



UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Researching BLE (Bluetooth Low Energy) and NFC (Near Field Communication) for use as a
personal identification service to unlock a vehicle.

Author

Jake Craven

Student ID

15160386

Supervisor

Dermot Shinnars-Kennedy

B.Sc. Computer Games Development

Department of Computer Science and Information Systems

Acknowledgements

Firstly I would like to thank my supervisor Dermot Shinnars-Kennedy who was a tremendous help in guiding this project and keeping me on track. Without Dermot advising me to constantly take notes of my ideas this project would not be near as robust as it is. He was also a great help in guiding me through completing this report which is the largest I have written to date and mostly uncharted territory for me.

Next, I would like to thank my family without whom I wouldn't even have made it to college never mind fourth year and their support both financially and emotionally has been greatly appreciated and the source of any and all of my successes.

I would also like to thank the many great friends I made through my time here in UL for the support and encouragement that helped me getting through the course.

Declarations

The work described in this document is, except where otherwise stated, entirely that of the author and has not been submitted in any part for a degree at this or any other University.

Signed: _____

Dated: _____

Acknowledgements	2
Declarations	3
1. Project Title	6
2. Project Summary	6
3. Context	7
4. Motivation	7
5. Research Question/Objective	9
6. Research: State of the Art	9
7. Pre-Development Work	12
7.1. Research	12
7.1.1. Overview	12
7.1.1.1. NFC	12
7.1.1.2. BLE	15
7.1.2. Security	18
7.1.3. Conclusion	21
7.2. Project	23
7.2.1. Vehicle Side	23
7.2.2. Mobile Side	24
8. Pre-Development Plan	25
8.1. Project Requirements	25
8.2. Schedule	26
8.3. System architecture	28
8.4. UI Designs	31
9. Project Development	33
9.0.1. Initial Communication set-up	33
9.1. NFC Implementation	35
9.1.1. Android HCE	36
9.1.1.1. Building the NFC Protocol Stack	38
9.1.1.1.1. ISO 14443-1 and ISO 14443-2	39
9.1.1.1.2. ISO 14443-3 (Anti-Collision and Card selection)	40
9.1.1.1.3. ISO 14443-4 (Transmission Protocol)	40

9.1.1.1.4. ISO 7816-4	40
9.1.1.1.5. Disconnection	41
9.2. BLE	41
9.2.1. How BLE communication works	42
9.2.2. Why BLE is too limited for this smart-lock?	42
9.3. Security	43
9.3.1. Encryption	43
9.3.2. Key Data	43
9.3.2.1. Rolling key	44
9.3.2.1.1. Rolling Key Tests	46
9.3.2.1.1.1. Check Shared locations	47
9.3.2.1.1.2. Shared Partners Test	47
9.3.2.2. Specific Element Request	48
9.4. Data Storage	49
9.4.1. Vehicle Side	49
9.4.2. Android Database	50
9.5. Temporary Keys	51
9.6. UI	52
9.6.1. Pi UI	52
9.6.2. Android UI	53
10. Retrospective	54
10.1. Overview	54
10.1.1. Implementation Issues	54
11. Conclusions	55
11.1. Delivered product	55
11.2. Which is better, NFC or BLE?	56
12. References	58

1. Project Title

Researching BLE (Bluetooth Low Energy) and NFC (Near Field Communication) for use as a personal identification service to unlock a vehicle.

2. Project Summary

The goal of my project is to examine which technology is better suited for identifying a person when paired with a smartphone. To do this, we will more be examining the area of car keys to see if we can use our smartphones reliably as a car key and to potentially allow for other features and options. The goal is to create an application that can securely communicate via Bluetooth low energy or NFC to demonstrate that we can use modern phones to supplement the use of a traditional car key and so that we can enable other abilities, one of these features which I am trying to implement is the ability to remotely give temporary access to your vehicle for a set amount of time. Other features would be to determine what doors to unlock and like in more modern and premium vehicles to specify to auto open specific doors. I hope to determine the strengths and weaknesses of both NFC and BLE by investigating various manners from capabilities, limitations, security and ease of implementation and possibly come to a conclusion on which technology is better suited to this function.

3. Context

Mobile phones are becoming an increasing part of our lives and many consider them an extension of ourselves. Many people are becoming more reliant upon them, even now Google/Apple pay are becoming increasingly popular and they cut back on the need to leave the house with a wallet. Recent years have shown us how phones can be treated as personal identifiers to allow us to make our daily lives much easier. Two technologies that will best facilitate this use of phones are NFC (Near Field Communication) and BLE (Bluetooth Low Energy). These technologies, while quite different will allow for a similar use case, the ability to detect devices in a nearby location and to send and receive information to identify a user and unlock their vehicle if it is authorised to do so.

4. Motivation

This project is partly inspired by a document released by the car connectivity consortium, a joint venture of mobile and vehicle OEMs, which hopes to standardise the way in which smartphone technology interacts with vehicles. These aren't laws but rather a set of guidelines and the document details a list of requirements for successful implementation. This document suggests potential technologies which could be used to develop this digital key one of which is NFC but given how both NFC and BLE are used in the highly security dependant mobile payments I decided to investigate if both would be suitable potential candidates for this feature. Before

delving too deep into this project I was aware of some of the requirements that aren't listed in the Digital key specification that I would have considered (e.g. prevalence within smartphones, cost and power draw). This further cemented my decision to focus on NFC and BLE as both require very low power and are relatively inexpensive and are used in most 2016 and up flagship smartphones.

([4] *Building Digital Key Solution for Automotive Content* n.d.)

I also worked with Jaguar Land Rover as part of my CoOp and gained both a great appreciation and understanding of the automotive industry which helped fuel my interest in doing a project such as this.

Going further beyond the automotive industry if this technology can work successfully in cars then it could also see use in stationary locks. For example, for your home or for a parcel motel or equivalent service, where the key can be digitally transferred to someone and potentially be removed upon request of the owner. There are some examples of this already in home locking systems. These locks, however, aren't held to the same standards of reliability as it would in a vehicle because were you to find an issue with the lock within your home you can take out a screwdriver and replace it easily. In a vehicle, the process isn't as simple and would likely require the manufacturer to make the change for you.

5. Research Question/Objective

The goal of my research is to see whether or not NFC and/or BLE are suitable technologies for implementing a digital key on a smartphone.

6. Research: State of the Art

The idea of a wireless car key is far from new at this point with RKSs (Remote Keyless Systems) being used for years. These systems are small radio transmitters that transmit a signal within the range of 20m to tell a car to lock or unlock and operates at a radio frequency of between 300 MHz - 400 MHz. In recent months different concept key ideas have appeared, that provides the key with more options, power and style. These concept keys generally have some sort of complex coloured dynamic screen controllable through some form of capacitive touch, missing mechanical buttons of the more traditional vehicle key fob. ([14.] howstuffworks.com 2001)



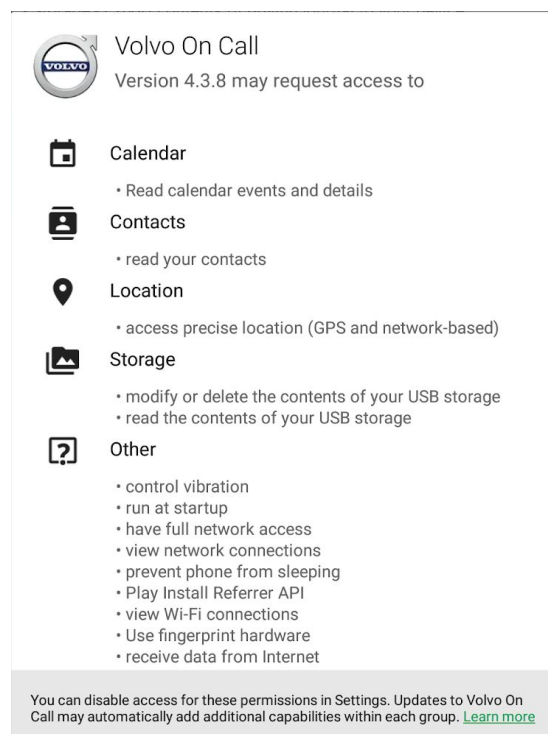
(DunderGifflin and Imgur 2018)



(Mohd 2019)

These concept keys are made by automotive enthusiasts and nothing of the sort has ever appeared in a commercial released or announced vehicle. While these users might want keys with more functionality the idea of introducing a new device with a screen and a battery for people to have to charge at regular intervals seems like too much of a hindrance for practical use.

