SYSC4907 PROJECT:
SENSOR-BASED ACCESS CONTROL SYSTEM

By
Craig Shorrocks, Jessica Morris, Richard Perryman
March 2017

A Fourth Year Project Report
submitted to the Dept. of Systems & Computer Engineering
in partial fulfillment of the requirements
for the degree of
Bachelors of Engineering

Abstract

This report tells you all you need to know about something.

Acknowledgements

I would like to thank my supervisor, anyone who paid me money, gave me equipment, etc.

Contents

List of Figures

List of Tables

List of Abbreviations

AID	Application IDentifier
APDU	Application Protocol Data Unit
API	Application Programming Interface
HMAC	Keyed-Hash Message Authentication Code
JSON	JavaScript Object Notation
MVC	Model-View-Controller
NFC	Near Field Communication
RFC	Request for Comments

Chapter 1

Introduction

Give an introduction to your project. This might include:

- Motivation for your project
- Problem you are trying to solve
- Scope of your project
- Organization of your report

You should tune this appropriately for what best suits your project.

Chapter 2

The Engineering Project

2.1 Health and Safety

Using the Health and Safety Guide posted on the course webpage, students will use this section to explain how they addressed the issues of safety and health in the system that they built for their project.

2.2 Engineering Professionalism

Using their course experience of ECOR 4995 Professional Practice, students should demonstrate how their professional responsibilities were met by the goals of their project and/or during the performance of their project.

2.3 Project Management

One of the goals of the engineering project is real experience in working on a long-term team project. Students should explain what project management techniques or processes were used to coordinate, manage and perform their project.

2.4 Individual Contributions

This section should carefully itemize the individual contributions of each team member. Project contributions should identify which components of work were done by each individual. Report contributions should list the author of each major section of this report.

2.4.1 Project Contributions

Give the individual contributions of the each team member towards the project.

2.4.2 Report Contributions

Give the individual contributions of the each team member towards writing the final report.

Chapter 3

Technical Background

3.1 NFC

3.2 Cloud Computing

3.2.1 AWS

3.3 Security

3.4 Singe-board computers

3.4.1 Raspberry pi

3.4.2 Arduino

Chapter 4

Business Use Cases

4.1 Online Order Secure Pickup

4.2 Central Mail Package Pickup

4.3 Long Term Storage

4.4 Service Provider

Chapter 5

Problem Analysis and System Design

5.1 Overall System Analysis

The requirements of our system from the use cases described above lead us to a simple outline for the entire system. The main component of the system would be the cloud server, which would store information regarding users, locks, and the relationships between them. The lock hardware would need to be able to securely accept data from the user, and securely send that data to the server for verification. Since handling electronic security can be difficult to do manually, having an application that handles much of the busy work would greatly help users interact with the system.

One of the main goals of the system was to improve the reusability of the electronic keys that are created. For this reason, the idea of a key was split into two parts: an authenticator, and an identity. Users would be able to create many authenticators, which were things like a password, or a PIN. These then could be combined into an identity, which would be used to open a particular lock. This allowed making identies that reuse the same authenticator, and reusing identities for different locks.

Another goal of the system was to provide additional behaviour on top of just being able to unlock particular locks. This extra behaviour was captured in the notion of a registration, which ties one lock to one identity. Through the registration,

we were able to implement notifications to the user as well as streaming of the lock's contents.

5.2 NFC

Determining how NFC communication should take place required analysis of two hardware systems: NFC card readers, as well as mobile devices. The most desirable protocol would be able to handle the widest variety of available hardwares for the two devices. The performance considerations between modes was fairly minimal, so preference was placed on the portability of the solution.

5.2.1 Card Readers

Since NFC cards are primarily designed for NFC communications, there were few restrictions that stemmed from potential choices in card reader. Since NFC communications are specified by the ISO, most cards support enough protocols that any decision on our part would be very likely to be supported by any card that would be desirable for any other reason.

