NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

FACULTY OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BACHELOR THESIS

An Android application for the trustees of a distributed, end-to-end verifiable, internet voting system

Vasileios S. Poulimenos

Supervisors: **Dimitra-Isidora Roussopoulou,** Associate Professor
Nikolaos Chondros, PhD Candidate

ATHENS

OCTOBER 2015

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μια Android εφαρμογή για τους trustees ενός κατανεμημένου, από-άκρο-σε-άκρο επαληθεύσιμου, συστήματος ηλεκτρονικής ψηφοφορίας

Βασίλειος Σ. Πουλημένος

Επιβλέποντες: **Δήμητρα-Ισιδώρα Ρουσσοπούλου,** Αναπληρωτής Καθηγητής
Νικόλαος Χονδρός, Υποψήφιος Διδάκτορας

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2015

BACHELOR THESIS

An Android application for the trustees of a distributed, end-to-end verifiable, internet voting system

Vasileios S. Poulimenos

A.M.: 1115200900156

SUPERVISORS: **Dimitra-Isidora Roussopoulou,** Associate Professor
Nikolaos Chondros, PhD Candidate

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μια Android εφαρμογή για τους trustees ενός κατανεμημένου, από-άκρο-σε-άκρο επαληθεύσιμου, συστήματος ηλεκτρονικής ψηφοφορίας

Βασίλειος Σ. Πουλημένος

Α.Μ.: 1115200900156

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Δήμητρα-Ισιδώρα Ρουσσοπούλου,** Αναπληρωτής Καθηγητής
Νικόλαος Χονδρός, Υποψήφιος Διδάκτορας

ΠΕΡΙΛΗΨΗ

Τα συστήματα ηλεκτρονικής ψηφοφορίας αποτελούν μια ισχυρή τεχνολογία η οποία βελτιώνει την εκλογική διαδικασία αυξάνοντας σημαντικά την ταχύτητα καταμέτρησης ψήφων και μειώνοντας το κόστος διεξαγωγής εκλογών. Ηλεκτρονικής ψηφοφορίες οι οποίες περιλαμβάνουν κάλπη, ενώ παρέχουν τα προαναφερόμενα πλεονεκτήματα, απαιτούν τη φυσική παρουσία του ψηφοφόρου (γεγονός το οποίο πιθανώς επιφέρει καθυστερήσεις και ενόχληση). Από την άλλη μεριά, τα διαδικτυακά συστήματα ηλεκτρονικής ψηφοφορίας επιτρέπουν στους ψηφοφόρους να ψηφίζουν γρήγορα και απομακρυσμένα, μέσω κινητών τηλεφώνων ή προσωπικών υπολογιστών (μια διαδικασία η οποία ονομάζεται επίσης i-ψηφοφορία), προωθώντας δυνητικά τη δημοκρατία μέσω της παροχής αρκετά βελτιωμένης προσβασιμότητας για άτομα με σωματικά εμπόδια, αυξάνοντας, επομένως, τη συνολική συμμετοχή των ψηφοφόρων. Επιπροσθέτως, υπάρχουν συστήματα ηλεκτρονικής ψηφοφορίας τα οποία επιτρέπουν στους ψηφοφόρους να επαληθεύσουν άμεσα ολόκληρη την εκλογική διαδικασία, επιβεβαιώνοντας, έτσι, ότι καμία οντότητα (ούτε και οι αρχές) δεν έχει αλλοιώσει το εκλογικό αποτέλεσμα με κάποιο τρόπο.

Κατά τα τελευταία χρόνια, οι εξελίξεις στην κινητή τεχνολογία (και για τα δίκτυα και ειδικά για τις συσκευές), οι οποίες έφεραν υψηλότερες ταχύτητες και χαμηλότερο κόστος, έχουν οδηγήσει στη διαδεδομένη χρήση βολικών εφαρμογών διαδικτύου και κινητών. Η παρούσα πτυχιακή εργασία περιγράφει το σχεδιασμό και την υλοποίηση μιας εφαρμογής για το λειτουργικό σύστημα (ΛΣ) Android η οποία αυτοματοποιεί τη διαδικασία που οι trustees ενός καινούριου, κατανεμημένου, από-άκρο-σε-άκρο επαληθεύσιμου, ηλεκτρονικού συστήματος ψηφοφορίας πρέπει να ακολουθήσουν. Ο ευρύτερος στόχος είναι η σκιαγράφηση της αρχιτεκτονικής και η παροχή μιας αναφορικής υλοποίησης για μια αποδοτική, σταθερή και ασφαλή εφαρμογή Android ηλεκτρονικής ψηφοφορίας, όχι μόνο για ένα trustee, αλλά πιθανώς για οποιαδήποτε εκλογική οντότητα, δείχνοντας ότι οι σημερινές συνηθισμένες κινητές συσκευές είναι αρκετά ισχυρές για να χρησιμοποιηθούν αποδοτικά για τέτοιες εργασίες.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ηλεκτρονικές ψηφοφορίες

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: η-ψηφοφορίες, κρυπτογραφία, trustee, κατανεμημένα συστήματα, ανάπτυξη εφαρμογών Android

ABSTRACT

E-voting systems comprise a powerful technology that improves the election procedure by significantly increasing tallying speed and reducing election costs. Kiosk-based e-voting systems, while providing the aforementioned benefits, require the voter's physical presence (which probably introduces delays and inconvenience). On the other hand, internet voting systems allow voters to cast their votes quickly and remotely, through one's mobile phone or personal computer (a process which is also called i-voting), potentially promoting democracy by providing considerably improved accessibility for individuals that are physically hindered, thus increasing overall voter participation. On top of that, there exist internet voting systems that allow voters to directly verify the entire election procedure, thereby confirming that absolutely no entities (even authorities) have altered the election result in some way.

During the latest years, the advancements in mobile technology (for both networks and especially devices), bringing higher speeds and lower costs, has led to the widespread use of convenient web and mobile application. The present thesis describes the design and implementation of an application for the Android operating system (OS) that automates the procedure which the trustees of a new, distributed, end-to-end verifiable, internet voting system have to follow. The broader goal is to outline the architecture and provide a reference implementation for an efficient, robust and secure e-voting Android application, not just for a trustee, but possibly for any voting entity in general, showing that today's ordinary mobile devices are powerful enough to effectively be used for such tasks.

SUBJECT AREA: Electronic voting

KEYWORDS: e-voting, cryptography, trustee, distributed systems, Android development

To my family.

ACKNOWLEDGEMENTS

For the completion of this thesis, I would like to thank my supervisors, associate professor Mema Roussopoulou (Computer Systems and Applications Division, Department of Informatics and Telecommunications, National and Kapodistrian University of Athens) and Nikos Chondros, for their supervision, help and guidance, as well as for their general interest and trust they showed me by allowing me to participate in their work.

ΕΥΧΑΡΙΣΤΙΕΣ

Για τη διεκπεραίωση της παρούσας Πτυχιακής Εργασίας, θα ήθελα να ευχαριστήσω τους επιβλέποντες, αναπληρώτρια καθηγήτρια .Μέμα Ρουσσοπούλου (Τομέας Υπολογιστικών Συστημάτων και Εφαρμογών, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών) και Νίκο Χονδρό, για την επίβλεψη, βοήθεια και καθοδήγησή τους, καθώς και για το γενικότερό τους ενδιαφέρον τους και την εμπιστοσύνη που μου έδειξαν επιτρέποντας τη συμμετοχή μου στο έργο τους.

TABLE OF CONTENTS

LIST OF DIAGRAMS

LIST OF FIGURES

LIST OF TABLES

PREFACE

The present work was written in the summer semester of 2015, Athens in the context of my graduation thesis.

After Nikos Chondros and Mema Roussopoulou's suggestion, I took part in an active group of knowledgeable people who were currently developing an innovative e-voting system that introduces new features and guarantees for such systems by combining the ubiquitous field of cryptography with the blooming field of distributed systems.

Since the designed system is new and demands a considerable theoretical background, the help of my supervisors and their willingness to answer all my questions was valuable for the completion of this thesis.

1. INTRODUCTION

1.1 What is electronic voting

Electronic voting (e-voting) is a type of voting which is performed through the utilization of electronic means. Depending on the particular case, these means may vary from optical scan voting systems and voting kiosks with self-contained direct-recording electronic voting systems to private computing networks or full-function online voting through ordinary Internet connected mobile or household devices. Similarly, they can aid the election process by encompassing a range of services from ballot casting and vote counting to complete automation that includes voter authentication, casting, tallying, encryption, data transmission and verification.

1.2 What is DEMOS

DEMOS [1] is an innovative, distributed, end-to-end verifiable, internet voting system [2] developed by a team of researchers at the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens. It is the first e-voting system whose operation is human verifiable, meaning that voters can verify its proper operation without the assistance of special software or trusted devices. DEMOS empowers voters by allowing them to verify the election tally as well as their vote themselves.

1.3 What is Android

Android is an open source [3] mobile operating system, currently developed by Google. [4][5] It is based on the Linux kernel and is designed primarily for mobile devices such as smartphones and tablets. The kernel is modified to use libraries or drivers that enable the efficient execution of the Android OS on mobile devices (which are characterized by limited resources).

1.3.1 Why Android

Smartphones are quickly becoming a ubiquitous computing platform. Being open source, Android has presented a tremendous increase in its market share and has developed a large community of developers and enthusiasts, dominating Apple's iOS or Microsoft's Windows Phone. In July 2013, Android surpassed the one million applications in the Google Play store and the fifty billion application downloaded milestones. [6] What is more, at Google I/O 2014, the company revealed that there were over one billion active monthly Android users, up from 538 million in June 2013. [7] Having such an immense user base, the Android OS makes an ideal platform for any new application.

2. THE DEMOS INTERNET VOTING SYSTEM

2.1 Background

There is a specific list of requirements that an ideal system would satisfy. The internet voting system in discussion addresses some properties among the most challenging to provide, which are the following: [2]

- **End-to-end verifiability:** All voters can verify that their votes were counted towards the tally without being altered and any party can verify that the election procedure was performed as intended.
- **Privacy:** It is impossible for a party to extract information about the voters' ballots without monitoring voters during the voting phase of the election.
- **Receipt-freeness:** It is impossible for a voter to prove what they voted to any party which did not monitor them during the voting phase of the election (i.e. the system does not facilitate vote-selling).
- **Fault tolerance:** The voting system should continue to operate correctly when up to a specific number of its components are faulty.

2.2 Components of the DEMOS voting system

The voting community generally uses specific terminology and abstractions when modeling and describing voting systems. A brief description of the ones related to the system in discussion follows (as they are defined in the corresponding paper "*A distributed, end-to-end verifiable, internet voting system*" [2]).

2.2.1 Election Authority

The *Election Authority* (EA) [2] is an entity that performs the initialization of the system. For this system, the EA is modeled as a trusted and independent node that is commissioned to set up an election. It generates and transmits all initialization data to all the other components of the system and finally gets destroyed for privacy reasons. It is assumed that for the aforementioned communications *Out-Of-Band* (OOB) channels are used that are authenticated and private.

In order to generate all initialization data, the following inputs are given:

- The election question.
- The election timeframe.
- The list of options that the voter will chose from.
- The number of ballots to generate (equal to the number of eligible voters).

2.2.2 Bulletin Board

A *Bulletin Board* (BB) [2] is a publicly accessible repository of all voting-related information, assisting in tasks such as vote collection, result tabulation and election auditing. In contrast to a lot of previous works that present it as a trusted and centralized component, the BB is modeled here as a distributed entity with the following two parts:

- **Vote Collection:** The *Vote Collection* (VC) is a separate subsystem that implements the vote collection functionality of the BB. It comprises a set of cooperating nodes, with authenticated private channels between them, that collect the votes submitted by voters during election hours and post them to the ABB (discussed next) after the election has ended.
- **Audit Bulletin Board:** The *Audit Bulletin Board* (ABB) is again a separate subsystem that implements the remaining functionality of the BB. It consists of multiple independent nodes, with each one being oblivious to the existence of other ABB nodes. The ABB allows public read access and, also, authenticates all write requests from VCs and trustees (discussed next).

2.2.3 Trustees

Trustees [2] are entities whose intervention is required for the election tally to be produced and published. More specifically, they are individuals entrusted with the power to unlock ABB information at will.

