Final Year Project Dissertation - Secure Smart Lock System

Secure Smart Lock System

March 24, 2018

1 The Problem

With smart devices being created left right and centre nowadays with very little care towards security, it is going to become more common that houses are being hacked with devices slowly controlling more and more sensitive areas of the home. An example of this would be the August lock, an idea similar to what I am trying to achieve here, however the device has multiple security flaws that were found and outlined at the security conference Defcon. This is far from the first smart home device that was found to have security issues, there have been hundreds of different WiFi Security cameras that connect themselves to the web with little care towards the security of that device which has the capability of letting someone spy on your home. Not to long ago, there was a huge attack that brought down DynDNS, and therefore all the sites that used it which included Twitter, Spotify and Reddit. This attack was using the internet connections of innocent people who happened to have insecure smart home devices in their homes. Millions of smart home devices were involved in the attack as they were hacked due to their inadequate or sometimes non-existent security.

I want to create a product unlike these other smart home devices, one that is secure enough that I would be happy to put it in my own home. At the end of the day, a camera being infected doesn't matter so much, but a system that unlocks your house needs to be bulletproof on the security front. Looking at the implementation of the August lock, it just isn't something that I would be willing to use to secure my house, yet thousands, maybe more unknowing customers have purchased the august lock and use it to secure their homes every day. I want to create something that is easy to use, and secure, something that most companies seem to have trouble with nowadays.

2 Aim of the Project - Technical

The project objective is to create a smart lock system that would be able to automate the unlocking and locking of a house's front door. The system will be comprised of multiple Raspberry Pi's with sensors connected to them to retrieve data from a variety of sources to determine whether the door should be locked or unlocked. The system will use RESTful APIs so that it can be easily extended to more devices in the future, and will ensure that all communication between devices is done so in a fully encrypted and secure manner. An example of some authentication methods that the system may use would be fingerprint, RFID/NFC, Facial Recognition, etc.

Security is key as this will be securing a home. It is imperative that all communication is done securely, and that the system is certain that the user is who they say they are before they let them into the house. I will be conducting research into the best and most accessible security methods that could be used for this project for the initial implementation and demonstration. The idea however, is that the system will be built in such a way that anyone could create a device for this system using the RESTful APIs that will be available from the core control centre in the house.

3 Aim of the Project - For the User

So security is all well and good, but the product must still be easy to use for the user. At the end of the day, a lot of users don't care about how secure a product is, and will trust that the company has put in the correct security measures. This reason alone is why there are so many unsecured smart devices out in the wild as your average person doesn't have the technical knowledge to ensure that the product is secure. The companies that make these products know this and so don't actually implement robust security, something that, in my opinion, should be regulated more closely. This product should make the users lives easier whilst still maintaining the security of any normal key based locking system.

A simple run down of what I would want from this lock for the user would be the following. They walk up to their home after having setup their system, and all they need to do to get in is tap their phone on the NFC pad in order to unlock the door, and have it lock itself again after they have entered the house, probably on a timer of some sorts. If the door fails to lock for any reason, the user should be notified (another issue with the august lock, if it fails to lock, it simply reports the door as locked, even if it isn't).

Lackner (2013)