5.2.2 Mobile Devices

The two most popular operating systems for mobile devices are iOS and Android []. Since iOS devices have NFC disabled for everything except Apple Pay[], the only option that remained was Android. Apple devices would represent a large part of the potential market, so alternatives to NFC would have to be considered.

Among Android devices, there exist devices which have hardware support for NFC communications, and those which rely on host-based card emulation. Devices with hardware support have a component called a Secure Element which performs all of the communication with the external NFC terminal. Later, applications can query this element to determine the status of the transaction, as well as other data. Devices which use host-based card emulation use a software implementation of secure elements. Since host-based card emulation is done through software, it will run on

all Android devices running version 4.4 or greater[], which represents over 99% of all devices currently in use.

Android offers an API called Beam which is the only way Anndroid devices can use NFC in active mode []. Beam, however, does not support sending more than one message between devices. Since the information we are sending can be fairly large in the interest of security, this was not feasible given the restrictions of the NFC protocols we used. Further, active communications are easier to eavesdrop on, as discussed in the background section. We decided that these costs outweighed the simplicity of the Beam API, so passive communications were chosen for the implementation.

Since more than one application could potentially want to handle an NFC message, the message protocol contains a field called the Application IDentifier, or AID []. These AIDs are just large numbers which identify which application should handle the associated message. The Android operating system is responsible for delivering the APDUs to the appropriate application's service []. Reserving a particular AID costs money, so we decided that the probability of a collision ocurring when taking a public AID would be an acceptable cost for the purpose of our demonstrative implementation.

5.2.3 Protocol

Since our NFC communications may require more data than can be fit within an Application Protocol Data Units (APDUs), we required a protocol which would handle segmenting and recombining the message. APDUs are defined in ISO 7816-4 [] and are the units used by ISO 14443-4 [], which describes the transmission protocol used by NFC devices. They are restricted to 256 bytes, including headers.

To work around this, the hardware device connected to the shield maintains a buffer. Under ISO 14443-4, messages can be reliably transferred, so managing this buffer is the main consideration of our protocol. The hardware determines the maximum amount of data that can be stored in one APDU, and fills in this value into the length field of the APDU that it sends to the Android device. Then, the Android application responds with the minimum of that much data, and all of the remaining

data that it has to send. Once the hardware receives an amonut of data less than the potential maximum, it deactivates the connection. In the event that the data from the Android application fits exactly into the last message that would be sent, the protocol still works, as the application will then respond with zero data bytes.

5.3 Android

The Android application's goal was to provide the end users with an easy way of interacting with the SBACS system. Therefore the application was designed to be as simple as possible while still maintaining flexibility when dealing with the cloud server as well as the many potential hardware devices.

5.3.1 User Interface

The natural decision for designing the basic workings of the application was to use the Model-View-Controller (MVC) design pattern. This pattern separates the underlying data (model) from the display that the user sees (view) as well as the components that the user interacts with (controller). Android provides APIs to support MVC. Model objects can be simple objects, but view objects can subclass the View class or its more specific subclasses, and the controller objects can subclass the Adapter class or its subclasses.

Android applications themselves should follow the patterns set out by the standards. Activity classes represent the pages that contain the various views and controllers that users can interact with. We decided on a design where the user first encouters a login Activity, which prompts them to either sign up or log in. Once the user has logged in, we provide a hub Activity which leads to the various other Activities in the application. These other Activities show the various data related to the user, such as their authenticators and identities.

Communication between Activities is handled by a class provided by Android called an Intent. Intent objects contain information about the nature of the request to begin the new Activity. In this way, information such as the user's identification

number could be sent from the login Activity to the hub Activity, which would allow the application to correctly load the information from that user when displaying the data Activities.

5.3.2 Notifications

The design for the notifications that would inform end users of information such as their newly available registration with a particular lock was based partly on decreasing load on the server. Having the server maintain a large amount of session information as well as spending time and memory on the timers to be able to update the user of these changes was thought to be too great a cost. For this reason, it seemed simpler and more effective to have the application poll the server at a particular access point for new information.

