and directories. Multiple formats are available for the output file including an HTML file. The visualisation that is provided by the `.html` format is appealing as the file level differences are very easily noticeable.

As a result of running `diffoscope` on the binaries, the files with differences in them were highlighted (see Figure 1). Due to the missing signing block some of the files were compared incorrectly, yet the actual differences can still be clearly recognized.

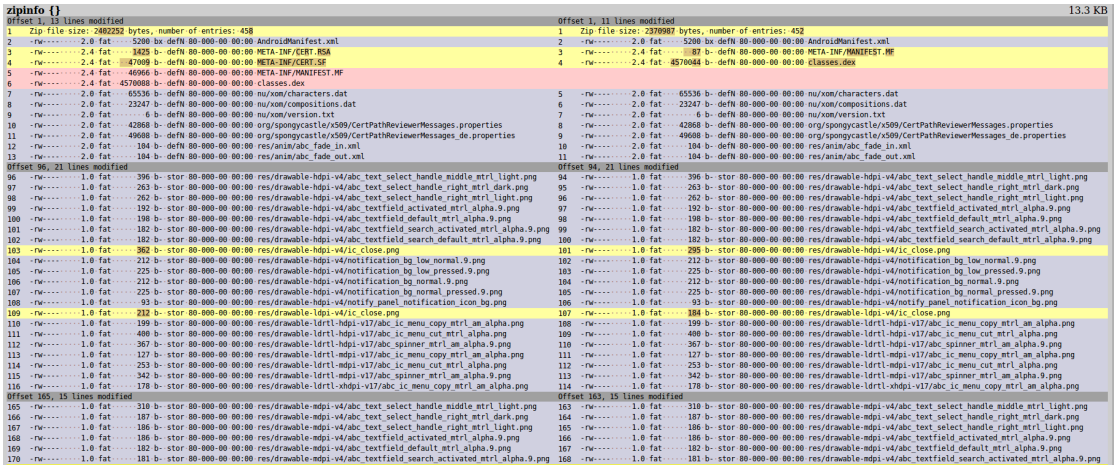

Figure 1: Differences in APKs

The second tool that provided a clear and understandable result was `apkdiff` which is actually meant for creating patches from the differences of the files. But as the patch archive also includes a summary file of the files with differences, the tool is useful also if just the files with differences in them need to be located.

```
 sha1 97ff22e8da32324bd1c79fd7b3da8a5b0c5f6dd1
-res/raw/test_of_esteid_sk_2015.crt
-res/raw/test_of_esteid_sk_2011.crt
-res/raw/tarne.bks
-res/raw/portal.bks
-META-INF/CERT.RSA
-META-INF/CERT.SF
c0|classes.dex
c1|resources.arsc
c2|AndroidManifest.xml
c3|res/drawable-hdpi-v4/ic_close.png
c4|res/drawable-mdpi-v4/ic_close.png
c5|res/drawable-xxhdpi-v4/ic_close.png
c6|res/layout/list_item_candidate.xml
c7|res/drawable-xhdpi-v4/ic_close.png
c8|res/drawable-ldpi-v4/ic_close.png
c9|res/drawable-xxxhdpi-v4/ic_close.png
c10|META-INF/MANIFEST.MF
```

Listing 2: List of files with differences

Looking at the output of the `apkdiff` summary file (see Listing 2) it is possible to see that there were quite a few differences in the compared binaries. First of all, there are four files in `res/raw/` folder (indicated with the "–" prefix) that are available in the distributed binary but missing from the self compiled version. The first two `.crt` files seem to be X.509 certificates of the Estonian Certification Centre (Sertifitseerimiskeskus). The other two files with the `.bks` extension are BouncyCastle Keystore[19] files. In both cases it is not clear why these files were added to the distributed version and what are they used for. Nevertheless, the files which were expected to be missing and different are the ones in the `META-INF` folder. As already said before, those differences are expected and are there because the distributed version is signed and the self-compiled version is not. What were not exactly expected, were the differences in the files listed with prefixes from "c0" to "c9". However, they prove the suspicion that the distributed version was compiled using source code which differs from the source code published in GitHub. More specifically, the changes in the `res/` folder indicate that the old `ic_close.png` icon has been replaced with a new one and some changes have also been made to the app layout specified in the `list_item_candidate.xml` file. Additional changes to the app visual look are indicated by the changes in the `resources.arsc` file. The differences in the `classes.dex` file, on the other hand, mean that in addition to the visual resources, also the Java byte code has been changed. Last but not least, the differences in the `AndroidManifest.xml` are also expected since the version code is defined there and was different.

A simple version string adjustment step would be tolerable in the process of reproducing a build. However, the steps that introduce changes to the functionality of the app are not allowed, because making such changes would mean that the functionality of the distributed app cannot be verified from the source code that is publicly available. In the current case, it is already clear that the source code which was used to compile the distributed versions, has not been published in GitHub. Thus, trying to reproduce a matching binary by only manipulating build variants is not possible and will be skipped.