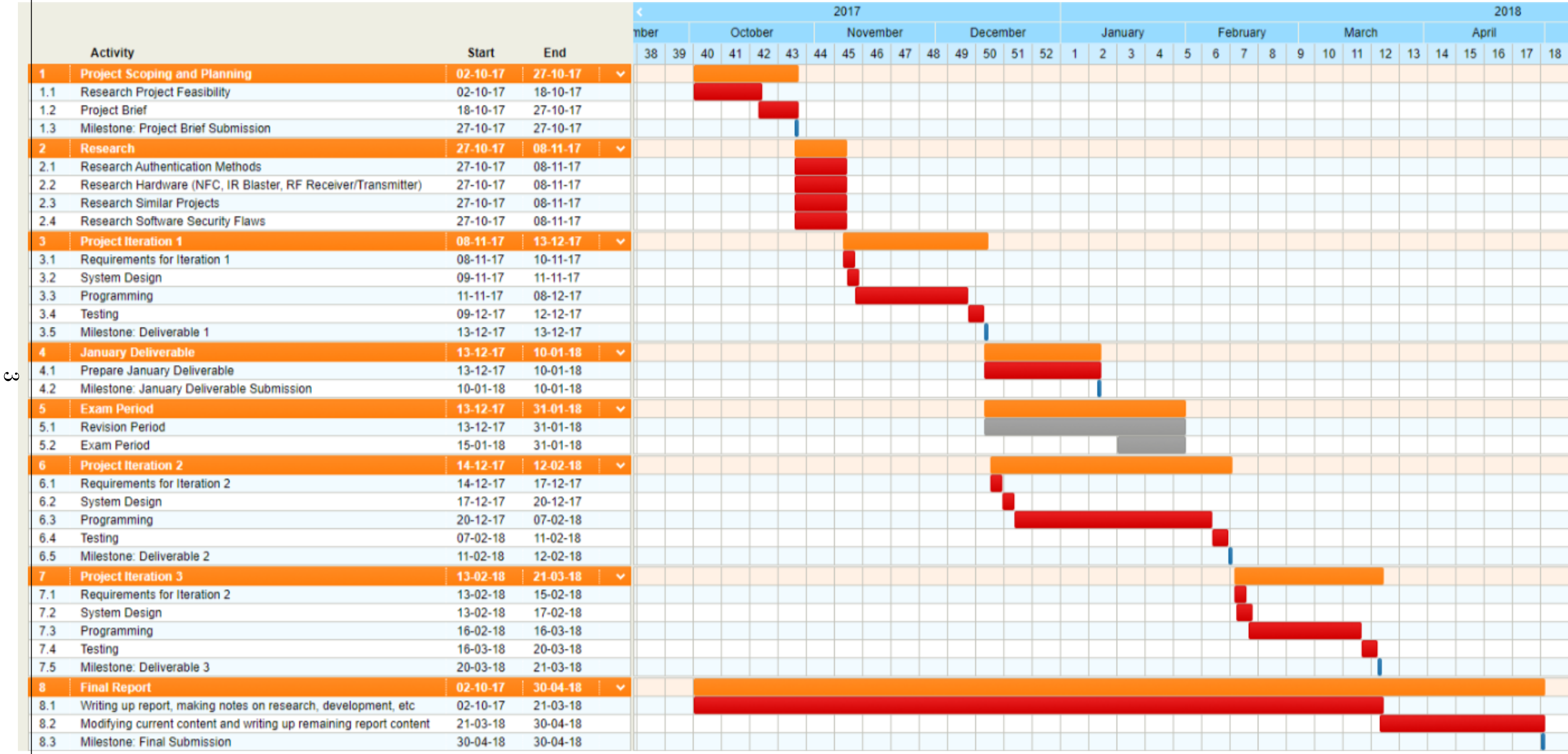
4 Project Objectives

- Design a Database Schema that can be used to store all the users data and settings efficiently
- Develop a core application for the house that will control the main system functions
- Develop a REST API for the core application that other systems (i.e. authentication systems) can talk to
- Implement a minimum of two authentication methods that can be used to unlock the door
- Develop a server system that can be used for the core to sync with and for the user to communicate with the system when outside of the home
- Develop a minimal Android application that the user can use to setup and manage their system
- Ensure that the security of the product is solid and that all communication is secured with TLS 1.2