This is not the first attempt at using a smartphone to unlock vehicles. Volvo and Tesla already have versions of this idea implemented. Volvo back in 2015 announced their ‘Volvo on Call’ application which when announced was said to work via Bluetooth. However after installing the application and checking it’s permissions I have seen that it doesn’t make any requests to use Bluetooth.



The full list of permissions used by ‘Volvo on Call’ Android

I haven't been able to find any definitive mention of what method the Volvo application uses to control vehicle locking but considering the other functionality in the app, it most likely uses the internet by using the LTE chip in many modern vehicles. If this is true then there are times when the application will not function. This could happen when you are out of mobile data and can't access the internet or when you are in a secluded area and both the vehicle and your phone are unable to get online. This, in my opinion, would render the locking function too unreliable for me to comfortably leave my vehicle and rely on my phone. If I were forced to bring my keys with me then the purpose of an application to lock and unlock a car would be pointless. Persistent internet access with the ability to lock and unlock a vehicle does create an additional security risk as with all wireless signals but the wider range of the whole internet does add more potential for vulnerabilities than a shorter range alternative. However, the internet connection isn't all bad as it does also allow for GPS tracking of your vehicle within the same application if it were ever stolen or even more simply if you lost it. Tesla's solution does, in fact, use Bluetooth and they also have an RFID key card that gives access to the vehicle. It also offers the same abilities of the Volvo application with additional features such as checking current charge and pre-heating/cooling the vehicle.

([6] D'Angelo 2017)

7. Pre-Development Work

7.1. Research

7.1.1. Overview

A general look at both technologies viewing how they work, their capabilities and their origin. The focus is on parts that are relevant to this project. For my research in this section, I'll be discussing different areas which I have researched and dividing that research by NFC and BLE.

7.1.1.1. NFC

NFC (Near Field Communication) is a subset of High-Frequency RFID radio waves. This means that it operates at a frequency of 13.56 Mhz. It was first discussed in 2004 when the NFC forum, originally comprised of Sony, Nokia and Philips, hoped to make use of the range limitations of High-Frequency RFID to ensure security and ease-of-use. They first proved this idea with the introduction of the earliest specification for NFC tags in 2006. Although NFC has only been around for a relatively short amount of time it has exploded in terms of adoption and is now becoming commonplace within the smartphone market as more and more OEMs have begun including it in not just their high-end handsets but also their mid to low-end hardware too. Its adoption has been boosted by Androids early adoption of the technology back in 2010 with the launch of Android 2.3 and by continuing support in all iterations to date with the launch of

Android 9.0 last year, (2018). ([12.] “History of Near Field Communication - NearFieldCommunication.org” 2010),

NFC operates at low speeds of between 106 kb/s - 424 kb/s within an incredibly small range of under 20 cm but generally about 4cm. This range can be looked upon as a negative but many see it as a benefit because it limits the number of devices that could be eavesdropping so it can be seen as a security benefit. Part of the reason for the major variance in the range comes from how it's powered. NFC has two different modes of power; passive and active. Passive nodes don't have a power source themselves and instead use the power of an active node to operate. An active NFC node will emit a small electromagnetic field to power a passive node, a feat only possible because of its low power draw. In this model, a passive node would be a transponder with an active node being an initiator. This is the basis for all of the NFC communication, even active-active. The initiator will send a request and the receiving device, transponder, will send a reply. However, an NFC tag can at the same time be an initiator and transponder as it has the capability to read and write simultaneously. There is an additional difference between active and passive and that is in the unique case where there is active-active communication operating at a Baud rate of 106 kBaud. In those circumstances it uses a modified Miller coding with 100% modulation otherwise it uses Manchester coding with 10% modulation. This essentially means that when there is no information being exchanged in active-active there is zero output from both devices as opposed to with Miller 10% modulation where there is a small amount of information output.

([20.] Mostafa and Allah 2011, [25.] Rahul *et al.* 2015)

One of the common uses of NFC is in marketing strategies where an NFC tag is included in a poster that allows people can tap their smartphone and get more information on the product. This is generally not secured as the information isn't sensitive. It is also being used in wireless payments both being used as the wireless tap function in credit/debit cards and also in mobile payments by being the primary technology used in Google pay and Apple pay. It is also commonplace in office environments as key cards. The benefit of this is that it allows employers to track which employees come and go from the office and it also in cases of vulnerabilities allows for the cards to be black and white listed as necessary. ([27.] Thrasher 2018)

7.1.1.2. BLE

BLE (Bluetooth Low Energy) as the name implies is a lower powered version of Bluetooth. Due to branding confusion, it has also been called Bluetooth smart through its lifetime. It was developed to compete with emerging IoT technologies that were trying to create low energy communication systems with the goal of being able to run for years without maintenance. In 2009 the Bluetooth special interest group (SIG) announced that BLE would be part of the Bluetooth 4.0 spec. One of the first commercial products to include BLE was the iPhone 4s and because of this it has gained a lot of traction very quickly and is part of the majority of mobile devices at every price range. Although BLE and Bluetooth are part of the same spec and standards they are currently incompatible as they both use slightly different controllers. Manufacturers can decide to implement only classic Bluetooth or only BLE, called single-mode. However, it is more common to include both, called dual-mode.

([11.] Gomez *et al.* 2012)

BLE has a range of up to 100 metres and can provide distance by performing a simple calculation upon the RSSI(Received Signal Strength Indicator) and the manufacturer determined MP(Measured Power), the expected RSSI at 1 metre. The calculation is

$$\text{Distance} = 10 ^ {((\text{Measured Power} - \text{RSSI})/(10 * N))}$$

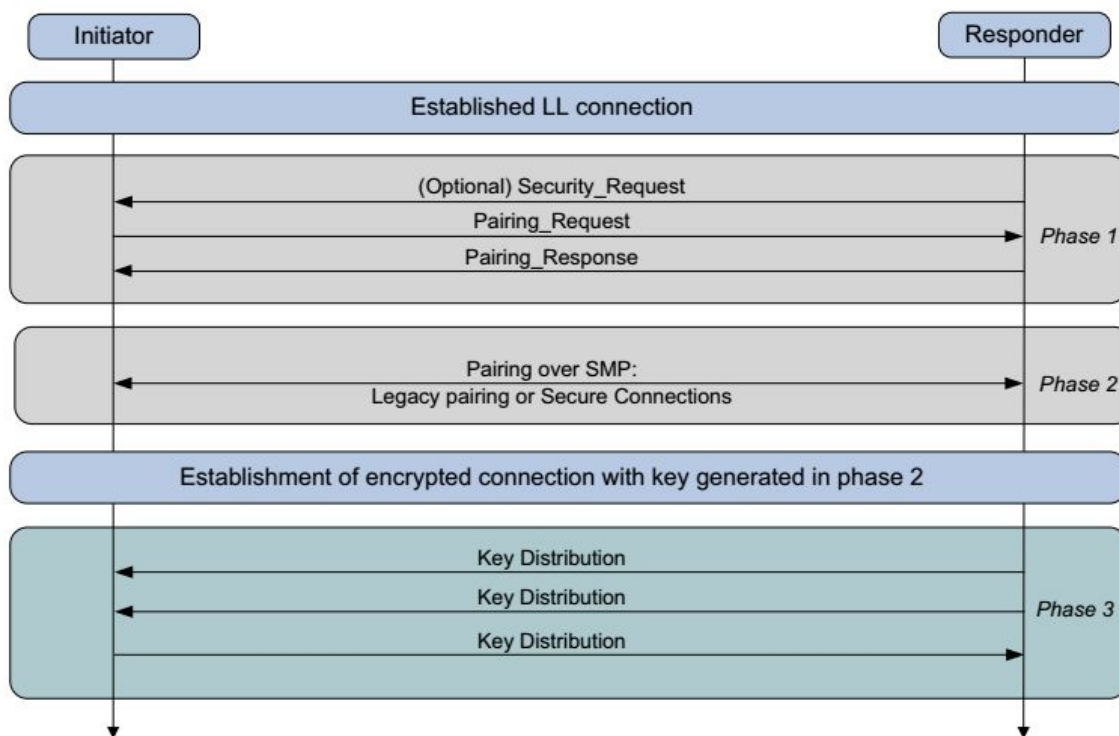
Where ‘N’ is a constant environmental variable meant to deal with potential barriers in a static location. Because it calculates this distance based on signal strength it can introduce an issue as a

single BLE device would be unable to determine when the range is causing a poor signal or an interruption/blocking force in the way that the 'N' variable would be unable to approximate for. There are methods of obtaining more accurate location data by daisy-chaining devices together and getting the range from each device to figure out a more precise location. Practical execution of this idea can be seen in Apple's iBeacon technology. ([15.] iotbymukund 2016)

BLE has a data transfer rate of anywhere between 125 kb/s - 2mb/s with a latency of around 6ms. It also has a power draw of 0.01W - 0.50W depending on usage. This means that a worst case scenario of BLE operating at full power is that it would last for about 12 hours on 2 standard 6Wh AA batteries.