Nevertheless, some more detailed analysis was made using `diffoscope` to look into the changes that had been made in the distributed version. For this analysis additional tools – `Apktool`[20] and `enjarify`[21] – were needed.

[19] `https://cryptosense.com/bouncycastle-keystore-security/`
[20] `https://ibotpeaches.github.io/Apktool/install/`
[21] `https://github.com/Storyyeller/enjarify`

Adding `Apktool` made it possible for `diffoscope` to also compare and show differences inside the files included in the APK. Figure 2 confirms that the differences in `AndroidManifest.xml` were indeed caused by the version differences.

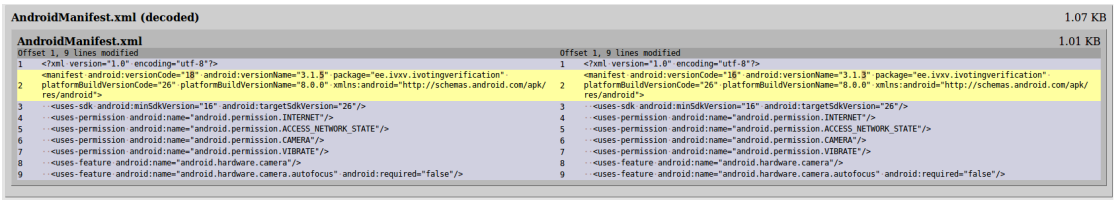

Figure 2: Differences in `AndroidManifest.xml`

The same feature made it also possible to see the differences in the `list_item_candidate.xml` file in more detail. Again, as suspected and now visible from Figure 3, there are actual code differences in this file.

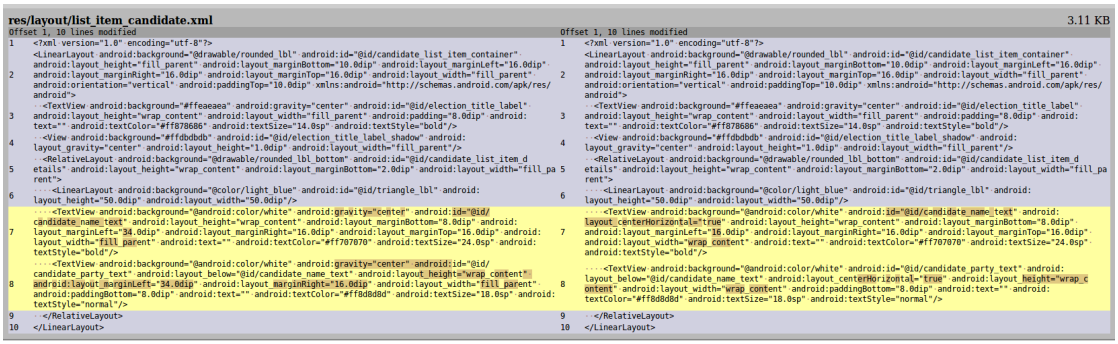

Figure 3: Differences in `list_item_candidate.xml`

Last but not least, adding `enjarify` made it possible for `diffoscope` to also highligh the differences inside the `classes.dex` file. Thus, in Figure 4 it is possible to see that two Java class file have been changed with the total difference of 126 bytes.

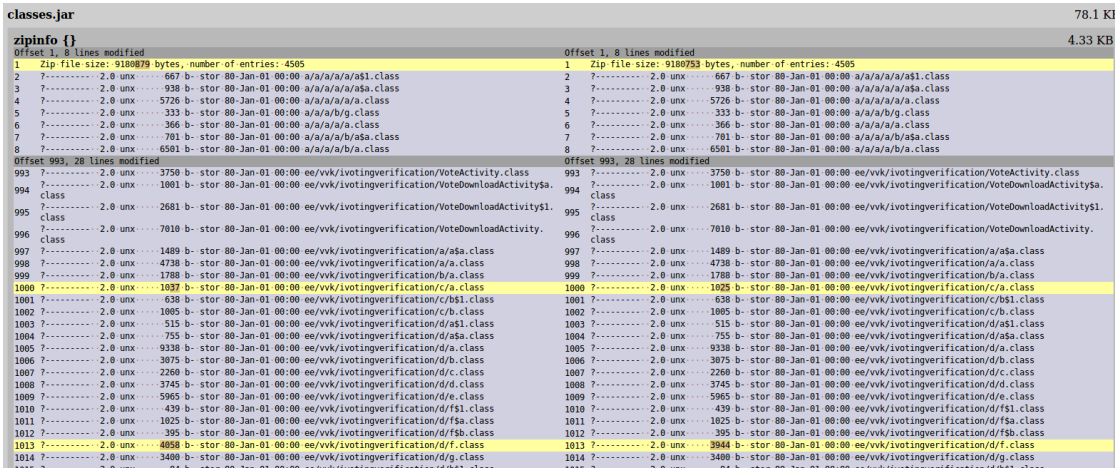

Figure 4: Differences in `classes.dex`

Going even further, `diffoscope` is also capable of showing the actual Java byte code differences in the class files. For example, Figure 5 highlights in detail what are the 12 bytes of differences in `a.class` file.