5 Risk Management

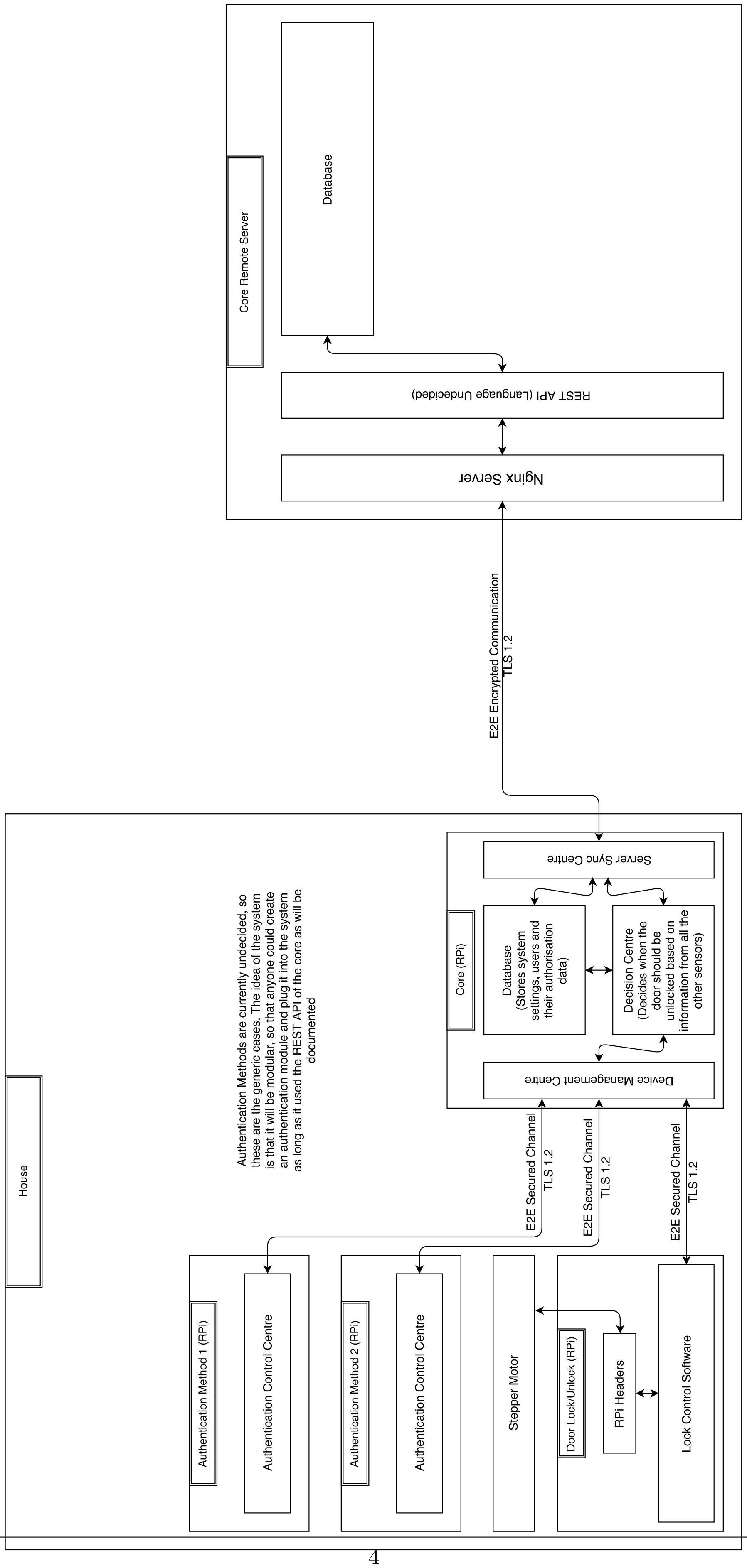
For a project such as this, there is obviously a large amount of risk involved if for some reason the system isn't secured correctly. My plan is to limit these risks as much as possible by ensuring that all communication is using enforced TLS 1.2 as well as not running the system on an actual door until I have a working system that I have run penetration testing on. Replay attacks were something that I had to consider, but these are averted by using TLS 1.2, as it has protection against that built in using the MAC secret and the sequence number. When running this system on a real door, there will be much larger risk involved as it is obviously securing a real house, however as long as all the proper testing is done before it gets installed on a real door, then that should eliminate the risk involved. Obviously, if this gets installed on any door, then it will be in my own house, so I will ensure that the system is functioning properly before installation and any updates are fully tested before being installed on the live system.

6 Project Plan



	Activity	Start	End
1	Project Scoping and Planning	02-10-17	27-10-17
1.1	Research Project Feasibility	02-10-17	18-10-17
1.2	Project Brief	18-10-17	27-10-17
1.3	Milestone: Project Brief Submission	27-10-17	27-10-17
2	Research	27-10-17	08-11-17
2.1	Research Authentication Methods	27-10-17	08-11-17
2.2	Research Hardware (NFC, IR Blaster, RF Receiver/Transmitter)	27-10-17	08-11-17
2.3	Research Similar Projects	27-10-17	08-11-17
2.4	Research Software Security Flaws	27-10-17	08-11-17
3	Project Iteration 1	08-11-17	13-12-17
3.1	Requirements for Iteration 1	08-11-17	10-11-17
3.2	System Design	09-11-17	11-11-17
3.3	Programming	11-11-17	08-12-17
3.4	Testing	09-12-17	12-12-17
3.5	Milestone: Deliverable 1	13-12-17	13-12-17
4	January Deliverable	13-12-17	10-01-18
4.1	Prepare January Deliverable	13-12-17	10-01-18
4.2	Milestone: January Deliverable Submission	10-01-18	10-01-18
5	Exam Period	13-12-17	31-01-18
5.1	Revision Period	13-12-17	31-01-18
5.2	Exam Period	15-01-18	31-01-18
6	Project Iteration 2	14-12-17	12-02-18
6.1	Requirements for Iteration 2	14-12-17	17-12-17
6.2	System Design	17-12-17	20-12-17
6.3	Programming	20-12-17	07-02-18
6.4	Testing	07-02-18	11-02-18
6.5	Milestone: Deliverable 2	11-02-18	12-02-18
7	Project Iteration 3	13-02-18	21-03-18
7.1	Requirements for Iteration 2	13-02-18	15-02-18
7.2	System Design	13-02-18	17-02-18
7.3	Programming	16-02-18	16-03-18
7.4	Testing	16-03-18	20-03-18
7.5	Milestone: Deliverable 3	20-03-18	21-03-18
8	Final Report	02-10-17	30-04-18
8.1	Writing up report, making notes on research, development, etc	02-10-17	21-03-18
8.2	Modifying current content and writing up remaining report content	21-03-18	30-04-18
8.3	Milestone: Final Submission	30-04-18	30-04-18