This entity is the main concern of this thesis, as the app that was developed automates the procedure they perform.

2.2.4 Voters

Voters [2] have the freedom to vote from any device of their choice, whether trusted or untrusted, without sacrificing their privacy, and still be assured their vote was cast as intended, since it is assumed that the communication channel between them and any VC node is unencrypted and unauthenticated. This means that voters may use the system over the web, even when their client stack is potentially unsafe.

To achieve this degree of freedom, the voter is given a ballot with distinct vote codes assigned to the election options. In addition, a short expected receipt is assigned to each vote code. When a voter casts their vote (by casting the vote code that corresponds to their voting preference) and the distributed BB accepts it, they receive a receipt as a reply. This can optionally be used by the voter to visually match it with the expected receipt in their ballot, thus allowing them to use any terminal for casting and verify, nonetheless, the proper operation without the need of special software or trusted devices that perform cryptographic operations.

3. THE ANDROID OPERATING SYSTEM

3.1 Fundamentals

Since Android is an enormous framework that attempts to be a universal platform which covers the needs of all mobile applications and their particular use cases while supporting a wide range of devices with completely different capabilities, it is impossible to succinctly describe it in its entirety. Instead, only an overview of its essential (for this app) architectural components is provided in order to allow any reader to understand the design that was selected for this app. Developers that are interested in the implementation details and the specific Android APIs that were chosen, are referred to the application's source code [8].

Android apps are written in the Java programming language. The Android Software Development Kit (SDK) tools compile your code - along with any data and resource files - into an *APK*: an *Android package*, which is an archive file with an **.apk** suffix. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app. [9]

3.1.1 Security

Each Android app lives in its own security sandbox. By default, the Android operating system assigns a unique Linux user ID to every app, so each one is actually a different user. In addition, every app runs in its own Linux process by default and each process has its own virtual machine (VM) that runs code in isolation from other apps. Moreover, the system sets permissions for all files in an app, so that only it can access them. Therefore, an application has access only to the components in needs to do its work and no more (this is known as the principle of least privilege). [9][10]

3.2 Processes and Threads

When an application component starts and the application does not have any other components running, the Android system starts a new Linux process for the application with a single thread of execution. By default, all components of the same app run in the same process and thread (called the "main" thread). If an application component starts and there already exists a process for that app (because another component from the app exists), then the component is started within that process and uses the same thread of execution. [10]

Nevertheless, an application developer can (and usually does) create additional threads for any process, in order to execute concurrent tasks, avoid blocking the UI or take advantage of devices with multi-core processors. They may, also, arrange for different application components to run in separate processes (even though most applications should not change this). [10]

In order to provide the best user experience, the Android system tries to maintain an application process in memory for as long as possible, so as to quickly resume it when needed, but it eventually needs to remove older processes to reclaim memory for new or more important ones. To make these decisions, the system creates an "importance hierarchy" based on the state of the components running in each process, with processes with the lowest importance being eliminated first. [10]

There are five levels in this importance hierarchy. These are the following, starting from the most important: [10]

1. **Foreground process:** A process that is required for what the user is currently doing.
2. **Visible process:** A process that doesn't have any foreground components, but still can affect what the user sees on screen.
3. **Service process:** A process that is running a service and does not fall into either of the two higher categories.
4. **Background process:** A process holding an activity that's not currently visible to the user.
5. **Empty process:** A process that doesn't hold any active application components.

3.3 Components of the Android OS

App components are the essential building blocks of an Android app. Each component is a different point through which the system can enter an app. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role – each one is a unique building block that helps define the app's overall behavior. [9]

There are four different types of app components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed. [9]

3.3.1 Activities

An *activity* [9][11] is an application component that provides a single screen with which a user can interact in order to do something, such as composing and email or making a phone call. Each activity is given a window in which to draw its user interface. It represents an activity – thus, the name – that the user can perform. Note that an activity may include multiple actions, so the two terms are not used interchangeably. For instance, a client app for a cloud file storage service might have one activity that shows a list of all files and folders, another activity to create a new text file and another activity for viewing the contents of a file. The activities of an app work together to create a cohesive user experience. It is important, nevertheless, to understand that each one is completely independent of the others. As a result, an app may permit all other apps to start one of its activities (without prior negotiations). For example, a social networking app may start the activity in a camera app that captures a photo, in order for the user to share it with their friends.

As mentioned before, each activity can start another activity. When a new activity starts, the previous one is stopped and preserved in a stack (that implements the known "last in, first out" mechanism), while the new one is pushed onto the stack. When the user returns to the previous activity (usually by pressing the *Back* button), the current activity is popped from the stack, destroyed, and the previous activity is resumed.

It is now clear that an activity can exist in many states, depending on its interaction with other activities and the user. These are essentially the following three states:

1) **Resumed**: The activity is running in the foreground and has user focus.
2) **Paused**: Another activity is in the foreground and has user focus, but the one in discussion is still visible (because the activity in the foreground doesn’t cover the entire screen or is partially transparent).

3) **Stopped**: The activity is completely hidden by another activity (and is now in the "background").

It is crucial to realize that the Android OS was designed for devices with limited resources, so it is possible (and likely) that when the system finds itself in a low memory situation, it will drop a paused or stopped activity from memory either by asking it to finish or by simply killing its process. Consequently, it is essential to always consider this fact when designing a robust app and managing the lifecycle of its activities. In order to do so, the system provides the following fundamental callback methods (as shown in the skeleton activity taken from [11]):

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be "paused").
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

The following table (reproduced from [12]) gives a more detailed description of each callback method:

Table 1: A summary of the activity lifecycle's callback methods [12]

Method	Description	Killable?	Next
`onCreate()`	Called when the activity is first created. This is where you should do all of your normal static set up: create views, bind data to lists, etc. This method also provides you with a Bundle containing the activity's previously frozen state, if there was one. Always followed by `onStart()` .	No	`onStart()`
`onRestart()`	Called after your activity has been stopped, prior to it being started again. Always followed by `onStart()`	No	`onStart()`
`onStart()`	Called when the activity is becoming visible to the user. Followed by `onResume()` if the activity comes to the foreground, or `onStop()` if it becomes hidden.	No	`onResume()` or `onStop()`
`onResume()`	Called when the activity will start interacting with the user. At this point your activity is at the top of the activity stack, with user input going to it. Always followed by `onPause()` .	No	`onPause()`
`onPause()`	Called when the system is about to start resuming a previous activity. This is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, etc. Implementations of this method must be very quick because the next activity will not be resumed until this method returns. Followed by either `onResume()` if the activity returns back to the front, or `onStop()` if it becomes invisible to the user.	**Pre-** HONEYCOMB	`onResume()` or `onStop()`
`onStop()`	Called when the activity is no longer visible to the user, because another activity has been resumed and is covering this one. This may happen either because a new activity is being started, an existing one is being brought in front of this one, or this one is being destroyed. Followed by either `onRestart()` if this activity is coming back to interact with the user, or `onDestroy()` if this activity is going away.	**Yes**	`onRestart()` or `onDestroy()`
`onDestroy()`	The final call you receive before your activity is destroyed. This can happen either because the activity is finishing (someone called `finish()` on it, or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the `isFinishing()` method.	**Yes**	*nothing*

Finally, the following diagram (as taken from [11]) indicates the transitions in the activity lifecycle:

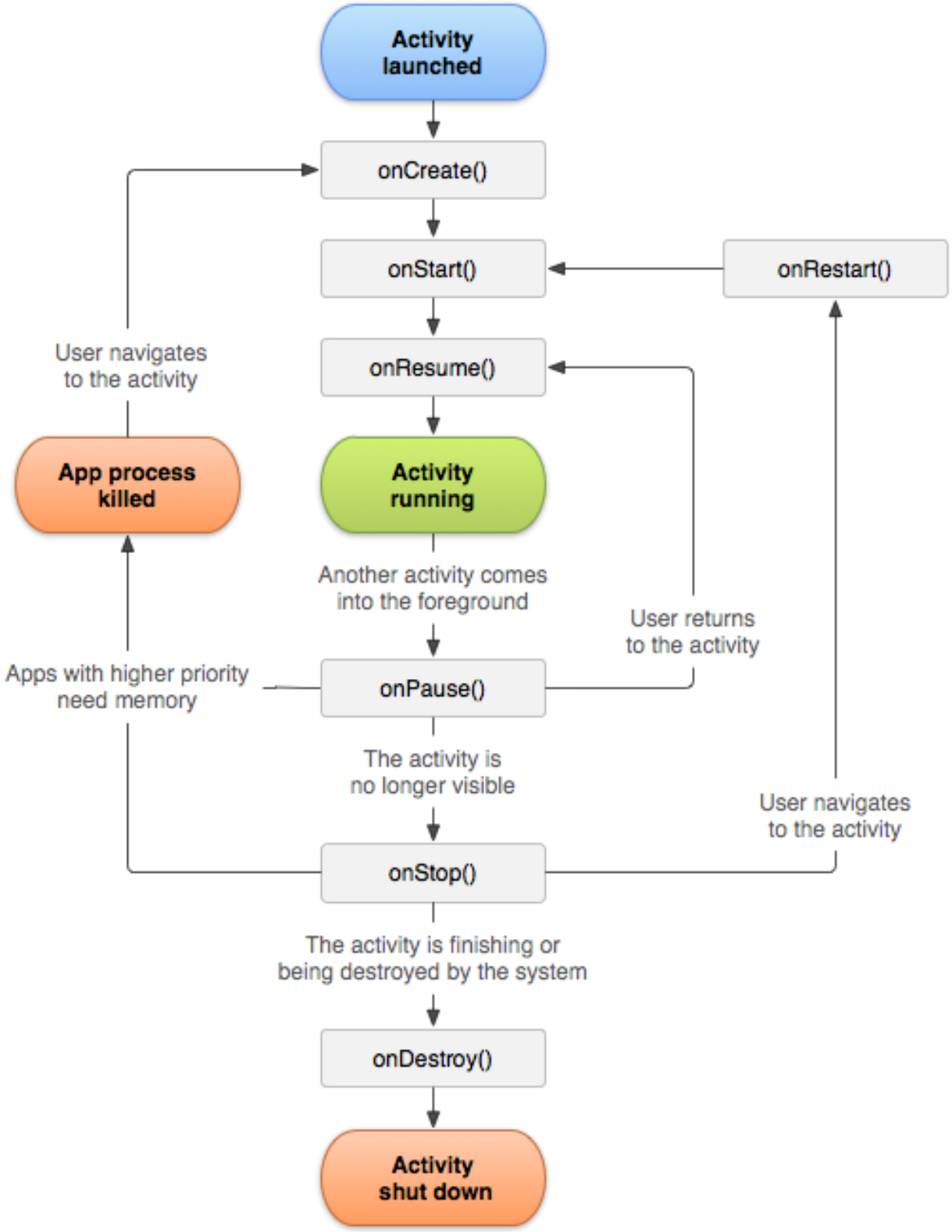

Diagram 1: The paths that an activity might take between its states [11]

3.3.2 Services

A *service* [9][13] is an application component that does not provide a user interface but runs in the background. It can be used to perform long-running operations or work for remote processes. For instance, a service might play music in the background while the

user interacts with different apps, or it might fetch data over the network allowing the user to freely interact with an activity. Services are usually started by an activity and may run until either they stop themselves or another component (such as the activity that initiated them) stops them, so they will continue to run in the background even if the user switches to another application. Even though they do not display a user interface, they may create status bar notifications to alert the user of events that require their attention.

There are two types of services:

- **Started:** A “started” service is one that is launched by an app component, such as an activity, and can run in the background indefinitely, even if the component that started it is destroyed. It usually performs a single operation and stops itself when the work is done.
- **Bound:** A “bound” service is one that has app components bound to it. It offers a client-server interface that allows components to send requests to it and receive results. A bound service runs as long as other components are bound to it (whether one or many at the same time) and when all of them unbind, the service is destroyed.

It should also be noted that a service can be both started and bound. Regardless of its type, though, a service simply declares the need to execute work in the background. It always runs in the main thread of its hosting process and it never creates threads on its own. Such decisions are left to the application developer.