BLE has two modes of communication within its link layer, advertiser and data channels. Advertiser channels work by broadcasting the signal regularly and using a scanner to listen for a response. The rate at which the advertiser sends out data is called the advertiser interval. To avoid colliding with an alternate source a random time of 0ms - 10ms is added to the advertiser interval. When a target device responds to the advertisers signal then the scanner on the advertiser's side intercepts it and the devices will usually pair. The device then switches to the data channel for bidirectional communication. BLE pairing occurs in three phases. Phase 1 is where the request to pair is initiated and the devices can exchange whether or not they want to proceed to bond, which will be discussed later. Phase 2 is where the two devices exchange TK (Temporary key) information or if bonding flag was sent in phase 1, an LTK(Long term key) will be sent if one has been previously set between the two devices. If a bonding type has been

chosen then phase 3 begins. In phase 3 an LTK is set. Once an LTK is set then the devices can pair more easily in future. This is a much more secure way for BLE devices to communicate although it does initially take longer for the first pairing. Paired devices are also more reliable as they avoid interference by using a method called frequency hopping to avoid other devices communicating on the same frequency. Even though BLE operates on the 2.4 GHz band it still has a small amount of variance up to 2.48 GHz where it can change frequency to as part of frequency hopping. ([11.] Gomez *et al.* 2012)



Phases of BLE pairing. ([3.] “Bluetooth Pairing Part 1 Pairing Feature Exchange | Bluetooth Technology Website” 2016)

7.1.2. Security

Security is of the critical importance even in the most trivial of wireless communication but when the wireless communication is responsible for protecting your vehicle and all items in it, then security is taken to the forefront of implementation. Typically WKS, wireless key system used in vehicles today, uses encryption paired with rolling code. The Encryption is used to try and hide the data being sent and rolling code so that even if someone is eavesdropping on the data being sent they would be unable to use it as the expected value will change upon use. However, even this has been proven to not be. At Defcon 2015, an annual hacking convention, a hacker named Samy Kamkar who detailed his method of intercepting the signal from a WKS. His way of doing this was by both intercepting the signal from the key and blocking it from the car. This prevented the rolling code system from kicking in and once he had the copy he had one-time use access to the vehicle. ([17.] Kamkar 2015)

As mentioned earlier the short range of NFC is often seen as a security benefit. However, it is still open to a number of attacks. BLE whom despite its ability to pair is also vulnerable to attacks. Both happen to share many common threat types but the methods of execution of these threats differ. For the rest of this section, We will consider the different threat types and the methods of prevention used to combat each threat. ([5.] Coskun *et al.* 2013, [29.] Zegeye *et al.* n.d.)

Eavesdropping/Cloning/Impersonation

The general form of attack here is a device listening in on the communication and then using this information maliciously. The information, in this case, could, in theory, be used to gain unauthorized access to the vehicle.

Solution: Rolling code, Secure Channel Establishment, Specific key agreement protocols

MitM (Man in the Middle)

This is where a third party device mimics either the second or first party device to trick the other one to release information to it. This can lead to information being sent to the wrong device whose user can apply it maliciously. It could also alter the state of the device by instructing it to act differently which could corrupt future communication.

Solution: Secure connection, Encryption

Phishing/Social Engineering

People trying to convince device users to give away sensitive information which may compromise the security of the said device.

Solution: Intuitive UI that limits users ability to accidentally reveal or share sensitive information. In addition, the system should prevent sensitive information from being something that may come up in another context and prevent users from accessing such information. Add option for additional security such as fingerprint scanning and perhaps a pin.

Malware

Malicious software that can harm a device, causing it to act unpredictably and perhaps expose sensitive information.

Solution: To hold sensitive information in a SE (Secure Environment) and limit the functionality of each system so, for example, the door in the car can't tell the accelerator to go and same for the phone side.

DOS(Denial of Service)

When a connection or location gets flooded with interfering signals so that required information can't be sent between devices either at all or uncorrupted.

Solution: Establish a secure connection.

Data Injection

Where unexpected data is received and interpreted as code which causes system errors

Solution: Encryption, Establish a secure connection. Verify all data received before acting upon it.

7.1.3. Conclusion

	NFC	BLE
Speed	< 0.426 mb/s	< 2 mb/s
Range	< 0.20 m	< 100m
Cost	~ €0.10	~ €5.00
Power Draw	0W	< 0.5W

Looking at the above table it appears it is split evenly when it comes to categories won, NFC winning out in power draw and cost and BLE having advantages in speed and range, it fails to show the whole picture. BLE seems to have an advantage in terms of capability as it has a greater data transfer speed and much greater potential range, however, this doesn't account for every element as there are other factors that can't be summarised as easily.

Ease of use is a major factor to consider as the application of this project would need to be convenient on top of everything else to be worthwhile and garner adoption. BLE would require a more complicated development and use as the pairing will require explicit user input to be reliable and secure. This additional requirement, while infrequent, will make it an extra step that a traditional key or WKS doesn't face. Though NFC also has a drawback in this sense as the

limited range means there is the potential that a user would need to tap their phone against a vehicle for it to work, given its limited range. This may be unintuitive and if this is the case it wouldn't be much easier than using a key. However to resolve this, one could use an NFC enabled smartwatch to detect when their hand was in the range of the door though this would require the user to have additional hardware.

As mentioned earlier the potential for signal interference is much higher for Bluetooth and could reach a point of being inoperable should you find yourself in a busy car park for example where there would be lots of cars all searching via BLE for a phone to unlock them. This may interfere with the level of success. This isn't really an issue for NFC due to its very small range.

Thus far in my research, I haven't found one to be definitively better than the other. The added versatility of BLE, which could enable some additional future features, does make it seem like it could be a better choice going forward. But from what I've found in my research there is still the potential that manufacturers may prefer to use NFC simply for its cost and power saving potential. I have also found that the two technologies can be used in tandem to provide an overall better experience. An idea for using both in the context of this project could be to use NFC on the inside of the vehicle to aid in the setup of BLE by communicating the information required to automatically pair the phone to the vehicle.

7.2. Project

I am developing this system in two parts, the vehicle side and the mobile side. The system will be a simulation of how it could potentially work so I won't be implementing it in an actual car.

7.2.1. Vehicle Side

The vehicle side of the project will be demonstrated using a Raspberry Pi 3. My reasons for choosing this is because I have no prior experience with hardware and the Pi family is well known for its ease of use and also has a wealth of online forums and tutorials to help me. It also runs in a Linux like an environment and can run C and python both of which are dominant in the automotive industry so a solution I develop within those languages could, in theory, be used in a production vehicle. The Pi 3 also has a built-in Bluetooth chip with BLE capabilities and I will also be using the RFID RC522 reader which I can use for the NFC portion of the project. In an effort to try and further emulate a vehicle I will be developing an additional controller system on the Pi to act as the BCM (Body Control Module), a central hub computer within the vehicle. This will be a simple program that will trigger the rest of the systems to turn on in the correct order but also track the data that the 'lock' mechanism receives and provide some validation. It will also be able to alter the state of the 'lock' mechanism to demonstrate different possibilities and also for testing purposes. For ease of development, I'll be developing the 'BCM' with the aid of an SQL database to store different information and also to log what interactions have occurred.