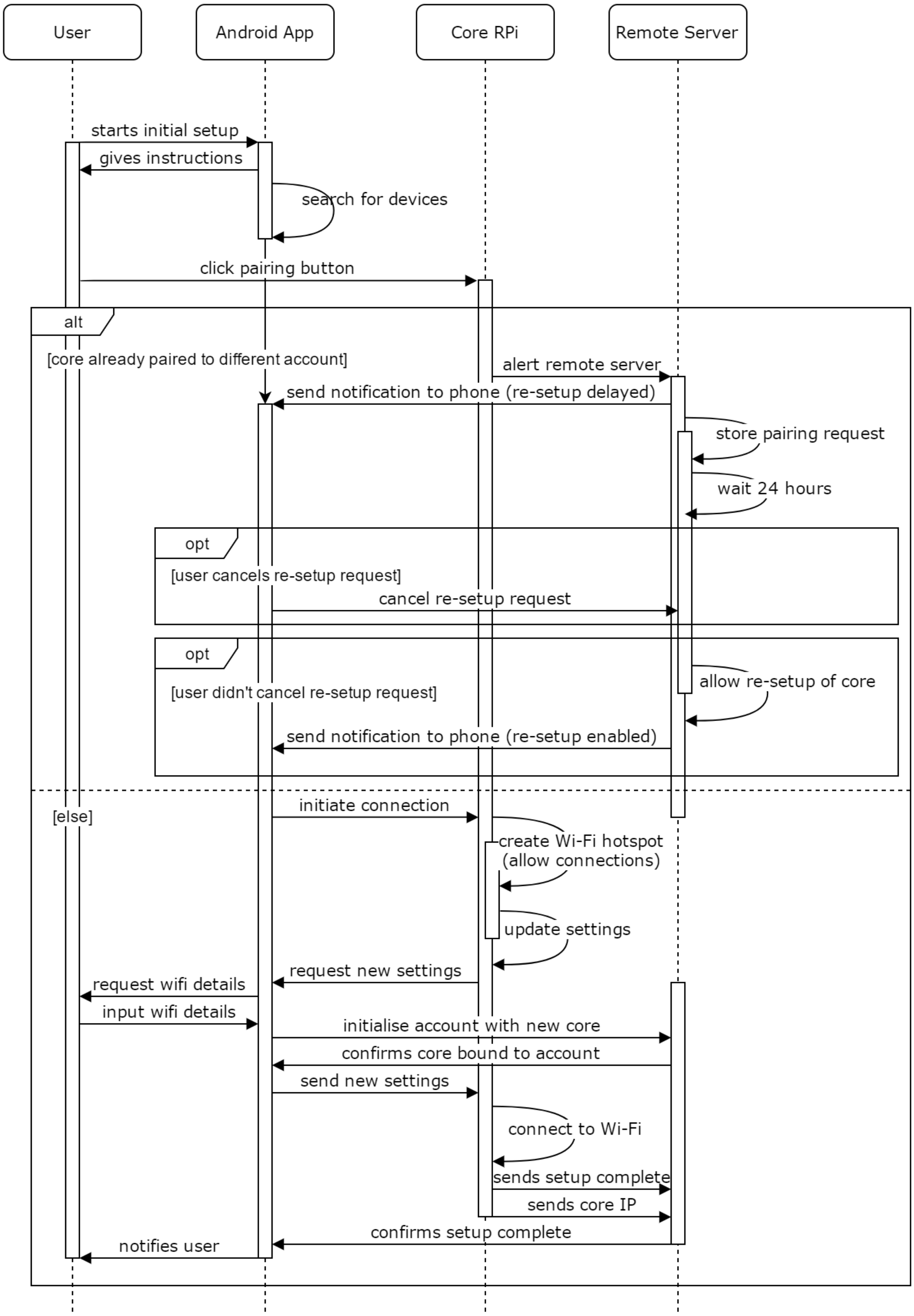
7 Block Diagram


Core Remote Server
Database
REST API (Language Undecided)
Nginx Server
E2E Encrypted Communication
TLS 1.2
House
Core (RPi)
Server Sync Centre
Database
(Stores system settings, users and their authorisation data)
Decision Centre
(Decides when the door should be unlocked based on information from all the other sensors)
Device Management Centre
E2E Secured Channel
TLS 1.2
E2E Secured Channel
TLS 1.2
E2E Secured Channel
TLS 1.2
Authentication Method 1 (RPi)
Authentication Control Centre
Authentication Method 2 (RPi)
Authentication Control Centre
Stepper Motor
Door Lock/Unlock (RPi)
RPi Headers
Lock Control Software