As with activities, it is possible that the system may decide to kill a service when it is low on memory (but it will generally avoid doing so). Like an activity, a service has lifecycle callback methods, which are demonstrated with the following example (that was taken from [13]):

```
public class ExampleService extends Service {
    int mStartMode;       // indicates how to behave if the service is killed
    IBinder mBinder;      // interface for clients that bind
    boolean mAllowRebind; // indicates whether onRebind should be used

    @Override
    public void onCreate() {
        // The service is being created
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // The service is starting, due to a call to startService()
        return mStartMode;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
        return mAllowRebind;
```

```
    }
    @Override
    public void onRebind(Intent intent) {
        // A client is binding to the service with bindService(),
        // after onUnbind() has already been called
    }
    @Override
    public void onDestroy() {
        // The service is no longer used and is being destroyed
    }
}
```

In this thesis, we will only focus on started services, since these are the ones that are used in our app. The following diagram (as taken from [13]) shows the transitions in the service lifecycle:

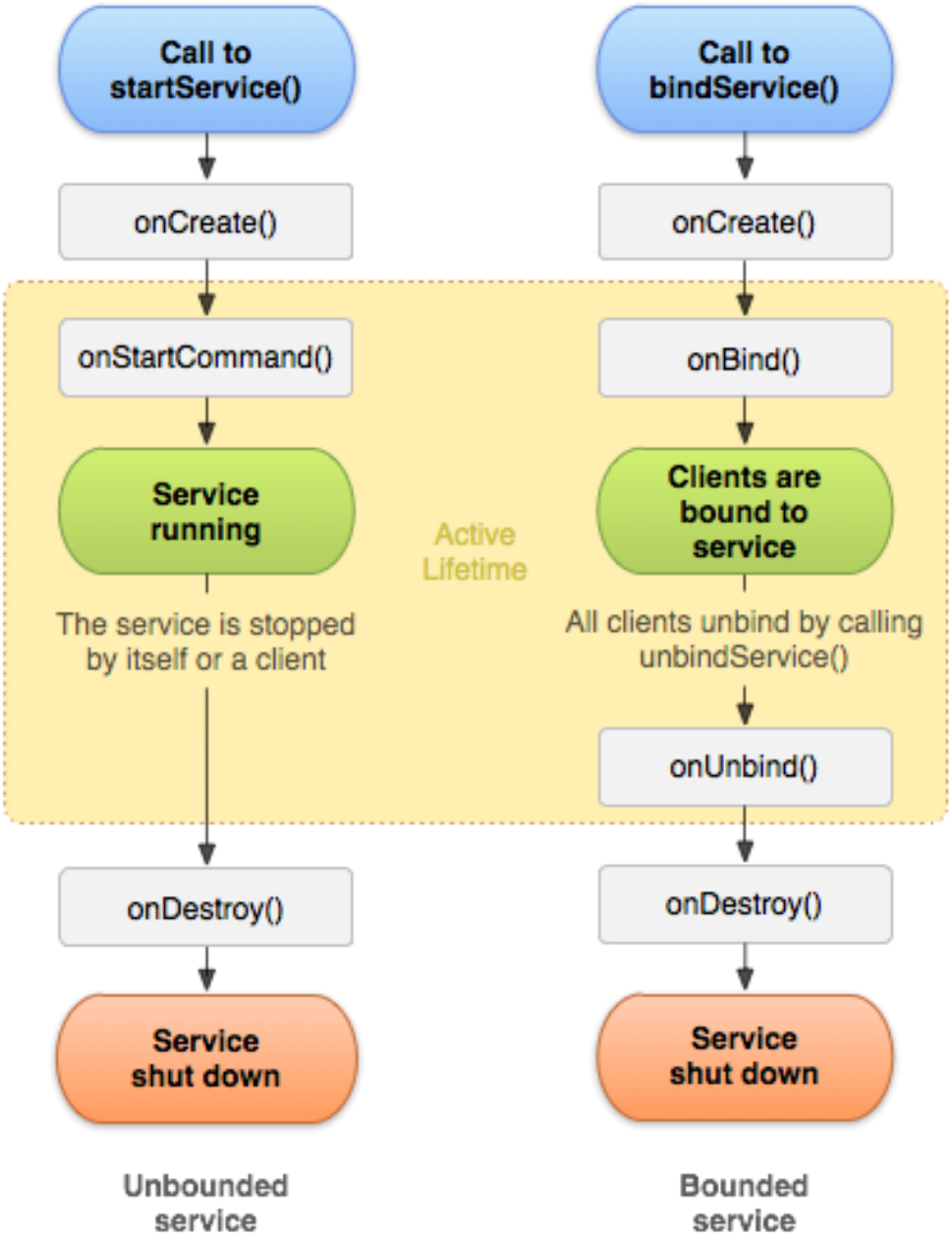

Diagram 2: The service lifecycle (we focus on unbounded services) [13]

3.3.3 Broadcast receivers

A *broadcast receiver* [9][14] is a component that responds to (receives) system-wide broadcast announcements. A broadcast may originate from the system (for instance, a broadcast announcing that the battery is low) or from an app (for example, to let other apps know that some data is available for them to use). Broadcast receivers are usually used to do a very minimal amount of work, which is commonly to start another application component, such as a service to perform some work based on the event. Like services, these components do not display a user interface and they may, also, create status bar notifications.

3.3.4 Content providers

A *content provider* [9][15] is a component that manages a shared set of app data. It enables developers to store data in the file system, an SQLite database, the web or any other persistent storage location that the app can access, while allowing other apps to query or even modify the data (if permitted) through a uniform and standard set of APIs. For example, the Android system provides a content provider that manages the user's contact information. In contrast to the previous components, no content provider is actually used in the current version of our app.

3.4 Other Android Objects

The four app components described more extensively in the previous sections provide the skeleton of an android application. They are the major parts that can interact with each other and define the basic behavior of an app. It is natural, however, for the Android framework to provide a vast variety of objects and utilities that the major components can use to provide greater flexibility and functionality. A very brief summary of some of the most significant parts for our application is given below.

3.4.1 Adapters

Adapters [16][17] are objects that take data from an underlying source (commonly an array or a database cursor) and covert them to a form (more accurately a view [18]) that will be used in the UI. In addition, they allow the efficient manipulation and display of large collections of data by recycling user interface components. Finally, they allow observers to monitor the underlying data and update them when a change has occurred, enabling, therefore, dynamic updates of the UI.

3.4.2 SQLite

SQLite [19] is an embedded, transactional, SQL database engine. SQLite provides an elegant and efficient way of organizing and saving structured data in permanent storage.

Android provides full support for SQLite databases [20], so this is an excellent choice to store, process and query data that cannot (due to size) or should not (due to possible loss) be stored in memory.

3.4.3 Loaders

Loaders [21] are objects that perform asynchronous loading of data. Loaders define a standard interface for performing longer-running operations (e.g. that load data from a file or more commonly a database). Being asynchronous, they can monitor the source of their data and deliver new results when the content changes or even offload the loading task to a separate background thread, so that it does not block the application's UI.

3.4.4 Android NDK / Java JNI

The Android *Native Development Kit* (NDK) [22] is an extension of the Android SDK that allows developers to implement parts of their app using native code, in languages such as C, C++ or even assembly. It provides toolchains for compiling code for all architectures that the Android OS currently supports (i.e. ARM, MIPS and Intel for 32 and 64bit systems).

The NDK is typically used for CPU-intensive applications – such as game engines, physics simulation or cryptographic protocols – or for reusing existing code libraries written in those languages.

The NDK makes use of the *Java Native Interface* (JNI). [23] JNI defines a way for managed code (written in the Java programming) to interact with native code (written, for example, in C/C++).

4. THE ARCHITECTURE OF THE TRUSTEE APPLICATION

4.1 Election phases

From the point of view of a trustee, there are the following phases that an election can be in, in this specific order:

1. **Initialized:** The trustee has processed the initialization data that the EA has generated for the election and awaits the end of the election in order to proceed to its verification.
2. **Verified:** After the election has ended, the trustee has retrieved and verified the results from the ABB.
3. **Completed:** The trustee has published the results to the ABB.

These phases are stored and used in the app to determine following actions (if any).

4.2 The recommended approach

The application follows the architecture that Android engineers recommend for such use cases. An overview of the approach follows, as described in the *Google I/O 2010* session by *Virgil Dobjanschi* (an, at the time of the session, software engineer at Google). [24][25]

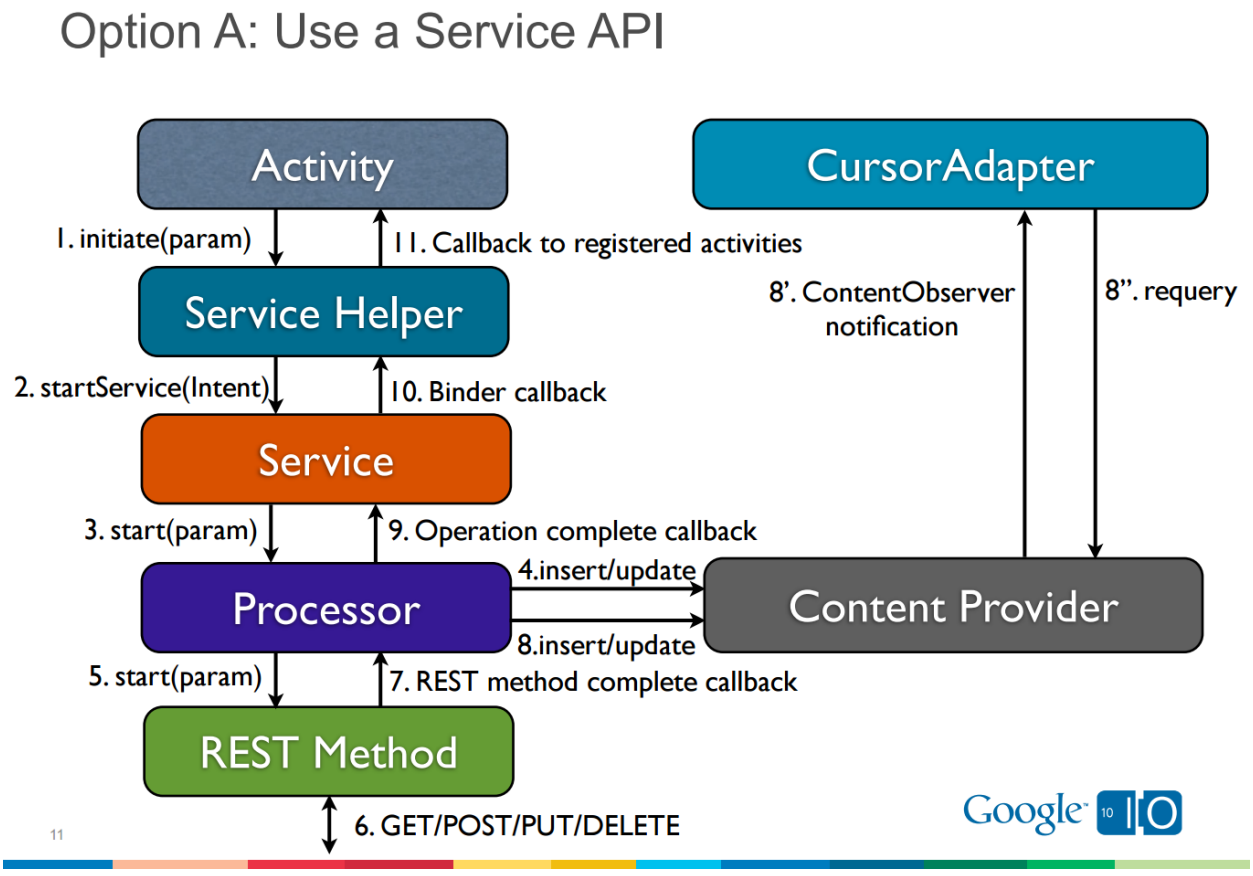

Diagram 3: One of the architectural patterns for an Android REST client application [24]

1. The **REST method** is an entity that prepares an HTTP URL (and HTTP request body if needed), executes the HTTP transaction and processes the HTTP response. The app has a few such entities, each one modeled as a task (a

Runnable) that is submitted for execution in a thread pool. In order to communicate with the ABB, the trustee uses an HTTP server with a REST API. The data are currently transmitted with a simple text format.