7.2.2. Mobile Side

I am developing the mobile side of the project for an Android 8.0 device on Android studios (Java). I will also be using the internal SQLite database that is provided natively in Android applications for storing the applications persistent information which I will use as my SE (Secure Environment) for information such as key data. I am using these software tools simply because they are what I am most experienced with and they have the best support online in terms of Android development from both Google and other users. I am doing this project on Android simply because it is the hardware I already own and I don't have access to an Apple computer which would be required to develop for iOS. I'll be using Android Studio 3.1 because it is a recent stable build. There is the potential that a future IDE update could alter how you should develop and break my version of the project so I won't be updating the IDE throughout development. My test devices will be the Xiaomi Mi Mix 2 and a Xiaomi Redmi 3 the latter of which unfortunately doesn't have NFC capabilities as it is a slightly older and cheaper phone.

8. Pre-Development Plan

8.1. Project Requirements

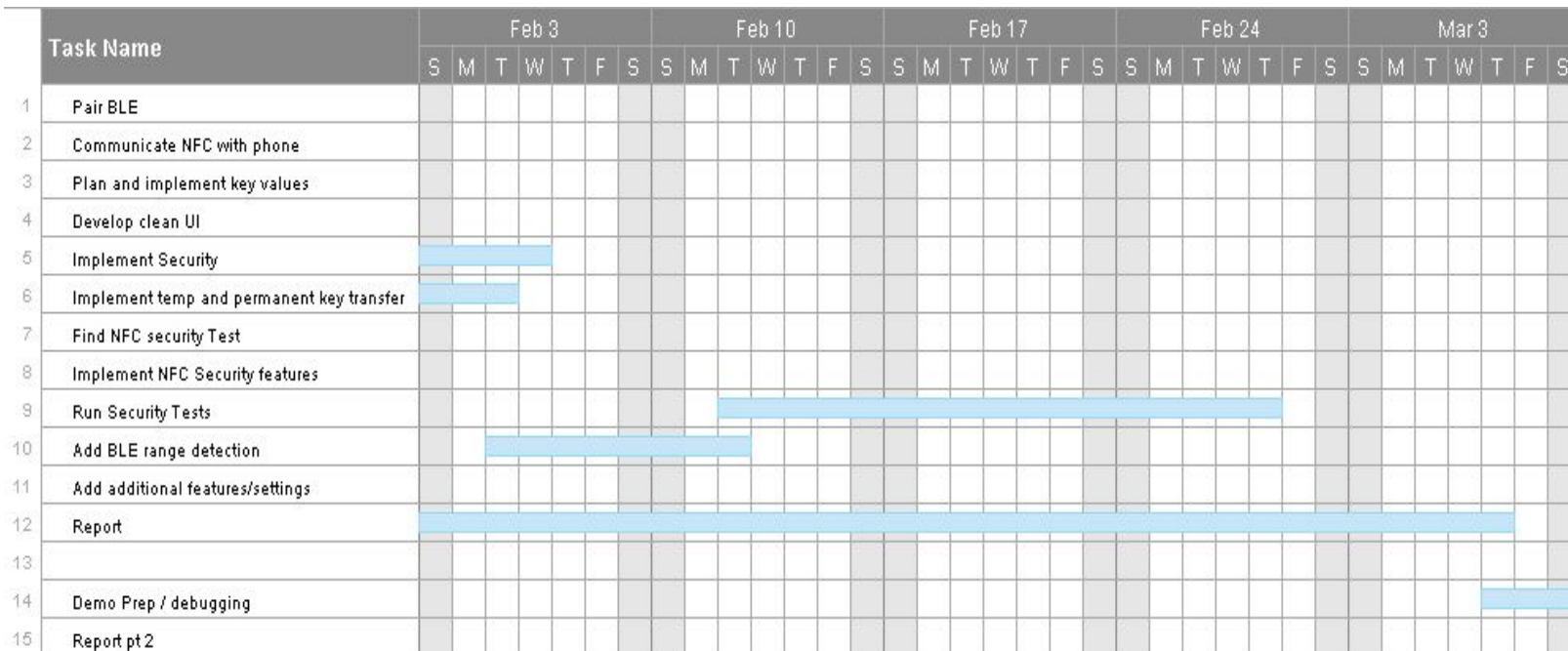
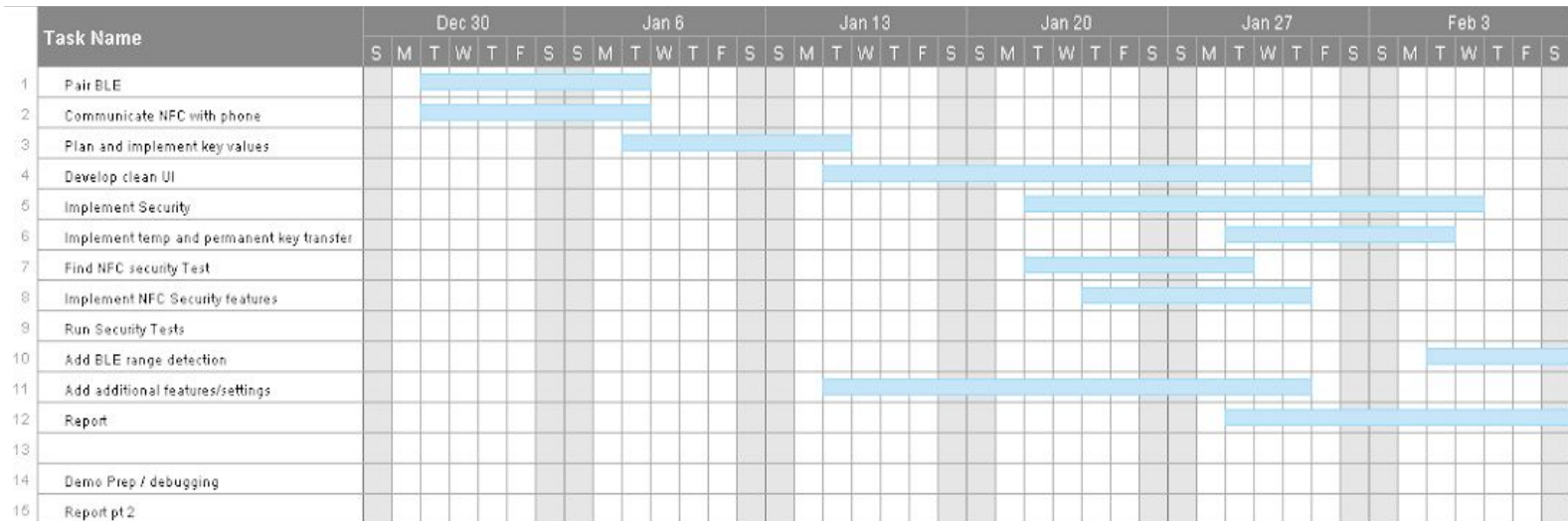
In line with the CCC's (Car Connectivity Consortium) digital key specification, I plan on implementing most of the requirements they layout such as, temporary key share, permanent key transfer or key disabling and security.

There are however a few additional requirements I will be attempting to implement. One such addition is adding a set timer to disable the temporary key. This would allow you to share your vehicle and have more peace of mind that you'd get it back. I will also try to implement optional 2-factor authentication to allow people the option of either an additional pin or fingerprint before unlocking the vehicle. Finally, I am focusing my efforts on an implementation that doesn't require internet access as I want a system that is as reliable as possible and won't be dependant on LTE signal strength/availability.

The security requirements of a project like this are too broad for me to comprehensively tackle so I will be focusing on tackling MitM attacks as I feel if I try to deal with this issue my solution would also address many similar issues such as eavesdropping. I hope to develop a solution that will cover most issues when paired with encryption.

8.2. Schedule

Going forward I'll be shifting my focus much more heavily towards development. The following is a Gantt chart of my schedule in 2019.