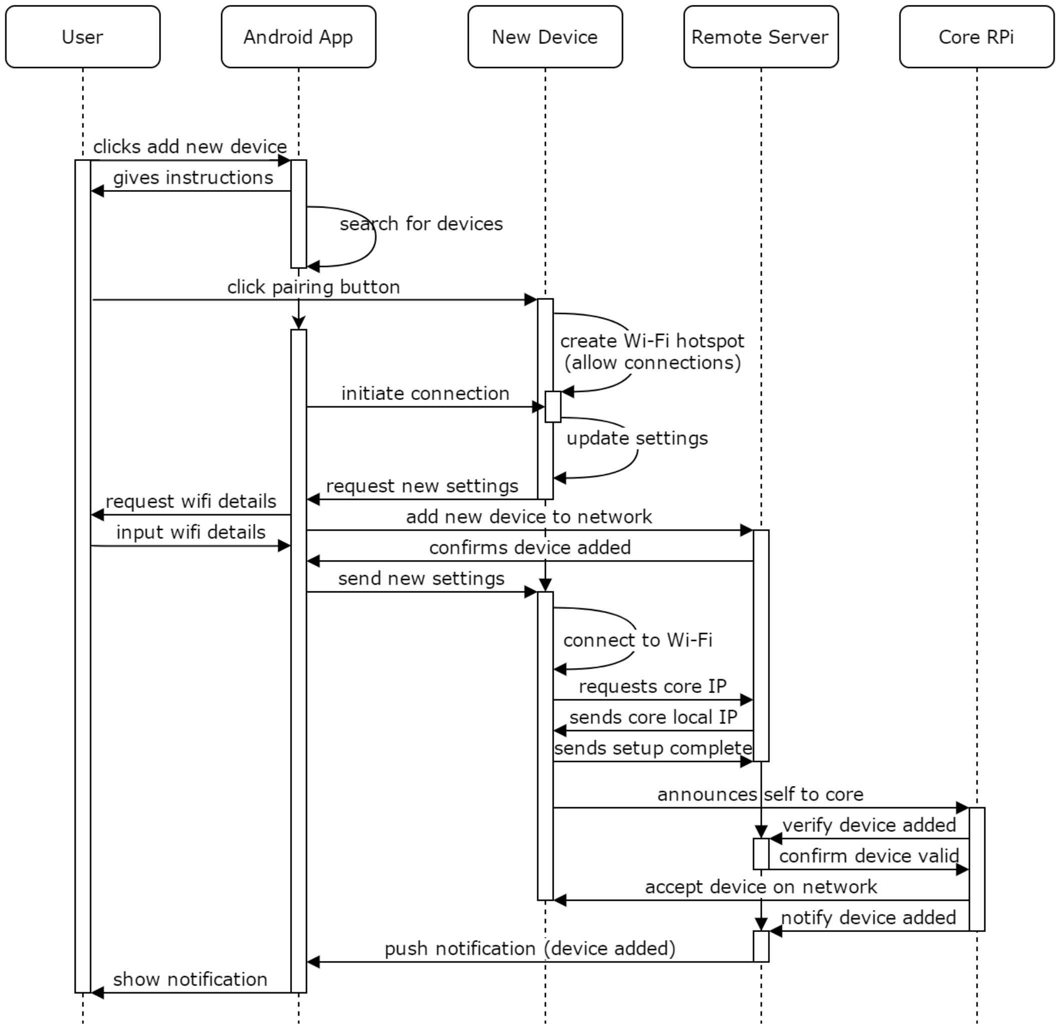
Authentication Methods are currently undecided, so these are the generic cases. The idea of the system is that it will be modular, so that anyone could create an authentication module and plug it into the system as long as it used the REST API of the core as will be documented

8 Sequence Diagrams

Initial Core Raspberry Pi Setup



8.1 New Device Pairing to Existing Network



9 Literature Review

9.1 Aim

Investigate different aspects of a secure smart lock system to see where potential security issues could be and how to avoid or counteract them

9.2 Objectives

- Evaluate different areas of the Secure Smart Lock System
- Investigate methods of ensuring the security of the system
- Investigate ways to avoid potential security risks imposed by necessary parts of the system
- Ensure that security doesn't make the system considerably more difficult for the user to setup, use day-to-day or manage

9.3 Current Smart Locks

Smarts locks and internet of things (IoT) devices, in general, have increased massively in availability and use over the last couple of years and according to Buhov et al. (2016), there will be 50 billion devices connected to the internet by 2020. Having these devices in and around your home constantly connected to the internet introduces some risk if they are not properly secured. This risk is exaggerated if they are devices that secure your home such as a smart lock.