2. The **Processor** is an entity that mirrors the state of server resources in the app's local database. It also stores additional flags that indicate the status of an operation and may perform caching so as not to continuously retrieve the same data from the server. In our app, the processor is not a separate entity, but is embedded in the previously mentioned tasks. In addition, instead of a content provider, a raw SQLite database is used.
3. The **Service** (one of its four components that were described in the previous chapter) is the major component of this architectural pattern. The service handles operations in the background. In our app, it creates one or more worker threads in a thread pool (in actuality, there is more than one pool so as to execute different types of tasks), executes tasks according to the requests it receives, implements a queue of such requests when there are not enough threads, sends updates to the rest of the system (via local broadcast messages) and stops itself when there is no more work to do.
4. The **Service Helper** is a singleton that provides a simple asynchronous API to be used by the user interface and is implemented as such in the trustee app.
5. An **Activity** initiates requests to the service through the service helper and registers callbacks in order to receive the results of those requests. Note that instead of a cursor adapter, our app currently uses a custom adapter.

4.2.1 Benefits of the recommended approach

The design outlined solves all the problems that a wrong approach may encounter, while remaining efficient. Its major advantages are (as mentioned less specifically in [24]):

- The Android operating system was designed to work on devices with limited memory, so it may shut down the application process. This pattern accounts for that by storing data persistently in the local database.
- The pattern avoids wasting CPU resources and more significantly battery power or network bandwidth, by caching data in the local database.
- The pattern makes sure that there is consistency between the app and the server.
- The user is able to navigate away from the app and use their phone however they desire while a request is being processed.

4.3 SQLite

The app can manage multiple elections at the same time. For each election, it receives a large amount of structured data from the EA. The bigger an election is (i.e. the more eligible voters participating), the larger the dataset will be. As a result, it is generally impossible or inappropriate to store the data in memory. Therefore, SQLite was chosen to save the initialization data to disk. Apart from permanent storage, SQLite offers the following guarantees and features that make its use valuable:

- Data integrity and transactions. Even when an application is killed by the OS or the phone battery is suddenly depleted, SQLite guarantees that the database will always be at a valid state.
- Performance. SQLite can use indexes for quickly retrieving or manipulating data. This is especially useful during election verification, when the data retrieved from the ABB are checked against the initialization data from the EA.
- The ease of use of the SQL language.

4.4 NDK / JNI

We utilize the NDK in order to use MIRACL. *MIRACL* (Multi-precision Integer and Rational Arithmetic C Library) [26] is a cryptographic SDK that provides efficient implementations for a wide range of cryptographic algorithms. It is written in the C programming language, but also supports generation of optimal assembly language for all common architectures (which include all architectures that the Android OS currently supports).In addition, it is designed for highly constrained environments, including mobile apps.

We use MIRACL in this app for performing the elliptic-curve cryptographic operations of the e-voting system.

4.5 Google Protocol Buffers

Protocol buffers (also referred to as *protobuf*) [27] are an extensible, language and platform neutral, mechanism created by Google for serializing structured data. They provide a relatively simple declaration language for defining the structure of your data and then generate code in the C++, Java or Python programming languages for reading and writing from and to streams.

The initialization data generated from the Election Authority for a trustee are encoded with protobuf in order to store and transmit them efficiently, so the app uses the same format for initializing an election.

5. BENCHMARKING

Some basic time measurements were performed for a single election (for both initialization and verification) when using the Android Emulator [28] and a real device.

The Android emulator was running an x86 Android 5.1 Lollipop (API 22) system image on an x86-64 Windows 7 desktop computer. Under this configuration, the emulator was hardware accelerated (for both the CPU and GPU) with 1536 MBs of RAM available. Since it lets us simulate various network transfer rates and latency levels [28], the network speed and network delay were set to those of a GPRS network (in contrast to a modern UMTS or HSDPA network), so as to test the application in an environment more typical of the actual conditions in which it will run.

The Android device was an Archos 40 Titanium with an ARMv7-A Dual-core 1.3 GHz Cortex-A7 CPU and 512 MBs of RAM, running Android 4.2.2 Jelly Bean (API 17). [29] Note that it is not possible to throttle network traffic for a real device through the Android development tools, so the results cannot be directly compared.

The measurements in discussion follow. It should be highlighted that these numbers are given as a proof of concept, simply to show that a mobile device can handle the load of such tasks and complete them in a reasonable amount of time, but are otherwise not representative of other configurations. Since the Android OS supports a multitude of devices (both phones and tablets), it is impossible to extrapolate the app's exact performance for other scenarios.

All results are for a release build of the app and were rounded to one decimal place.

Table 2: Elapsed time for election operations in the emulator

Ballots (10 options)	Initialization	Verification
1.000	4.0s	0.4s
10.000	45.0s	2.8s
50.000	227.5s (3m 47.5s)	11.8s
100.000	434.4s (7m 14.4s)	23.9s

Table 3: Elapsed time for election operations on a real device

Ballots (10 options)	Initialization	Verification
1.000	6.3s	0.6s
10.000	69.7s	4.2s
50.000	370.3s (6m 10.3s)	18.4s
100.000	735.1s (12m 15.1s)	37.8s

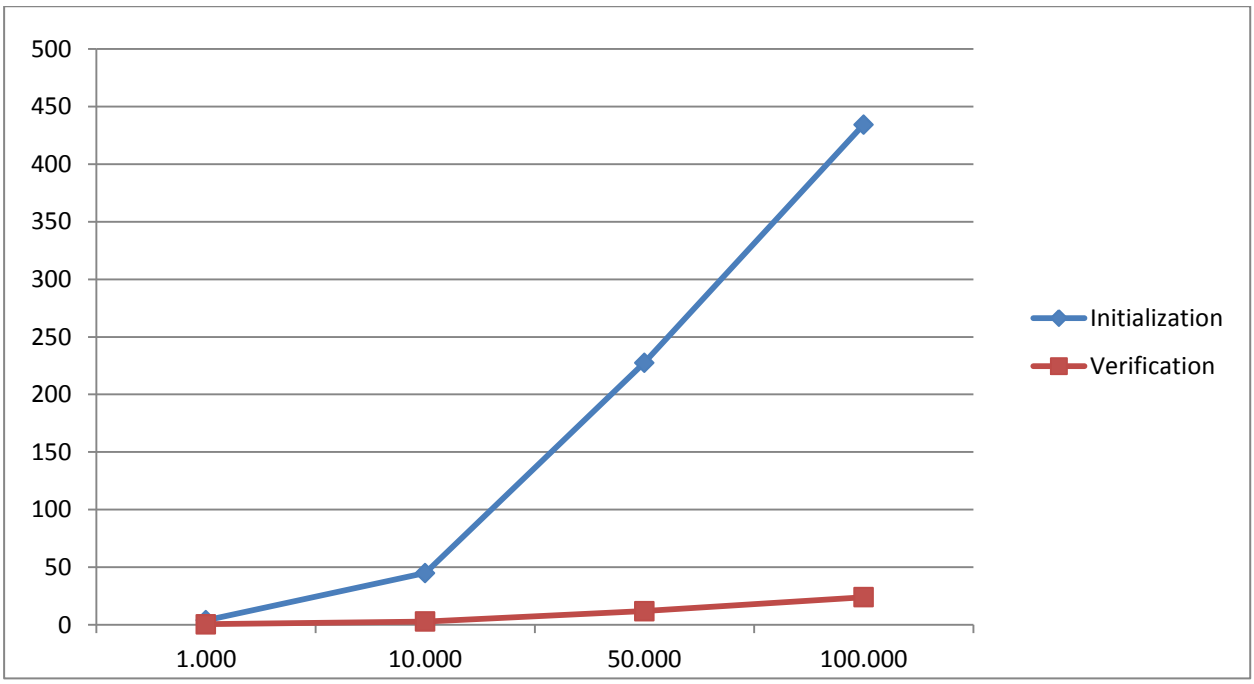

Diagram 4: Elapsed time for election operations in the emulator

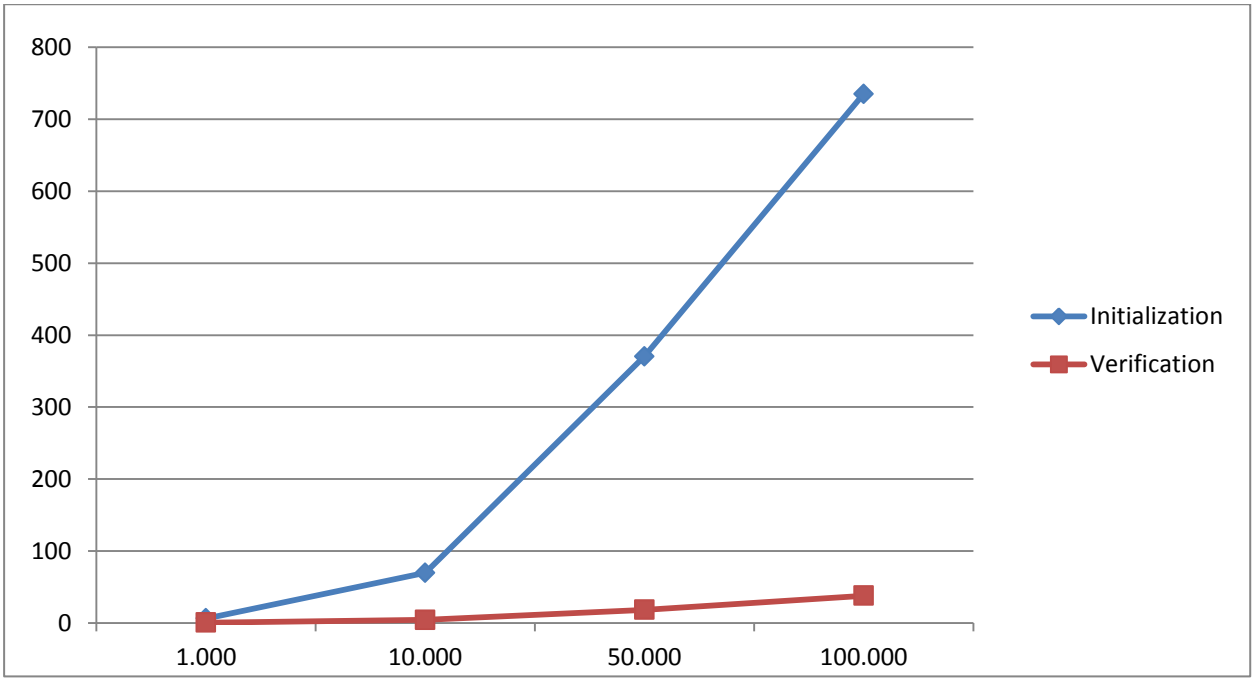

Diagram 5: Elapsed time for election operations on a real device

Some further remarks:

- It was expected that verification would be faster, because only the vote codes that are actually voted are processed (in contrast to initialization where all vote codes – for all options for both parts – are processed).
- It is noted that for *100.000* ballots with 10 options on each part of the ballot, the database uses around *225MB* of data. Therefore, the main problem for such an application is the available internal disk space on the device, rather than its computing power.

6. CONCLUSIONS

In the present thesis, we gave a brief description of the entities of the distributed, end-to-end verifiable, internet voting system and summarized the procedure that a trustee has to follow. We also described the architecture that an Android application has to implement in order to provide a pleasant user experience, while pointing out its benefits and the problems it solves.

We made clear that, despite running on hardware with limited resources, such apps can be robust and relatively fast, handling easily small, medium and even larger elections. As a consequence, a trustee partaking in an election may use their mobile Android device to automate the task at hand and generally take advantage of the conveniences such devices offer.

Finally, it should be noted that the architecture outlined can be adjusted to meet the needs of similar apps for other entities of the system. These may be most notably voters, giving them, too, the power of a mobile device, therefore, increasing voter participation even further and promoting democracy.

ABBREVIATIONS – INITIALISMS – ACRONYMS

adb	Android Debug Bridge
App	Application
ABB	Audit Bulletin Board
API	Application Programming Interface
APK	Android Package
AVD	Android Virtual Device
BB	Bulletin Board
CPU	Central Processing Unit
EA	Election Authority
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JDK	Java Development Kit
JNI	Java Native Interface
NDK	Native Development Kit
OOB	Out-Of-Band
OS	Operating System
RAM	Random Access Memory
REST	Representational State Transfer
SDK	Software Development Kit
SE	Standard Edition
SQL	Structured Query Language
UI	User Interface
URL	Uniform Resource Locator
USB	Universal Serial Bus
VC	Vote Collection
VM	Virtual Machine
ΛΣ	Λειτουργικό Σύστημα

APPENDIX I

Screenshots from a device running on the Android emulator showing the app in action.