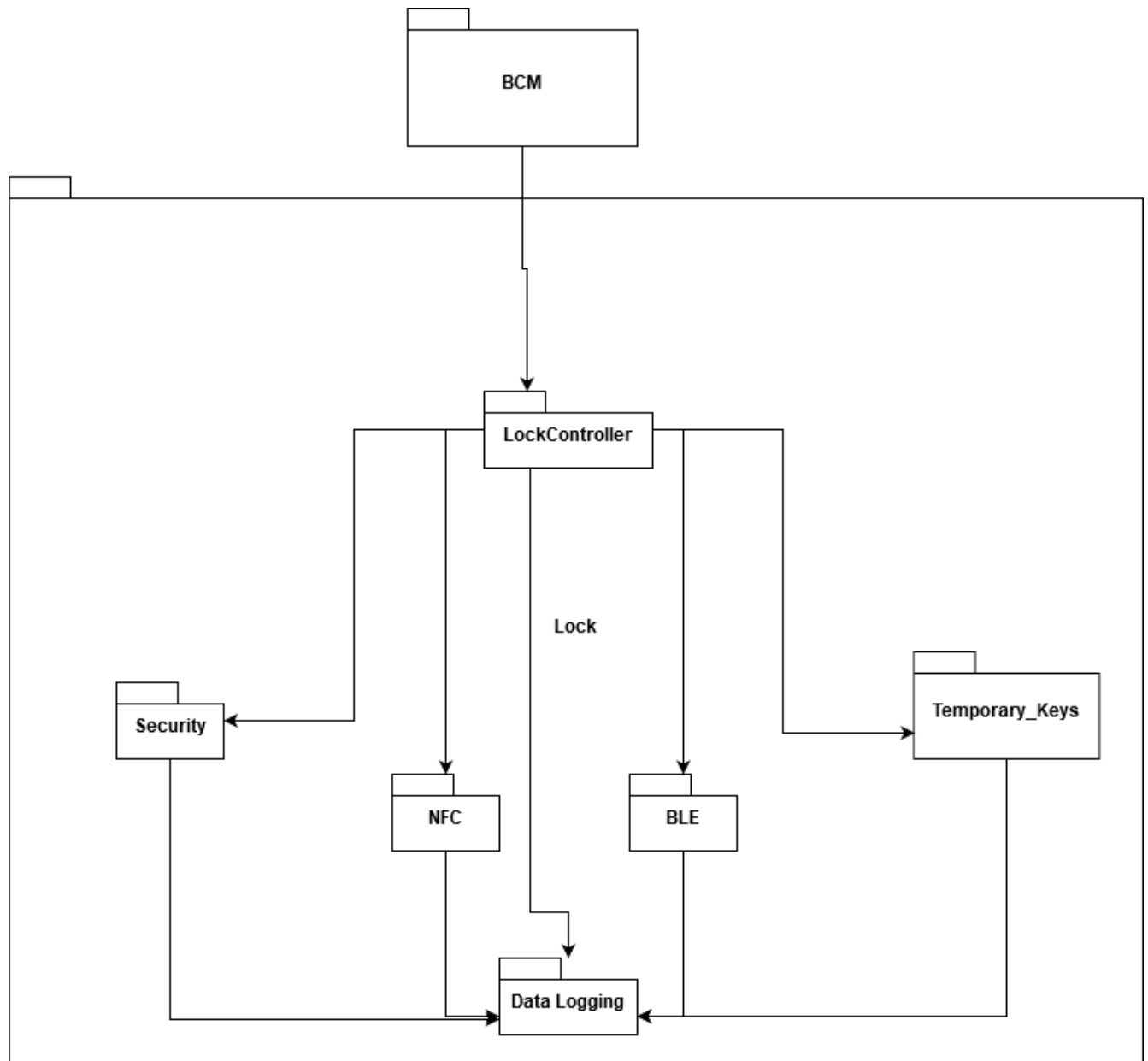


	Task Name	Mar 3							Mar 10							Mar 17							Mar 24							Mar 31							Apr 7								
		S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S		
1	Pair BLE																																												
2	Communicate NFC with phone																																												
3	Plan and implement key values																																												
4	Develop clean UI																																												
5	Implement Security																																												
6	Implement temp and permanent key transfer																																												
7	Find NFC security Test																																												
8	Implement NFC Security features																																												
9	Run Security Tests																																												
10	Add BLE range detection																																												
11	Add additional features/settings																																												
12	Report																																												
13																																													
14	Demo Prep / debugging																																												
15	Report pt 2																																												

Clearer Images can be accessed [here](#).

8.3. System architecture

I have also done some very high-level diagrams of the architecture that I hope to implement in the coming weeks.



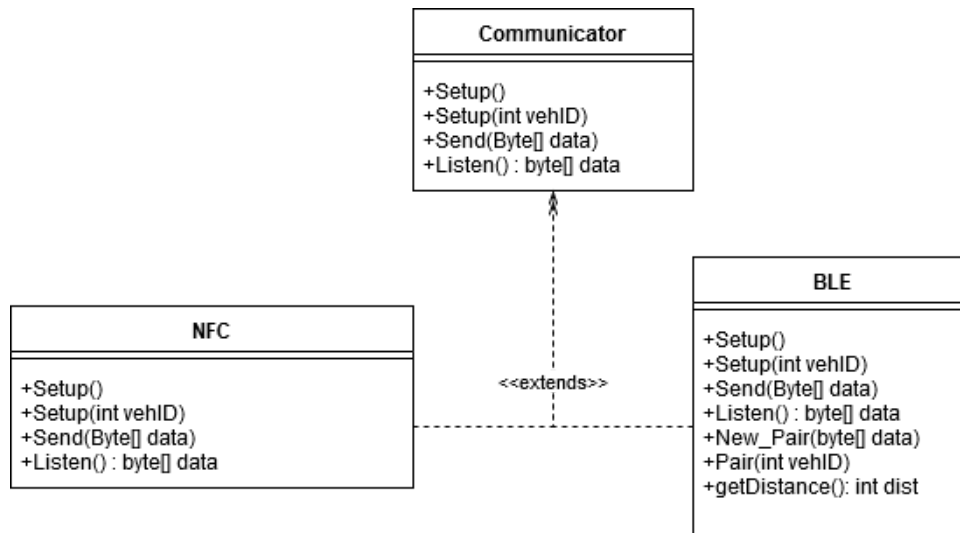
The above image is a very high-level overview of the vehicle side of the system. In this diagram I want to draw close attention to a few aspects:

The temporary key package would operate within a low power mode that would track the time remaining in real time for any active temporary keys. This is necessary because I want to ensure the time is tracked independently of the systems date/time to prevent users from altering the date to extend or remove access to the vehicle.

Data Logging will keep track of all the relevant information however this package may move to the BCM in implementation. I may also include a similar copy in BCM for complete records of every run of the lock along with the current run.

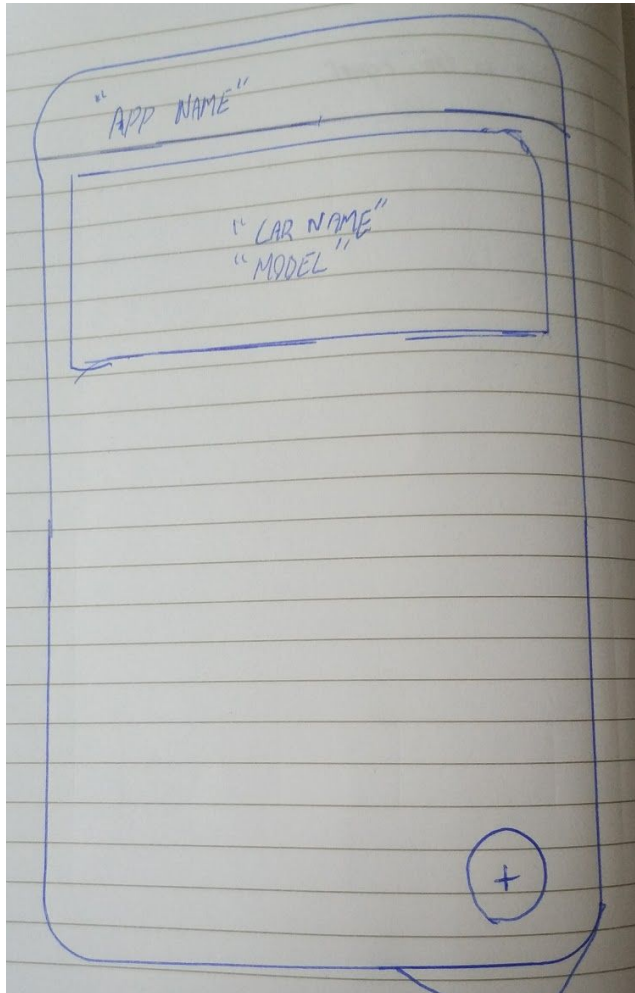
The security package will hold classes such as encryption and any other security elements required by the project. My reasoning for this is it should allow for the best maintainability for the security features and also easier extensibility as the standards for security change and update. This would also lower the dependencies within the classes.