Ho et al. (2016) compare 5 commonly available smart locks that are on the market right now. These locks operated in three different ways: touch-to-unlock, mobile app unlocking, and automatic unlocking. For my purposes, I am mostly interested in the mobile app unlocking as it will be the main way to unlock the Secure Smart Lock System. All the smart locks except for the Lockitron connect to the internet through a Bluetooth connection to the phone, rather than connecting directly, this is interesting as my system will only use Wi-Fi, and will only communicate directly with the phone if they are on the same Wi-Fi network. The study concluded that all the smart locks except for the Lockitron were vulnerable to the attacks that they undertook, this was down to the Lockitron using Wi-Fi directly and therefore being able to verify with the server for the most up-to-date access control lists. They continue to explain that direct connections, while good, do mean that the device must include a Wi-Fi module which increases power usage on the device and so battery-only operation isn't feasible. A good point they make about the Lockitron is that it doesn't maintain its own access control list locally, so in the event of their servers being unavailable, the lock will not function. This is something I plan to avoid in my lock by maintaining a local access control list, however, due to the attacks mentioned in the study, I will ensure that any changes to the access control list are passed onto the lock before the user is told they are saved.

Ye et al. (2017) did a study specifically on the security of the August Smart Lock. This study focussed on more specific attacks such as moving the owner account from one phone to another and taking control of it. All of the attacks that they performed required root access to the Android device in order to perform, which is unreasonable to assume you will be able to get without taking the users device and knowing their unlock code to perform the rooting process. This is further shown by Fuller & Jenkins (2017) who also tried to penetrate the August Lock but without using rooting as their main point of entry. The attacks that Ye et al. (2017) performed certainly aren't ideal, but rooting compromises the integrity of the device and exposes all the data on the device including encryption keys, app data and anything else the device has stored. As shown by Fuller & Jenkins (2017) the August Lock has had a lot of past security issues, however, they have been fixed by the development team since then.

9.4 Certificate Pinning

Certificate pinning is one of many ways to ensure you are talking to the server you think you are. If you end up doing this badly, however, you can decrease the security of your app. A study performed by Buhov et al. (2016) does an evaluation of 25,000 android apps on the play store to see how many of them implement security properly. It turns out that only 21% of applications have no issues, with 36% having a broken implementation of either the Trust Manager or the Hostname Verifier. This issue is brought about by people using self-signed certificates or pinning certificates and trying to, therefore, implement their own version of the Trust Manager and/or Hostname Verifier but doing so incorrectly.

As pointed out by the study, many of the incorrect implementations of the Trust Manager lead to the app accepting any certificate it was given, therefore breaking the chain of trust that is usually in place on the device. Tendulkar & Enck (2014) back this up showing that out of the apps they tested, only 43% use SSL verification correctly with the remaining apps either accept all certificates or accept all hostnames. An even more interesting fact that they discovered was that 53% of the apps they tested implement this custom code to allow all certificates or all hostnames even though they are using valid and signed certificates. The inclusion of this incorrect code allows for man-in-the-middle attacks which would compromise the security of user data being transferred over the network.

Buhov et al. (2016) mentions the different security you get from different levels of certificate pinning. You can pin the end (leaf) certificate which tells you with absolute certainty that you are connecting to the server you think you are. One downside of this is that you must make updates to your app frequently as most certificates only have a lifetime of a year. This problem is made even worse with LetsEncrypt who's certificates expire after a maximum of 3 months. The study explains other solutions for certificate pinning including intermediate and root certificate pinning. Intermediate certificate pinning is less secure than leaf certificate pinning, however, as long as you trust the Certificate Authority (CA) you are using to not sign a certificate for your domain erroneously, then it is just as secure. The major benefit of intermediate certificate pinning is that they update a lot less frequently, the current LetsEncrypt certificate doesn't expire until the year 2021. The last option of root certificate pinning is a lot less secure as you are trusting a lot more parties as potentially multiple CA's will use the same root certificate for signing their SSL certificates.

The studies show me that certificate pinning is overall a good idea, however, it must be implemented correctly. Based off of the tests that were performed by these studies I will ensure that my app only accepts certificates signed by the LetsEncrypt intermediate certificate and that my app ensures the hostname on the certificate matches the hostname on the certificate. If either of these tests fail then it means I have implemented the security incorrectly and as these studies have shown that would make my app vulnerable to MITM attacks which would severely impact the security of my smart lock system.