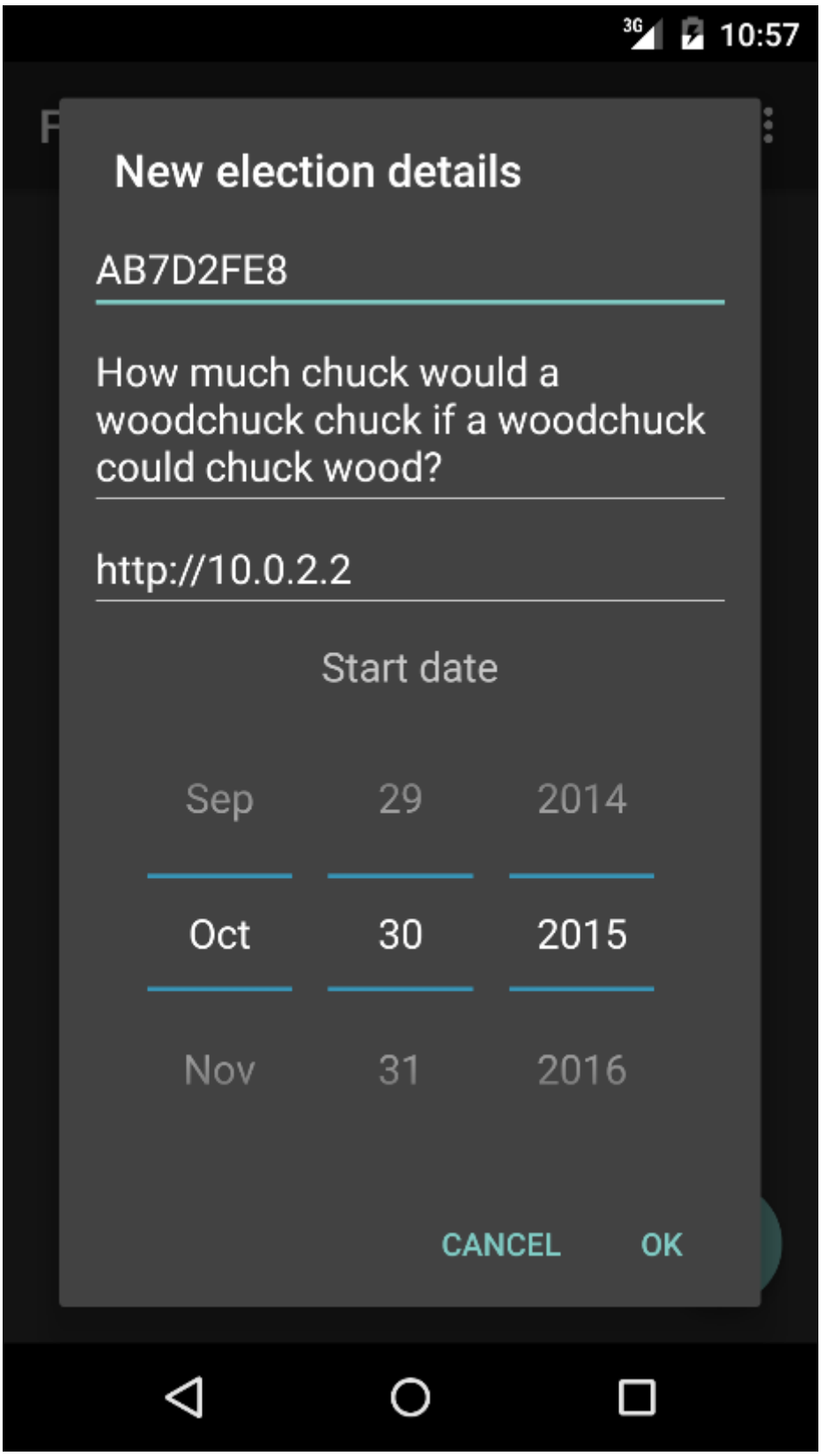

Figure 1: A dialog to insert the data of a new election

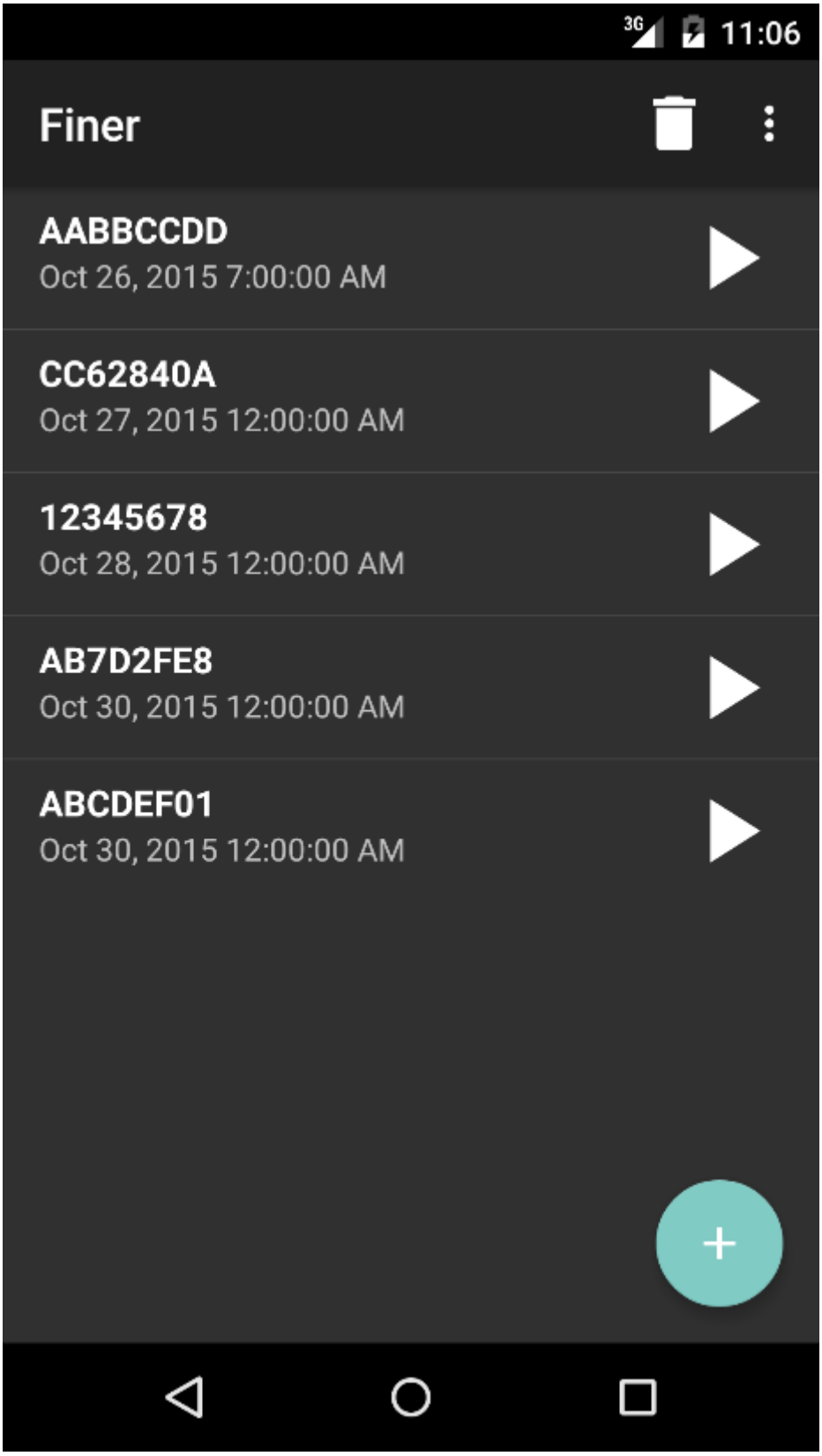

Figure 2: The main activity with a list of all the elections in the app

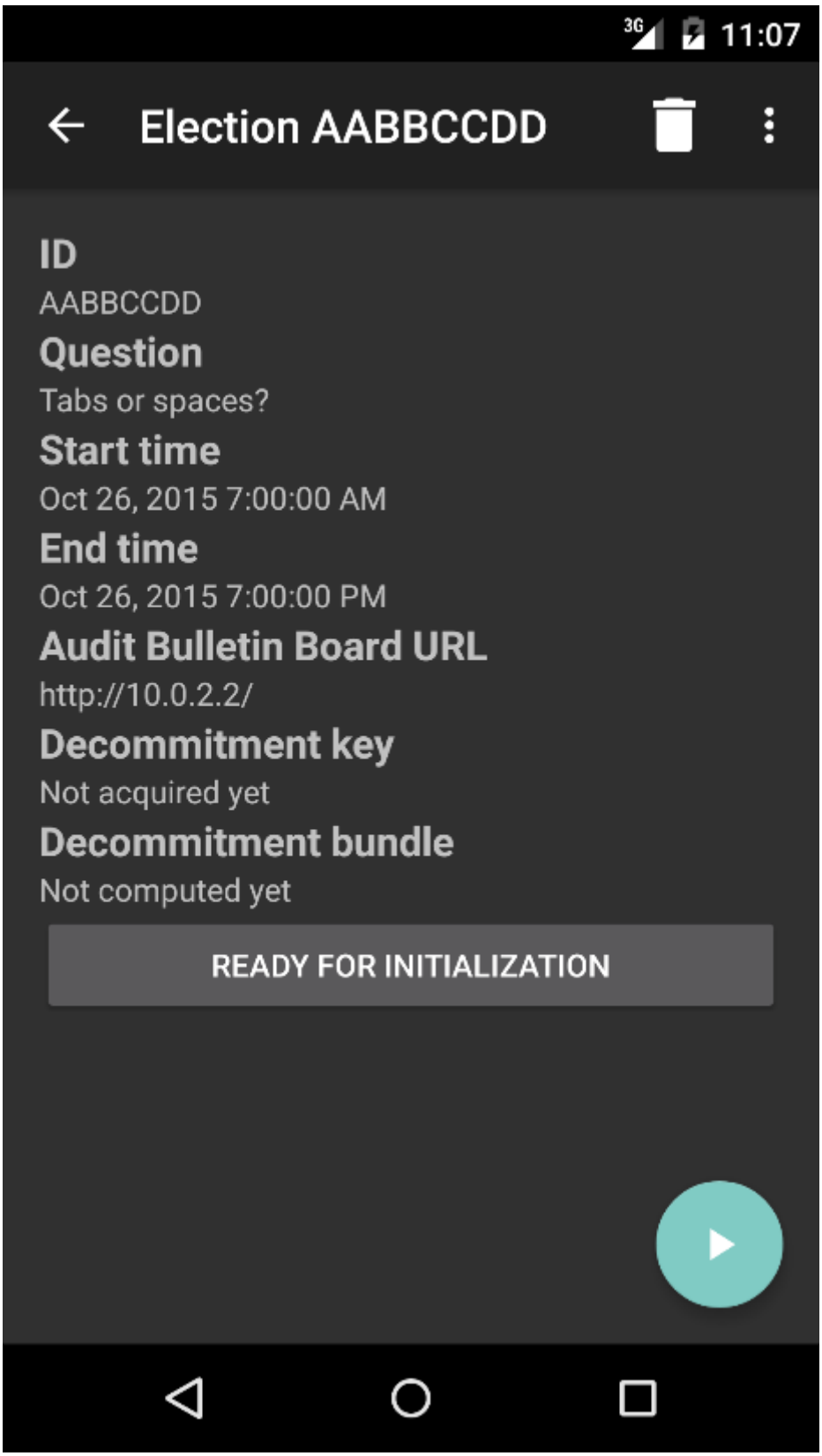

Figure 3: Another activity displaying a detailed view of the selected election

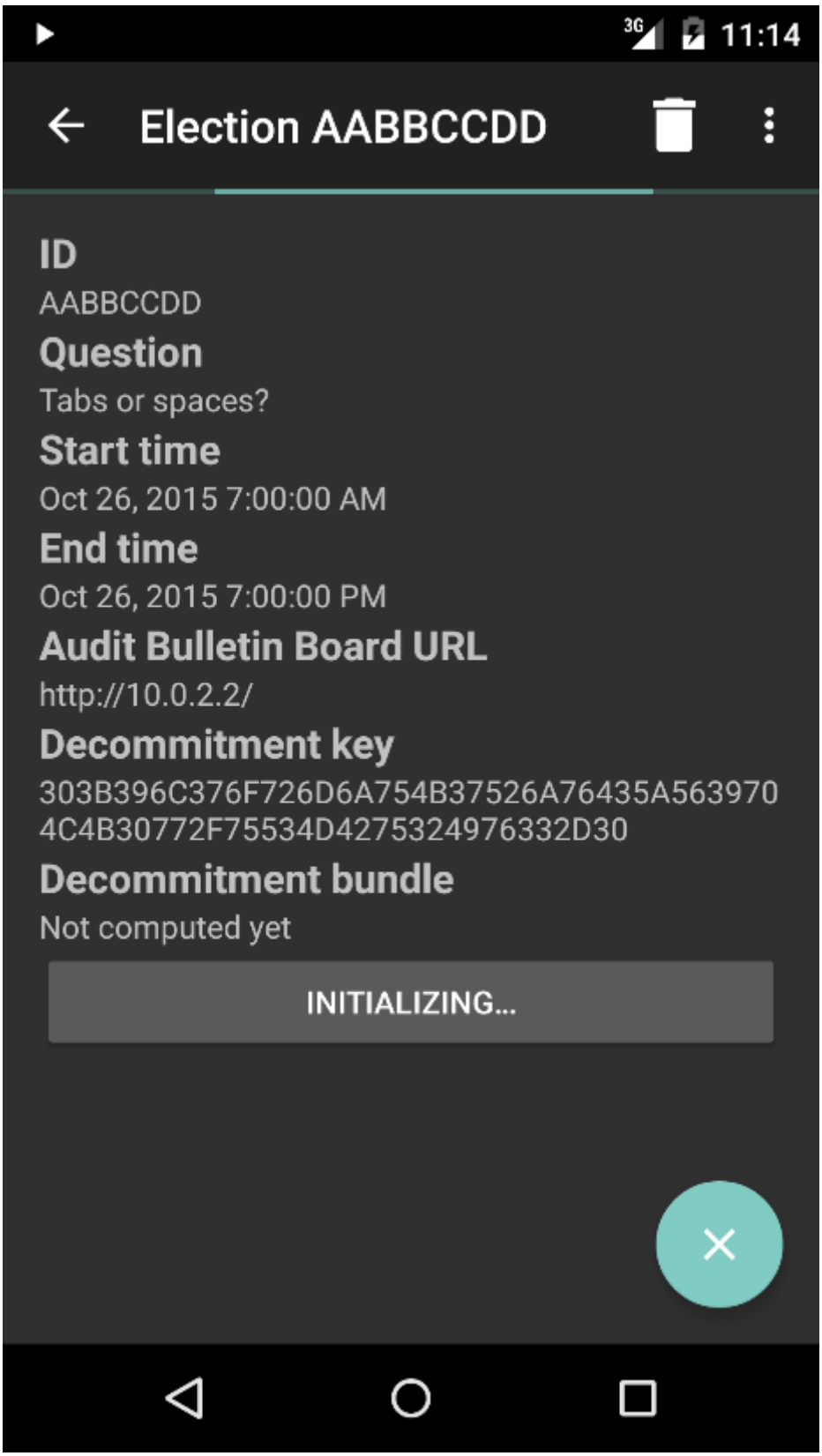