I have also made a small diagram to show how I plan to implement the Android portion of the project.



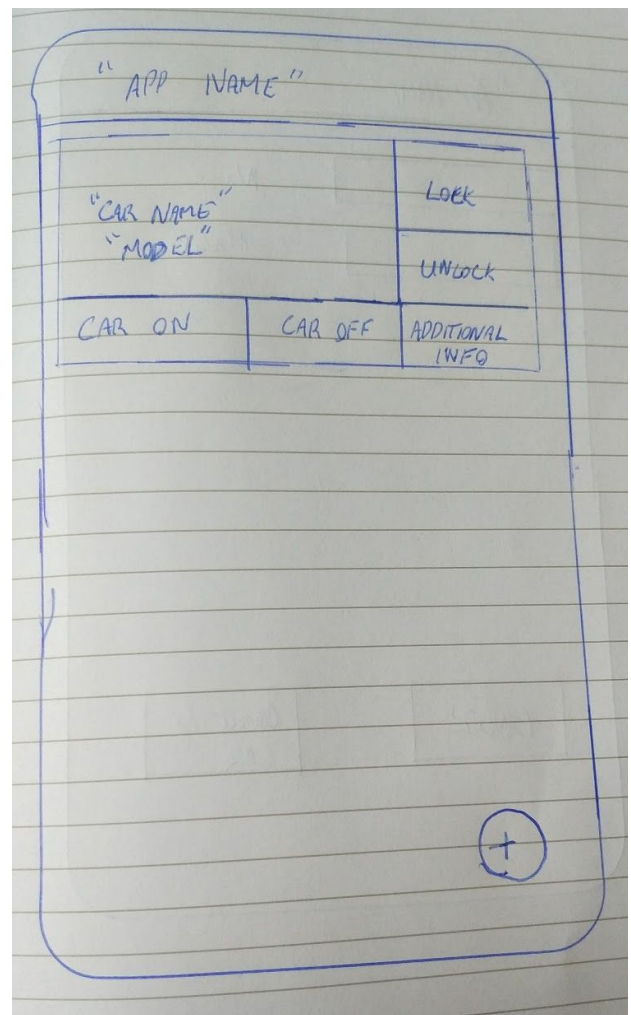
By making an interface for BLE and NFC to implement it means that it will be easier for me to develop a system that can easily swap between the two communication types. It also means I will be able to reuse a lot of the functions.

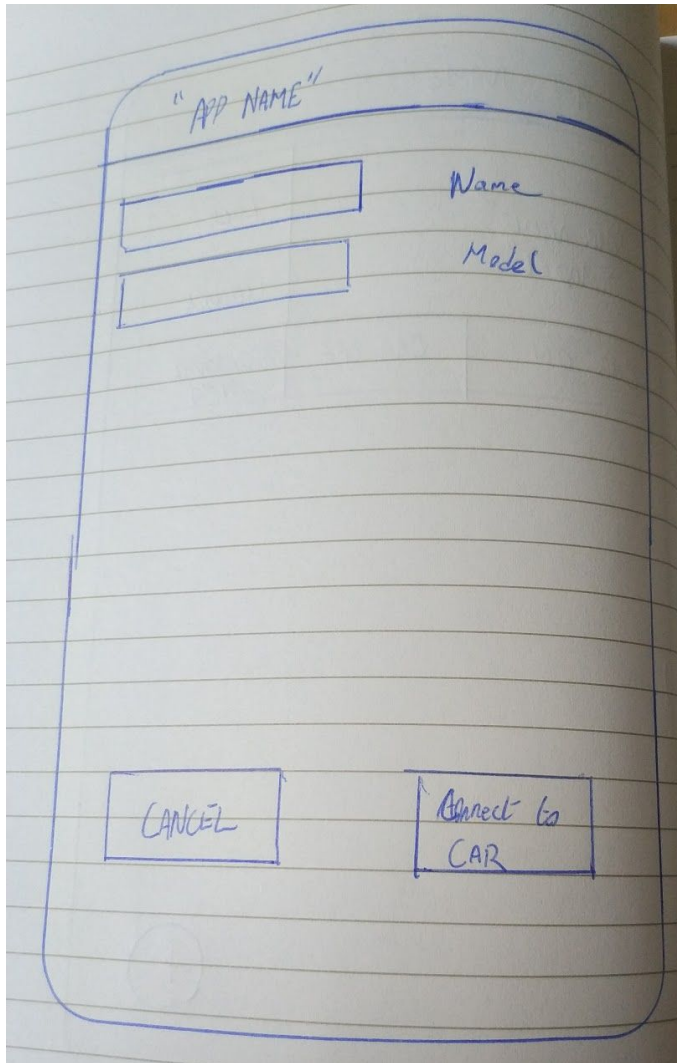
8.4. UI Designs



When a vehicle has been selected the image to the right will be displayed to show all available options to the user. If the user wishes to register a new vehicle to the application they must click the floating action button in the bottom right.

The image on the left will be the initial screen of the application where the user will see a list of potential vehicles where they can select the vehicle to be expanded for further options.





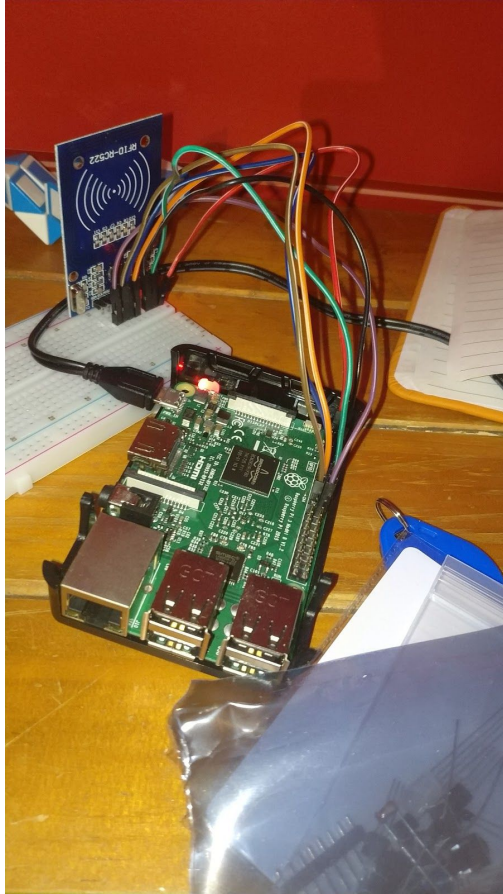
This is the screen where you can enter information on the vehicle then press connect to the car and the information should synchronise and store between the two devices.

9. Project Development

I started off my development as per my Gantt chart by working on implementing basic BLE and NFC. This led to further research on both technologies for the specifics of implementation of the technologies on my target platforms; Android and Raspbian, a Debian Linux offshoot made specifically for a Raspberry Pi. First, though I had to try to set up a very simple write/read system using in NFC and BLE. to

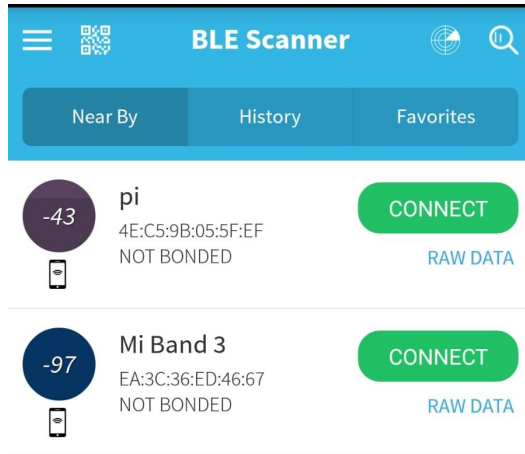
9.0.1. Initial Communication set-up

The first step was sourcing NFC hardware and I had to go with the cheaper option of the NFC MFRC522 chip over more capable and reliable hardware such as the PN532 for budgetary reasons. These two chips are the best supported NFC hardware peripherals available for the Raspberry Pi. Once I had the hardware the first step was to solder the pins to the chip and wire it to the Pi.



Then using the python I wrote code to detect a basic NFC card just to test it was working correctly. There is no real communication at this point as there is no application to receive information from the Pi. This was done using the Simple MFRC522 python library which allows for basic NFC card reading.