This polling was implemented simply in the application using a service which runs in the background of the Android device. This service regularly hits the server at a particular endpoint designed for handling these notifications. Since many users may be using the application at once, it was important to consider the performance of the endpoint's code. The endpoint returns information valuable to the application for display in the notification. The notifications make use of Android notification's ability to launch an Acticity with an Intent to take the user to the appropriate Activity associated with the notification.

5.3.3 NFC Handler

As mentioned earlier, the NFC communication method that we decided on used host-based card emulation. Android provides and API to accomplish these communications, through the HostApduService class []. This class provides a framework for creating a service which runs in the background on the Android device. The service handles setting up and closing NFC communications, while also providing overrideable methods to determine what data should be sent based on the incoming message.

Services also take an Intent object to determine their purpose. This feature was used to have the login Activity start the service with the correct key information for

a user's NFC authenticator. This way, the data sent could be overridden with the user's secret key value. Since passive NFC communications are difficult to eavesdrop on, this was thought to be a safe method of conveying the secret key to the lock without requiring too much data to be sent over the slow NFC physical links. This passing of information of course was also configured to follow the protocol designed above for NFC communications.

5.4 Hardware

5.5 Cloud

5.6 Lock Demonstration

Chapter 6

System Implementation

6.1 Android

The Android application was developed using Android Studio exclusively, and is primarily composed of Java classes, XML documents, and gradle build files. Android studio was selected since it seemed to be the natural tool for writing an Android application, and didn't have any negatives with respect to our requirements. Gradle was chosen as the build tool as it is naturally integrated with Android studio.

6.1.1 Server Interfacing

Connections to the server were managed through the android Volley API. Volley provides a simple interface over which HTTP and HTTPS messages can be sent. Standard operation of Volley follows a simple procedure: first a RequestQueue is set up, and then various Requests are enqueued. These requests are then dequeued by the RequestQueue's thread, which then creates an HTTP/HTTPS connection. The response that comes back over the connection is handled by another thread.

The code for handling these responses is put in a class that extends a Response.Listener of a given type. Since these classes have just a few relatively simple methods, anonymous classes were used in all cases. Further, we decided that handling the parsing shoudl be done as safely as possible, so all responses were first parsed simply as

Strings. Then, the response was attempted to be parsed as a JSON object of the type expected from the server. If that parsing failed, the message was instead considered a error, and handled from there.

6.1.2 HMAC

Since the server used HMAC to handle security issues, the application needed to make frequent use of the headers that the server expected to handle authentication. For this reason, a helper class called HMACHelper, was created. This class provided methods that were commonly used by all requests about data particular to the current user. Most notably, HMACHelper provided a method which calculated the secret using the same algorithm that the server would use. To reiterate how HMAC functions, the body of the message is hashed using a private key shared by the server with the application. When the application sends data that requires authentication, the server checks that the secret value provided by the application matches what it calculates using the body of the message as well as the private key associated with the user. These private keys are generated by the server when the user logs in to the application, and expire over time, creating the notion of a session.

The precise algorithm used for the hashing function matches the decision made for the server. This is necessary, since otherwise the secret values would not match between the application and the server. The application relies on a version of the Password-Based Key Derivation Function algorithm using SHA256 to be available on the Android device. The only workaround for this added requirement would have been to implement the algorithm (specified in RFC 2898 with test vectors in RFC 6070) ourselves, however doing so is known to be quite dangerous as any minor error could result in a security breach. In addition, the number of iterations and the resulting length also had to be kept as the same value.

6.2 Hardware

6.3 Cloud

Chapter 7

Testing and Bug Fixes

Chapter 8

Conclusions

References

[1] T. Me and R. You, "A great result," *Wonderful Journal*, vol. 5, no. 9, pp. 1–11, 1998.

[2] J. Him and K. Her, "An even better result that you won't believe," *Best Journal Ever*, vol. 4, no. 8, pp. 55–66, 2002.

Appendix A

Extra Simulation Results

Appendix B

Review of Linear Algebra