Figure 4: An election during initialization with progress indicators

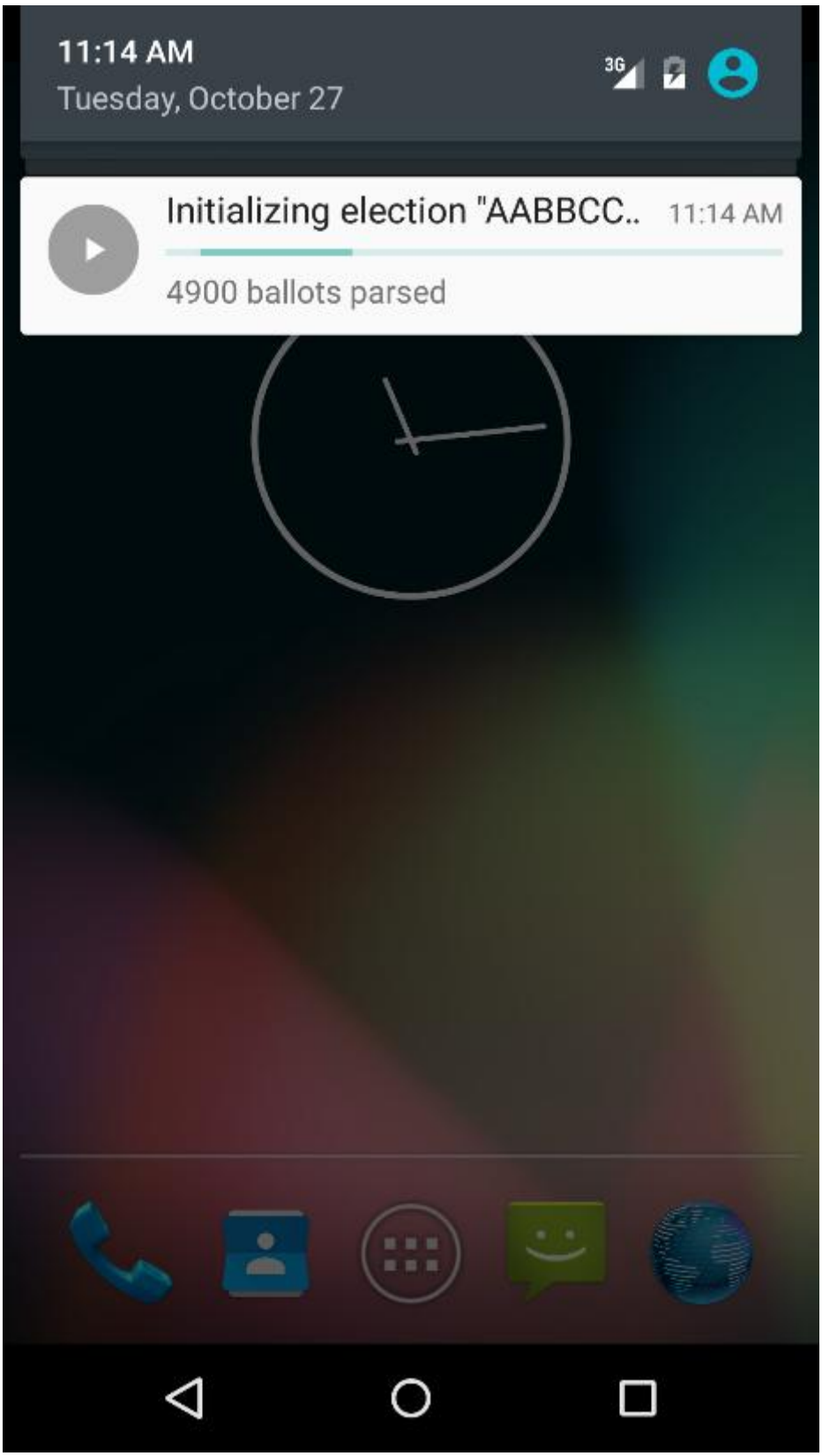

Figure 5: The user can navigate away from the app and monitor the progress

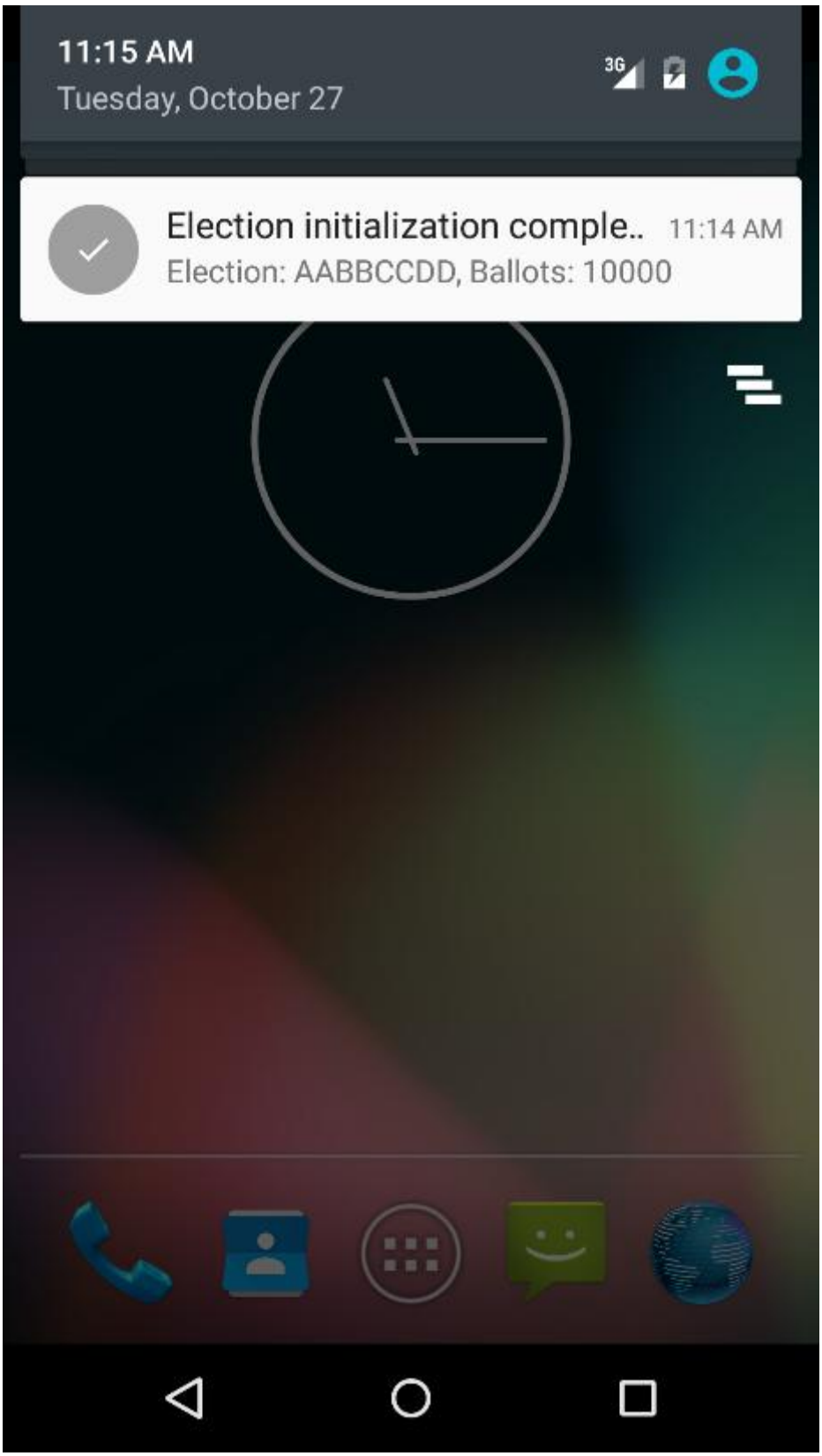

Figure 6: The user is notified when an operation has finished

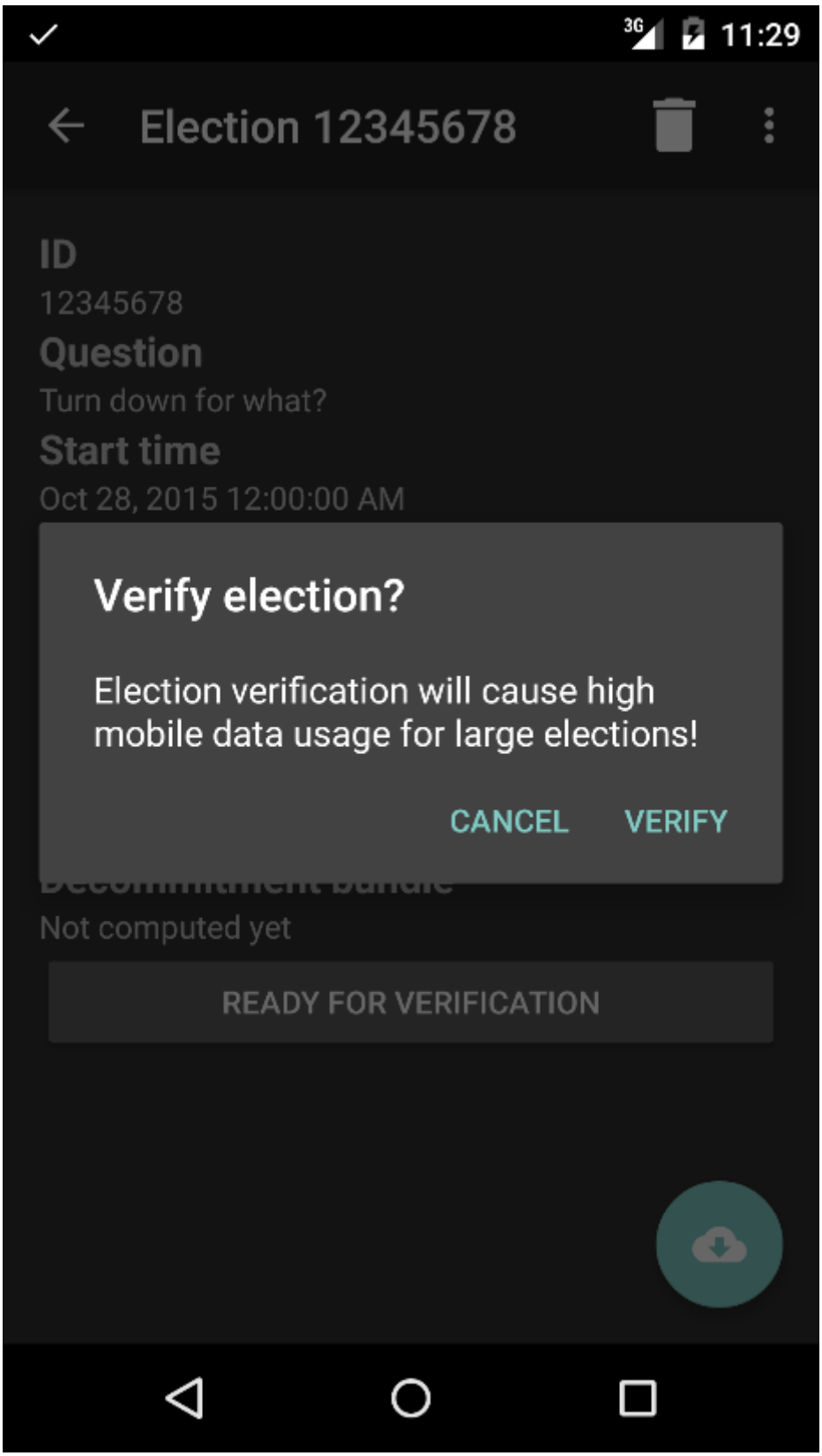

Figure 7: When on mobile data, a dialog is displayed before verifying the election