On the BLE side of things, I activated the BLE on the Pi and turn it into an advertiser to broadcast. I haven't gotten to the point of pairing at this point but I used an application called 'BLE Scanner' to detect nearby BLE advertisers to confirm that I had successfully activated it on the Pi. ([9.] Encarnacion 2016)



The application detects both Pi and my smartwatch, the mi band 3

From here I opted to lead with NFC as from my prior research I felt that it could have a greater impact on the project.

9.1. NFC Implementation

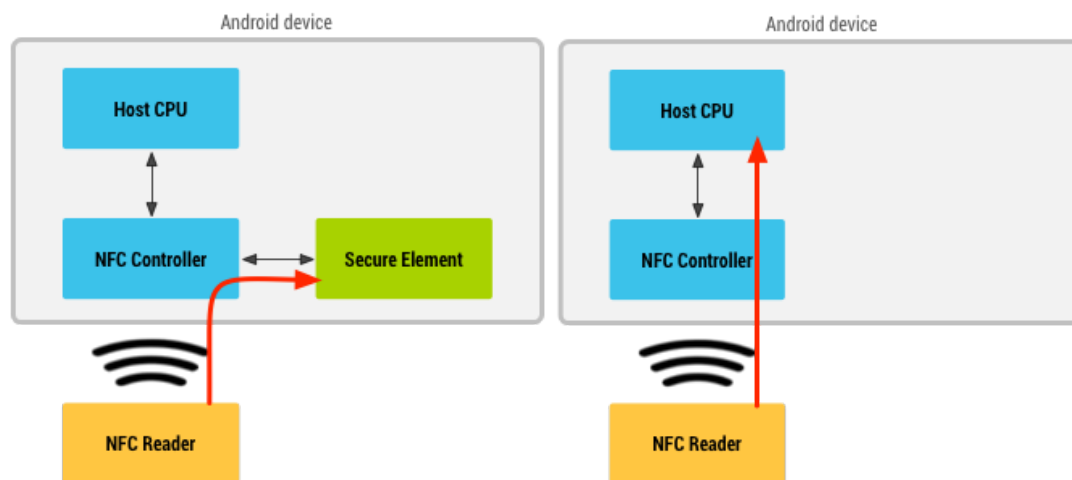
As per my pre-development plans, I began researching how to communicate NFC in a peer to peer or active-active manner. It was my plan from the beginning to implement the NFC portion like this however whilst I was following the tutorial on the Android development guide I wasn't satisfied with how it would work. This implementation would require the application to be open and active for it to be able to communicate with another NFC reader. The problem with this is the user would likely be required to place their phone in an area where they would be unable to

see the screen so they may be unable to open the application. Also, these extra steps would make the use of the phone as a key less efficient and as such, I searched for an alternate route for implementing. ([22.] “NFC basics | Android Developers” 2019)

9.1.1. Android HCE

I found a solution built in Android, one which is also used for mobile payments, that would allow NFC to communicate without the application being actively open; HCE (Host-Based card emulation). HCE allows the smartphone to act as a basic NFC smartcard. Using this my application can receive specific requests and respond with the relevant data and not just the static pre-written data that would be found on a standard NFC card. ([13.] “Host-based card emulation overview | Android Developers” 2019)

Standard card emulation within Android uses the NFC data within the secure element and as such no installed applications are involved in the transaction. However, with HCE, the secure element is bypassed and an Android application is accessed in order to facilitate the communication.



([13.] “Host-based card emulation overview | Android Developers” 2019)

To advise Android which application to utilise you must specify an AID

(Application ID). When an AID is received from the NFC reader will then transmit all subsequent messages to that application until disconnection.

```
<host-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/service_name"
    android:requireDeviceUnlock="false">
    <aid-group android:description="@string/card_title" android:category="other">
        <aid-filter android:name="F0010203040506" />
        <aid-filter android:name="F0394148148100" />
    </aid-group>
</host-apdu-service>
```

The code above should be set in the res/values folder within android studios. Despite its name the requireDeviceUnlock section only requires screen on so the added security of requiring the phone to be unlocked is not natively supported. The aid-filter section is simply a set of bytes that represent the AID for the program. Once the AID is received by Android it gets redirected to the following method.

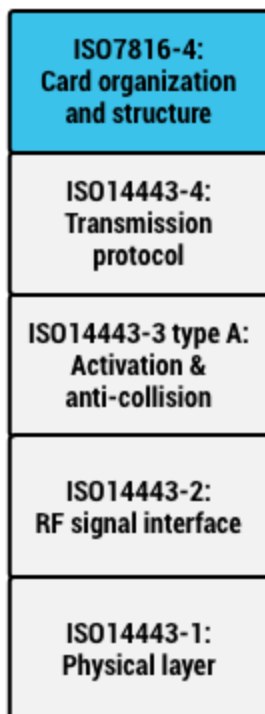
```
public class CardService extends HostApduService {
    private APDUCommand command = new APDUCommand(this.getApplicationContext());
    @Override
    public byte[] processCommandApdu(byte[] apdu, Bundle extras) {
        return command.process(apdu);
    }
}
```

When the method above gets called I have the option to open up the UI automatically for the user but for efficiency reasons I opted to not display the application but to instead use the Command design pattern to interpret and return the relevant information. Now that Android has received its initial AID all following communication will be via called APDUs, Application Protocol Data Unit, which is essentially a set of bytes that contains optional headers from both card and read and the CRC checks which will verify that the full data is sent uninterrupted. ([26.] Simon, J.2012)

Once I had my bare-bones HCE application up and running I tried to communicate with the NFC reader on the Raspberry Pi. I kept receiving empty data from Android and my application was never notified. I checked the status of the messages and it indicated that Android was throwing errors. From here I did further research and found that the MFRC class of NFC reader which I

had purchased was not compatible with card emulation. From here I was aware that Android was still receiving the NFC data and returning an error status so I decided to breakdown the MFRC library I was using and figure out where it was going wrong. Once this was done I found that the code would successfully initialise communication without Android throwing an error so I tried to figure out the network stack required for HCE communication. ([23.] *NXP ® NFC and Contactless Reader Solutions NFC TAG IC SOLUTIONS* n.d.)

9.1.1.1. Building the NFC Protocol Stack



The Image on the left represents the different network protocol layers for Android HCE. The lower 2 layers; ISO 14443-1 and ISO 14443-2 were fully supported by my NFC reader, the RC522, which allows for communication with very low-end MIFARE classic chips. The RC522 libraries didn't have full support of the following layers, as indicated by the Android returned error codes. I began researching how to build the full protocol stack by looking into ISO 7816 as it relies upon the previous protocol standards before being used. ISO 7816 is an international standard which is concerned with the electronic identification with smart cards ISO 7816-4 is more specifically related to secure command interchanging via radio frequency. Examples of other systems which use this standard would be HCE applications like Google pay as well as general wireless tap functions built into modern credit and debit cards. ([13.] "Host-based card emulation overview |

Android Developers" 2019, *Smart Card Technology FAQ* n.d., [16.] *ISO/IEC 7816-4 Second edition Identification cards -Integrated circuit cards - Part 4: Organization, security and commands for interchange Cartes d'identification -Cartes à circuit intégré - Partie 4: Organisation, sécurité et commandes pour les échanges Reference number ISO/IEC 7816-4:2005(E) 2005)*

To implement the full protocol stack I stripped back the Simple MFRC522 python library which I used in my initial setup of basic NFC card reading. The library utilises spi-tools which is supported by most Linux distributions and handles the GPIO transactions. From here I was able to manually input bytes to send to and from the NFC reader without the library repacking the data to be 14443-2 compatible. From this, I layered the code for basic ISO 7816-4 support.