9.5 Avoiding NFC Relay Attacks

During my research it became apparent to me that NFC has no way of verifying that the device it is communicating with is actually right next to it, making it subject to relay attacks (Francis et al. 2010). This study explains the relay attack as the ability to send the NFC communication over a separate communication channel using other phones as the proxy. This introduces problems if the app is always able to receive NFC communication and unlock the door, as an attacker could then unlock the door by having someone by the door NFC reader and also by a person with permission to unlock the door and just send the data between the two attackers. Oh et al. (2015) suggests distance bounding as a potential solution to this issue as communicating between the proxy devices will add a delay that would normally not be present in the system. The NFC reader should be able to determine the distance between the NFC device and the reader by determining the latency in sending and receiving data. The study points out that this kind of verification can vary massively in latency and therefore you may sometimes end up rejecting legitimate users if their device doesn't respond quickly enough. Two-factor Authentication is another solution that the study suggests and is a much more reliable solution as you can request that as well as the NFC card a user must input a pin, provide a fingerprint or enter a password. Another suggestion they make to directly counter relay attacks is to emit a jamming signal around the reader so that the device can't communicate externally via 3G, 4G, Wi-Fi or Bluetooth. Whilst this is an interesting solution, there are many situations where this would be impractical or illegal to implement a jammer in an NFC enabled device.

Francis et al. (2010) shows how trivial it is to relay NFC data across a Bluetooth connection even on old devices. This study shows that another potential attack mitigation strategy would be to include the location in the NFC transaction as the attacks can only relay the data, not modify it. If this was done the reader could check that the location was within the expected range to be considered near the reader. The issue with this strategy is that the location is not always accurate, getting accurate location through GPS takes time and may not be possible if the reader is undercover.

Using NFC for my project will require investigation into the methods specified above to prevent relay attacks. This has certainly opened up a lot of potential work for me to ensure that the smart lock is easy to use but also secure from relay attacks. I'll be looking into how effective the distance bounding is, but if that proves ineffective or unreliable I will make it so the device has to be unlocked in order to respond to the NFC signal. I will be investigating and comparing these options to the security of a conventional key based lock system.

9.6 TLS 1.2 Security

TLS 1.2 is the most recent version of the transport layer protocol used to secure communications across a network to prevent eavesdropping and tampering. The security of TLS 1.2 is vital for the project as it is what will be used for all communications between the phone, the lock and the server.

Meyer (2014) offers a fairly comprehensive study of SSL/TLS from the very beginning when SSL 1.0 was released. The study points out that SSL/TLS have had many vulnerabilities throughout the time that they have been commonplace in securing internet traffic. It also points out that there are lots of different implementations of the TLS standard, all of which have their own differences and bugs. OpenSSL is the most used implementation on the web at the time of the study based on the usage of web servers and browsers that use the implementations. Part of this study lists a lot of different attacks that have been possible against TLS or some of its implementations over time, all of which have now been resolved in the latest implementations of the TLS standard. The existence of this list does show however that TLS is not perfect, and neither are the implementations. As pointed out by Turner (2014) a lot of the security is left down to the system administrators, whether that is ensuring that only the most secure algorithms are allowed to be used, or patching a security flaw by making a configuration change. TLS can be a very secure protocol if setup in the correct way using up-to-date implementations, but it can also be insecure if badly configured or not updated when exploits are found and patched.

This research has shown me that TLS 1.2 is secure and uses cryptographically secure and proven algorithms for all the encryption it forms, but must be configured to not use the old algorithms and must be kept up-to-date by the system administrators managing the systems using it. I conclude that TLS 1.2 is sufficient for securing the smart lock system from this research, more technical investigation will be done during implementation to ensure that the system is configured as securely as possible.

9.7 Android Fingerprint Keystore Security

This system will utilise the android fingerprint keystore functionality to store and protect the keys used to communicate with and unlock the smart lock. The security of this keystore is fundamental to the security of the app. Does & Maarse (2016) show in their study a few different attacks on the fingerprint authentication methods in Android and how you would go about doing them.

The first attack Does & Maarse (2016) show is trying to get the device to accept a fingerprint which isn't enrolled on the device. Due to the Android system itself not knowing about any of the fingerprints enrolled into the device and therefore having to query the Trusted Execution Environment (TEE) about whether the user provided the necessary authentication, this attack isn't feasible without replacing a core component of the Android system. The attack was possible, but only by replacing fingerprintd to return a value other than 0, which is the only value the system determines as a failure, every time it is invoked. To replace fingerprintd requires root access to the system, and the system will warn you on every boot that the /system directory has been modified and could be corrupt. This means the attack is not feasible as you would need to root the device in order to be able to perform this attack.

The second attack that Does & Maarse (2016) performed on the Android system was trying to replay AuthTokens from the Keystore to the Keymaster which would allow them to perform cryptographic operations authenticated by the replayed AuthToken. To do this they forced the system to give them the same AuthToken every time and then waited for the challenge ID to become the same twice. When the challenge ID was returned as the same twice, they were able to send the same AuthToken and authenticate with the Keymaster to perform cryptographic operations. Two key factors of this attack are that they retrieved the AuthToken from the device memory which would not be possible without root permissions in a normal case. Secondly, they forced the device to always return the same AuthToken, which is not normal behaviour as generally, the device will return a different AuthToken with every request, along with a different challenge ID, therefore avoiding this issue.