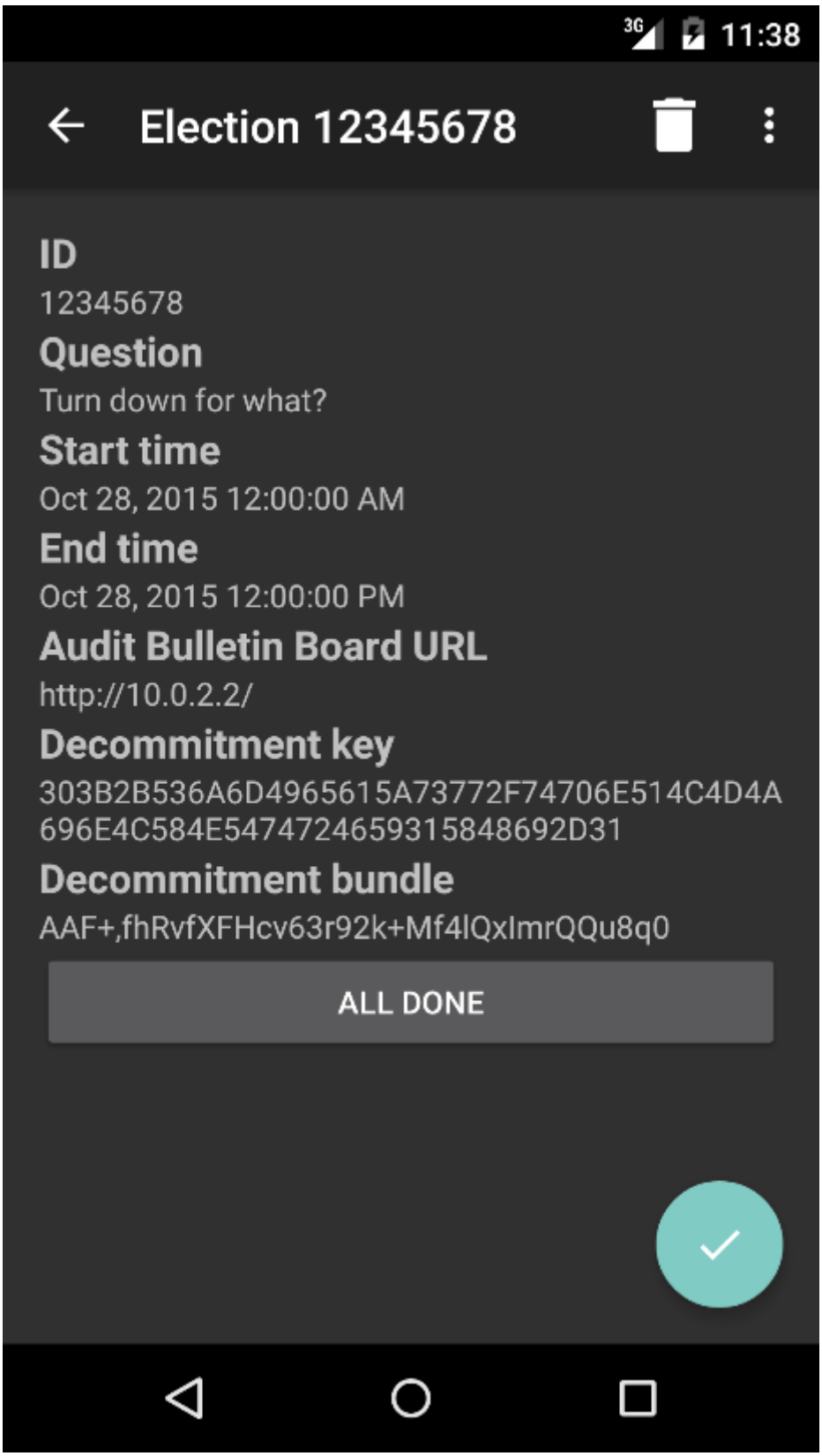

Figure 8: A completed election

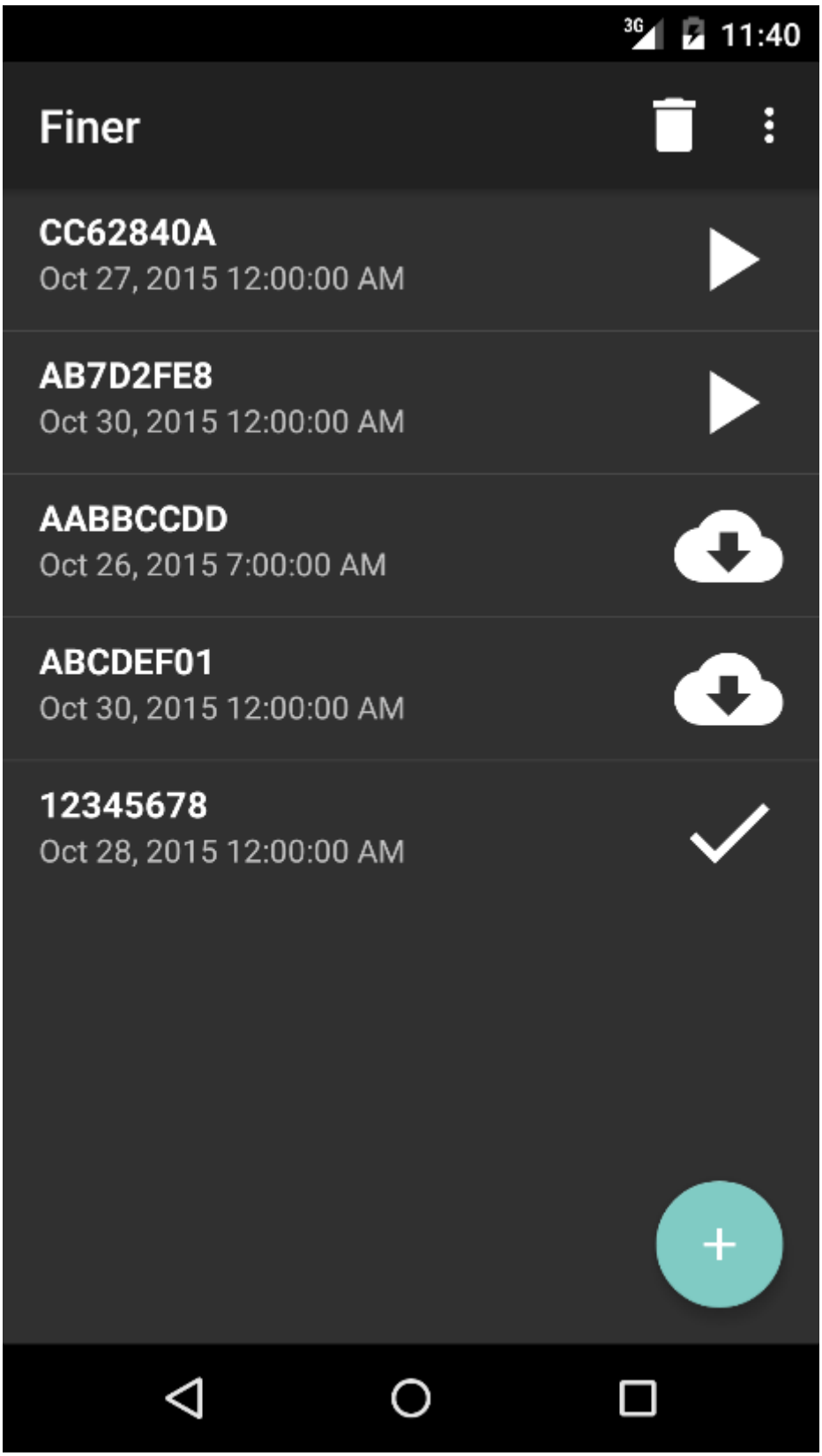

Figure 9: Elections are sorted according to their status, start time and ID

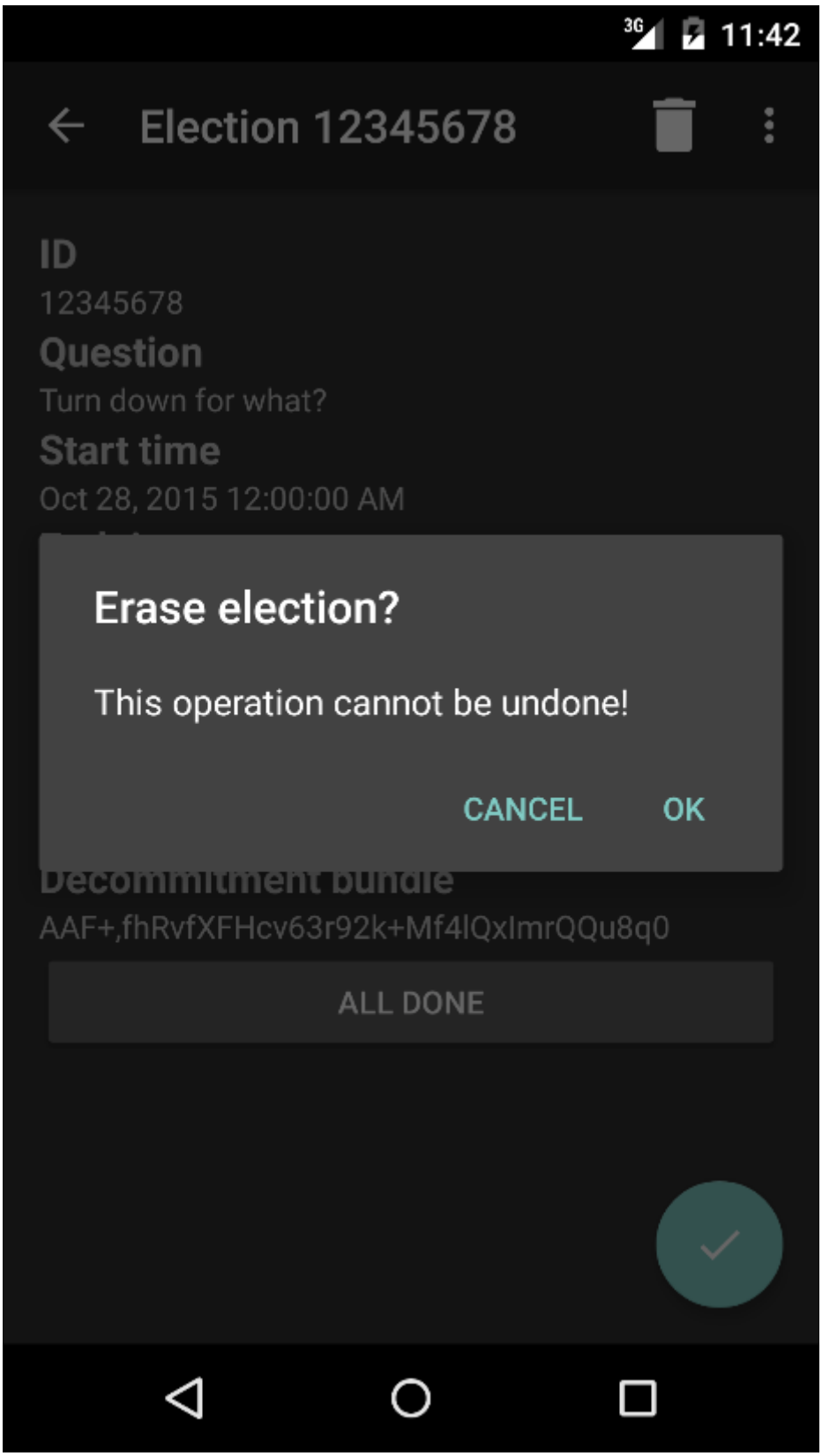

Figure 10: Erasing an election with an alert dialog preventing accidental deletion

APPENDIX II

A concise list of instructions follows for installing all the necessary tools that were used to develop and deploy the Android application. Since a detailed list of instructions for all operating systems would be rather long, the reader is often referred to the relevant online documentation.

1. Since Android applications are programmed in the Java programming language, it is necessary to install the *Java Development Kit (JDK)* of *Java Standard Edition (SE)*, before installing all of the other tools. At least Java SE version 7 is required. The JDK can be found in [30], whereas installation instructions can be found in [31]. Note that the installation procedure may differ significantly between operating systems. At the time of writing, the latest version was Java SE 8u66, while Java SE 7 was no longer supported.
2. *Android Studio* [32] is the current official Android Integrated Development Environment (IDE), – succeeding Eclipse with ADT – developed by Google and JetBrains, and built on IntelliJ IDEA Community Edition, the popular Java IDE by JetBrains [33]. It is the de facto IDE for developing Android applications. It can be downloaded (along with the Android SDK) from [32]. At the time of writing, the latest version of Android Studio was 1.4.1
3. The *Android SDK* [32] contains all the necessary command line tools for Android development. It is bundled with Android Studio, for an easier download and installation. If you do not intend to use the command line or build scripts (as was done here), then this is the recommended approach. Otherwise, the stand-alone SDK tools are also available in [32]. At the time of writing, the latest version was 24.4.1
4. After downloading Android Studio and the Android SDK, install them following the instruction in [34]. Note that the installation procedure for these tools may also differ significantly between operating systems.
5. By default, the Android SDK may not include everything you need to start developing. Using the Android SDK Manager (as stated in [35],[36]), you should check that the following few packages are installed and up to date: [35]
    - The Android SDK Tools (as mentioned above).
    - The Android SDK Platform-tools. At the time of writing, the latest version was 23.0.1.
    - The Android SDK Build-tools (highest version). At the time of writing, the latest version was 23.0.2.
    - One or more Android System Images. This is needed only if you are going to use an emulated device. For instance, you may choose the Android 5.1.1 (API 22) Intel x86 Atom System Image. At the time of writing, the latest system images available were for Android 6.0 (API 23).
    - The Android NDK. Its purpose and use was described in a previous section. At the time of writing, the latest version was r103.
    - The Android Support Repository and Android Support library. [37] The Support Library package is a set of code libraries that provide backward-compatible versions of the latest Android framework APIs (for devices running versions as old as Android 1.6 (API level 4), as well as some extra

features. At the time of writing, the latest versions were 25 and 23.1.1 correspondingly.

- The Intel x86 Emulator Accelerator (HAXM installer). [38][39] This is needed only if you want to use an emulated device. Without hardware acceleration, the emulator is going to be very slow or even unusable, depending on the configuration and the host machine. In such cases, it may be necessary to use a real device or an emulator other than the one that the Android SDK provides. Using a real device is discussed in more detail below. At the time of writing, the latest version was 5.5.
- The Google USB Driver (required for windows only in order to perform *adb* [40] debugging). [41] At the time of writing, the latest version was 11.

6. Install the OEM USB Driver for your device (usually required for windows only). [42] This is needed only if you want to use a real device.

After installing all the above software, you will be ready to download, install and run the application in a real or emulated device (phone or tablet).

1. Get the project from the online GitHub repository at [29].
2. Import the project to Android Studio using *File > New > Import Project*. If you have Git installed on your computer and have correctly configured Android Studio to use it, you may import the project directly from the online GitHub repository using *File > New > Project from Version Control > GitHub*.
3. Choose where you want to deploy or test the application.
   a. If you are going to use the emulator, then you need an Android Virtual Device (AVD). If you are using one of the latest versions of Android Studio, then it will have already offered to create an optimized AVD for your machine. If not, then you have to create a new one with the AVD manager [43].
   b. If you are going to use a real device, then connect it to your computer via USB. Remember that, even though it is a good idea for testing, using a real device is not necessary since you can try the app in an AVD with the emulator. Before installing the app, you need to prepare the device for that, as described in [44]. Note that the process depends on the Android version and the OS of the development machine.