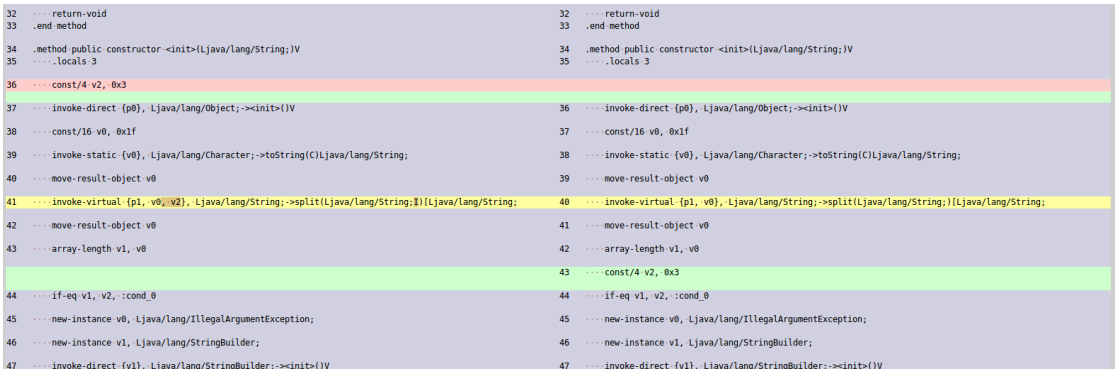

Figure 5: Code differences in `a.class` file

5 Recommendations for VVA Developers

Even though the reproduction of a matching binary was not possible, the process has led to several insights on what should be done differently in order to make building the Vote Verification App and verifying its distributed versions easier:

1. Any software project, especially when distributed publicly, should have clear instructions on how to build and run the software. Having these instructions either in a `README` file or in some set-up script would be recommended.

2. To help interested parties to reproduce the build from the source code published, it is recommended to define exact environment that was used to build the distributed app (operating system and version, SDK, JDK, Gradle etc. versions).

3. Tags should be used to mark the commits which were used to build a specific version of the app. The tagged commit should include exactly the same files that were used to build a specific version (excluding the private key that was used to sign the package, of course).

4. If the source code of the project is made publicly available, then it should also be kept up to date. The level of transparency is lost if the application is developed daily, but the source code is published only, for example, once a year. This makes tracing of the changes harder, leading the auditors to drown in the amount of changed code.

6 Conclusion

The aim of this report was to describe the steps that are needed to verify if an open-source app distributed in an app store is compiled from the same source code that is publicly available. Those steps were described in the context of the Android Vote Verification App that was distributed in Google Play Store during the I-voting period of the Estonian municipal council election held in October 2017. The report went through the different activities that were conducted during this

experiment – monitoring the binaries, building the app from the source code, comparing build result with the distributed version and trying to reproduce it based on the differences found.

Several challenges were faced during this research, starting from getting access to the application binaries that are officially distributed only to mobile devices. In case of iOS, the downloading task was complex enough to leave that platform out of the scope of the current work. In case of Android, the downloading of binaries was possible, but definitely not trivial. The complexity of the build process came from inadequate documentation, which meant that instead of being able to follow instructions, best guesses and assumptions had to be made. Since the application was simple and with small number of dependencies, a successful build was possible. Nevertheless, the instructions would have been very helpful. Last but not least, even before starting the actual binary comparison of the distributed version to the self-compiled one, there were several indications that the binaries will not match. The actual comparison confirmed this suspicion. Yet, in addition to the expected differences that came from the signature block that is missing from the self-compiled binary, there were also several other differences in the actual content of the app. This proved that in case of the Android Vote Verification app the source code that is publicly available, is not the one that was used to compile the app that was distributed during the elections. This also means that reproducing the app is not possible and currently there is no verifiable path from the source code to the distributed binaries.

This all led to several suggestions for the Vote Verification App developers, on what could be changed to simplify the build verification process that is essential for ensuring transparency. Currently, when the repository is not being updated and there are no instructions on how to build the application, the purpose of the National Electoral Committee's GitHub repository is questionable.

References

[1] Ivo Kubjas, Tiit Pikma, and Jan Willemson. Estonian Voting Verification Mechanism Revisited Again. Cryptology ePrint Archive, Report 2017/081, 2017. `https://eprint.iacr.org/2017/081`.

[2] Sven Heiberg and Jan Willemson. Verifiable Internet Voting in Estonia. In *6th International Conference on Electronic Voting 2014, (EVOTE 2014), October 28-31, 2014, Bregenz, Austria*, pages 23–28, 2014.

[3] Estonian National Electoral Committee. Source code of Estonian internet-voting system software components, July 2013. `https://github.com/vvk-ehk/`.

[4] National Electoral Committee. Municipal council election 2017, November 30, 2017. `https://www.valimised.ee/en/municipal-council-election-2017`.

[5] ZDNet. Apple's iTunes removes iOS App Store from desktop version, September 13, 2017. `http://www.zdnet.com/article/apple-removes-ios-app-store-from-desktop-versions-of-itunes/`.