In conclusion, the study couldn't find any exploitable flaws with the fingerprint keystore security under normal circumstances, only when they had root access to the device and could, therefore, replace binaries with ones they had constructed themselves. For my use case, I assume that if someone can gain root access to the device, they already had the ability to unlock the device and therefore access the smart lock app to unlock the door.

10 Software Development Project

10.1 Requirements Analysis and Specification

10.2 Design

In this section I will describe certain aspects of the project where I had to make decisions as to how they could be implemented best. In each section I will give a high level description of each method followed by which method I decided to go with and why.

10.2.1 Initiate Connection from Another Device to Core Raspberry Pi

This is a difficult issue to get around as finding a specific device on the network is not trivial, especially if the network is particularly large. There are various different ways I could solve this which are described below.

Method 1 - IP Broadcasting One option is to use IP Broadcasting, where the core uses the networks broadcast range to send data to all the other devices on the network. This would work, but I'm unsure of the reliability of this method and broadcasting the IP of the core to the entire network doesn't seem like something I would want to do. It would likely be fine on a home network, however if it was on a much larger network in a business, this method may not work at all as they could block broadcasting on their switches entirely, or if multiple cores are running on the network it could become quite congested with the number of devices trying to announce themselves to other devices.

Method 2 - Bluetooth Sync Device to Core This would use the Bluetooth on both devices to contact each other and find out the IP of the core on the network. The device would then use this IP to establish a connection to the core. This method has a few issues, namely the security of communicating over Bluetooth as it doesn't have the same kind of security as Wi-Fi. The other issue is that this means the two devices must be within Bluetooth range, which is much shorter than Wi-Fi range and can't be easily extended, unlike Wi-Fi range which has highly available and cheap range extenders.

Method 3 - Contact the remote server to get the Core IP For this to work, the Core would have to announce its IP to the remote server whenever it changed. This would be very simple as the Core will already have a connection established to the remote server. The device that is trying to connect to the Core would then simply have to contact the remote server and request the IP of the Core on the local network, and then establish a connection with it. This is the easiest method to secure as it gives one point of contact to retrieve this information, and eliminates the need for any kind of discovery. This also allows devices to be as far away from each other as required if they are on the same network.

11 Bibliography

References

Buhov, D., Huber, M., Merzdovnik, G. & Weippl, E. (2016), Pin it! Improving Android network security at runtime, *in* '2016 IFIP Networking Conference (IFIP Networking) and Workshops, IFIP Networking 2016', pp. 297–305.

Does, T. & Maarse, M. (2016), 'Subverting Android 6.0 fingerprint authentication'.

Francis, L., Hancke, G., Mayes, K. & Markantonakis, K. (2010), Practical NFC peer-to-peer relay attack using mobile phones, *in* 'Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)', Vol. 6370 LNCS, pp. 35–49.

Fuller, M. & Jenkins, M. (2017), 'Security Analysis of the August Smart Lock', p. 16.
URL: *https://courses.csail.mit.edu/6.857/2017/project/3.pdf*

Ho, G., Leung, D., Mishra, P., Hosseini, A., Song, D. & Wagner, D. (2016), 'Smart locks: Lessons for securing commodity internet of things devices', *Proceedings of the 11th* pp. 461–472.
URL: *http://dl.acm.org/citation.cfm?id=2897886*

Lackner, G. (2013), 'A Comparison of Security in Wireless Network Standards with a Focus on Bluetooth, WiFi and WiMAX', *International Journal of Network Security* **15**(6), 420–436.

Meyer, C. (2014), '20 Years of SSL/TLS Research An Analysis of the Internet's Security Foundation'.
URL: *www.nds.rub.de http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/MeyerChristopher/diss.pdf*

Oh, S., Doo, T., Ko, T., Kwak, J. & Hong, M. (2015), Countermeasure of NFC relay attack with jamming, *in* '2015 12th International Conference and Expo on Emerging Technologies for a Smarter World, CEWIT 2015'.

Tendulkar, V. & Enck, W. (2014), 'An Application Package Configuration Approach to Mitigating Android SSL Vulnerabilities', *MoST* .
URL: *http://arxiv.org/abs/1410.7745*

Turner, S. (2014), 'Transport layer security', *IEEE Internet Computing* **18**(6), 60–63.

Ye, M., Jiang, N., Yang, H. & Yan, Q. (2017), 'Security Analysis of Internet-of-Things : A Case Study of August Smart Lock'.