4. Click on the *Run 'app'* button on the toolbar at the top of the Android Studio window. This will automatically build your app and then prompt you to select the device (real or emulated) on which you want it to install and execute it. If no device is connected and the emulator is not started, you will be prompted to launch the emulator. In any case, select the device you want and click *OK*.
5. The app will be installed and start running on the selected device.

REFERENCES

[1] DEMOS, http://www-en.demos-voting.com/ [Retrieved 26/10/2015]
[2] Nikos Chondros, Bingsheng Zhang, Thomas Zacharias, Panos Diamantopoulos, Stathis Maneas, Christos Patsonakis, Alex Delis, Aggelos Kiayias, Mema Roussopoulos, “A distributed, end-to-end verifiable, internet voting system”, http://arxiv.org/abs/1507.06812 [Retrieved 26/10/2015]
[3] Open Source Initiative, http://opensource.org/ [Retrieved 26/10/2015]
[4] Android, https://en.wikipedia.org/wiki/Android_(operating_system) [Retrieved 26/10/2015]
[5] The Android Source Code, http://source.android.com/source/index.html [Retrieved 26/10/2015]
[6] http://www.phonearena.com/news/Androids-Google-Play-beats-App-Store-with-over-1-million-apps-now-officially-largest_id45680 [Retrieved 27/10/2015]
[7] http://www.techspot.com/news/57228-google-shows-off-new-version-of-android-announces-1-billion-active-monthly-users.html [Retrieved 27/10/2015]
[8] Trustee application GitHub repository, https://github.com/import-this/demos-trustee-android
[9] App fundamentals, http://developer.android.com/guide/components/fundamentals.html [Retrieved 28/10/2015]
[10] Processes and Threads, http://developer.android.com/guide/components/processes-and-threads.html [Retrieved 28/10/2015]
[11] Activities, http://developer.android.com/guide/components/activities.html [Retrieved 28/10/2015]
[12] Activity class, http://developer.android.com/reference/android/app/Activity.html [Retrieved 30/10/2015]
[13] Services, http://developer.android.com/guide/components/services.html [Retrieved 28/10/2015]
[14] Broadcast Receiver class, http://developer.android.com/reference/android/content/BroadcastReceiver.html [Retrieved 28/10/2015]
[15] Content providers, http://developer.android.com/guide/topics/providers/content-providers.html [Retrieved 28/10/2015]
[16] Adapter class, http://developer.android.com/reference/android/widget/Adapter.html [Retrieved 28/10/2015]
[17] “Turbo-charge your UI: How to Make your Android UI Fast and Efficient”, https://www.youtube.com/watch?v=N6YdwzAvwOA [Retrieved 28/10/2015]
[18] View class reference, http://developer.android.com/reference/android/view/View.html [Retrieved 28/10/2015]
[19] SQLite, https://www.sqlite.org/ [Retrieved 30/10/2015]
[20] Android SQLite package summary, http://developer.android.com/reference/android/database/sqlite/package-summary.html [Retrieved 30/10/2015]
[21] Loaders, http://developer.android.com/guide/components/loaders.html [Retrieved 29/10/2015]
[22] Android NDK, http://developer.android.com/tools/sdk/ndk/index.html [Retrieved 28/10/2015]
[23] JNI, http://docs.oracle.com/javase/7/docs/technotes/guides/jni/ [Retrieved 29/10/2015]
[24] “Developing Android REST client applications”, https://www.youtube.com/watch?v=xHXn3Kg2IQE [Retrieved 28/10/2015]
[25] “Developing Android REST client applications“ presentation, https://dl.google.com/googleio/2010/android-developing-RESTful-android-apps.pdf [Retrieved 28/10/2015]
[26] MIRACL, https://www.certivox.com/miracl [Retrieved 28/10/2015]
[27] Google Protocol buffers, https://developers.google.com/protocol-buffers/ [Retrieved 28/10/2015]
[28] Android Emulator, http://developer.android.com/tools/devices/emulator.html [Retrieved 28/10/2015]
[29] Archos 40 Titanium specs, http://www.gsmarena.com/archos_40_titanium-6064.php [Retrieved 14/11/2015]
[30] Java SE JDK, http://www.oracle.com/technetwork/java/javase/downloads/index.html [Retrieved 10/11/2015]
[31] Java SE installation instructions, http://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html [Retrieved 10/11/2015]
[32] Android Studio and Android SDK downloads, https://developer.android.com/sdk/index.html [Retrieved 10/11/2015]
[33] IntelliJ IDEA, https://www.jetbrains.com/idea/ [Retrieved 10/11/2015]
[34] Android Studio and Android SDK installation instructions, https://developer.android.com/sdk/installing/index.html [Retrieved 10/11/2015]
[35] Android SDK packages installation, https://developer.android.com/sdk/installing/adding-packages.html [Retrieved 10/11/2015]

[36] Android SDK Manager, https://developer.android.com/tools/help/sdk-manager.html [Retrieved 11/11/2015]
[37] Android Support Library https://developer.android.com/tools/support-library/index.html [Retrieved 11/11/2015]
[38] Emulator Hardware Acceleration, http://developer.android.com/tools/devices/emulator.html#accel-vm [Retrieved 11/11/2015]
[39] Intel Hardware Accelerated Execution Manager, https://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager [Retrieved 11/11/2015]
[40] Android Debug Bridge, http://developer.android.com/tools/help/adb.html [Retrieved 11/11/2015]
[41] Google USB Driver, http://developer.android.com/sdk/win-usb.html [Retrieved 14/11/2015]
[42] OEM USB Drivers, http://developer.android.com/tools/extras/oem-usb.html [Retrieved 14/11/2015]
[43] AVD Manager, http://developer.android.com/tools/devices/managing-avds.html [Retrieved 14/11/2015]
[44] Using Hardware Devices, http://developer.android.com/tools/device.html [Retrieved 14/11/2015]