9.1.1.1.1. ISO 14443-1 and ISO 14443-2

```
def MFRC522_Request(self, reqMode):
    status = None
    backBits = None
    TagType = []

    self.Write_MFRC522(self.BitFramingReg, 0x07)

    TagType.append(reqMode);
    (status, backData, backBits) = self.MFRC522_ToCard(self.PCD_TRANSCEIVE, TagType)
    if ((status != self.MI_OK) | (backBits != 0x10)):
        status = self.MI_ERR

    return (status, backBits)
```

This method is mostly unchanged from the original library but it is now being passed the byte “0x52” which signals a wake up for all NFC reader types, this was easier than trying to find the specific wake up request for Android HCE or other smart card implementations. This represents the first 2 layers of the protocol stack, ISO 14443-1 and ISO 14443-2. As this is the unaltered code I thought it best to use this opportunity to explain some of the method calls that will be seen in some of the code that I wrote. “Write_MFRC522” will prepare the reader to write to an NFC card correctly. MFRC522_ToCard will send raw bytes to the card and return a return value which is generally a status return e.g. success or error. Since this is a Smart Card and all messages from the card require a specific request from the reader we will be retrieving our desired return information from the card with the return to this method. A success here should see a return value of a single byte, “0x00”.

9.1.1.1.2. ISO 14443-3 (Anti-Collision and Card selection)

For this, we send the bytes “0x93 0x20”. “0x93” is the anti-collision command and “0x20” is the request for the cards UID, and identifying number, which is used to determine which card is in communication and helps prevent against DOS or eavesdropping attacks. The smart card should then return its uid which is, a few bytes long with the final byte representing a checksum. Some cards will have long UIDs which will require a more complicated Anti-Collision loop but for Android HCE I have found the preceding protocol to be sufficient. Finally, we send “0x93 0x70 [UID] [Checksum] [CRC]” where the UID and checksum are what we received from the smart card previously and the CRC is our own calculated checksum. This checksum is calculated using a method already provided within the MFRC522 library. A success here will receive a “0x20 [crc]” from Android.

9.1.1.1.3. ISO 14443-4 (Transmission Protocol)

This step is a way for both the reader and the smart card to clarify some parameters about the other. The first part of this, however, is to communicate maximum frame sizes and a request to select the target device for continued communication. So from the reader send “0xE0 0x50 [CRC]”. The return values varied however so long as the return status is not “0x02” indicating error then it has been successful. If not sending any additional optional parameters then the process can continue. For this project, additional parameters are never needed as the communication was basic request/reply.

9.1.1.1.4. ISO 7816-4

As mentioned earlier you must first send the AID to communicate to Android which application to select. Most of the header on this frame is optional so I opted to not use it. There is a single mandatory byte which must be toggled. In the first frame, which requests the AID, the first byte should be a “0x02”. However, when sending the first APDU it must be a “0x03” and then back to “0x02” for the following transmission and so on. So the first frame should be “ 0x02 [AID]

[crc]” and if successful will return a request to try again later indicated by a short 4-byte message where the first byte is “0xF2” and the second byte varies with the final 2 bytes being the CRC. When this is received you must send the exact message back until Android has prepared a response. It generally only requires a single additional request however it can require more so this should be sent in a loop like the following example.

```
def AndroidRetry(self, status, backData):
    while(self.verify_retry(status, backData) == 1):
        #print("\nAndroid wants more time")
        status, backData = self.READER.MFRC522_ANDROID_SEND_MESSAGE(backData)
    return status, backData
```

Once you no longer receive a 4-byte message beginning with “0xF2”, if there is no error status, you will receive the return value set in the processCommandApdu() method in Android. Once this is done you can proceed to send APDUs as the correct Android application should already be selected, using the format “[Toggle byte 0x03 or 0x02] [APDU] [CRC]”. (Simon, J.2012)

9.1.1.1.5. Disconnection

This step seems to be a best practice optional extra as it should automatically disconnect after a brief period of no communication but to confirm deactivation of communication simply send the following “0xC2 [CRC]”. Android will return the same to confirm. ([18.] Leszczyński 2018)

9.2. BLE

I lead with implementing the NFC side and as it relied so heavily on python. My goal was to try and implement the BLE side of the project in python but first I had to figure out how to communicate between two devices securely since at this point I had only managed to turn the Raspberry Pi into a BLE advertiser that would broadcast its information. I used the BlueZ package for this as it is one of the best-documented libraries for Linux BLE and my goal was to use that knowledge and then transfer to a python based library such as BluePy. At this point, however, I ran into an issue. I couldn’t figure out how to send information in a back and forth manner between the Pi and Android via BLE.

9.2.1. How BLE communication works

BLE uses two sides; a peripheral and a central. The peripheral sets its GATT, Generic Attribute, to contain a specific service from a predefined list set out by the Bluetooth SIG. These services have predefined characteristics which are suited to their specific tasks. This information is then made available to the central which is essentially a server which can accept multiple BLE connections and read their data. It can also send some very specific responses which are almost always an acknowledgement that data has been read and then the peripheral can update or alter the information as required. Unfortunately, there is no specific GATT service for something like a lock but there are services like `generic_access` which could be used to send Strings to the BLE central. ([28.] “Viewer | Bluetooth Technology Website” 2019)

9.2.2. Why BLE is too limited for this smart-lock?

Due to the nature of BLE discussed above it would be possible to send the key data from one device to the other but it wasn't made with the intention of having the devices intercommunicate back and forth, which is what my key system would require. It would be possible to have the devices trade places as required from peripheral to central and back but it is beyond the scope of this project to do so within the limited time frame. There are much simpler methods of implementing the “Smart Lock” function which could include moving the key generation to the Android portion of the project and setting the Pi side to store the information when requested. Unfortunately, this has specific drawbacks which would make the system too insecure. For instance, there would be no way for me to implement a rolling key or to request specific information as in this model the Pi has to be the central to enable multiple peripherals to connect. There would also be no way for me to implement my own encryption as the data has to be in a specific format. While it wouldn't just be publicly exposed data it is possible that someone could eavesdrop and resend the key data at a later point for malicious reasons.

For the foregoing reasons and because NFC ended up requiring more focus than anticipated, I decided to move my efforts in that direction as I wouldn't be able to implement a secure system. This doesn't mean however that BLE can't be considered for a smart key in an actual vehicle but

it would require further research and implementation work to make satisfactory from both a reliability and security point of view. ([2.] Avigezer 2017)

9.3. Security

This area of the project was particularly challenging. I was aware that an idea like this for a vehicular smart key that would be expected to work across different vehicles would have to be open source. The implications being that malicious users would have access to the inner workings of the code and could potentially reverse engineer the code. So I needed to create data that couldn't be reverse engineered or copied in any way.

9.3.1. Encryption

The encryption algorithm I opted to use for this project is AES (Advanced Encryption Standard). This project uses the AES 256 version which means the key length is 256 bits long as it is the most secure version of the algorithm. Generally speaking the larger the data the harder it is to decrypt. AES was developed for the US government in the early 2000s and has since been adopted by various departments such as the NSA. As of yet AES 256 is thought to be uncrackable but as hardware improves over time it may one day be cracked. However, because in both the Pi side and application side it is a separate class it should be easy to update the system with whatever advancements time brings. ([21.] Nash 2018)

9.3.2. Key Data

The key data for my project is an array of random alphanumeric characters. This means that there are 26 lowercase letters, 26 uppercase letters and 10 numbers in the range 0-9 a total of 62 potential characters. For bandwidth and storage reasons I have opted to use an array size of 250. This translates to a 449 digit number of potential keys. That number is calculated by the number of potential characters to the power of the size of the key. In this case 62^{250} . To try contextualize the impossibility of brute forcing this number if you were to try a new unique combination every Planck second, the shortest unit of time possible, which is significantly faster

than any computer can operate at then you wouldn't even reach 0.0001% (Actually around $4^{(10^{-43})\%}$) of all possible permutations before the death of all life and decomposition of all solid objects in the universe. ([24.] Pimbblet 2019). However, as large as that number is if malicious users managed to eavesdrop on or access the key data just once then it would be compromised forever. So there needs to be more than just a large key and encryption applied to it. As a result of this, I have a few additional steps in place to help protect the precious key data that will help make it more secure.

9.3.2.1. Rolling key

This is an idea inspired by the WKS (Wireless Key System) mentioned in research; state of the art. It involves having a key that changes on both systems to the same number on both systems without trading any information. This poses a large challenge as the change needs to be predictable to both the vehicle and phone side of the system but unpredictable to anyone with access to the source code or even eavesdropping on communication between car and phone. I had many ideas like having all items in the array shift their positions but this movement is too predictable and not complicated enough for peace of mind or future proofing. The solution I arrived at was quite simple. When setting up a new key the vehicle will not only generate and send the key data but it will also send an unsigned 64-bit integer, greater than 0 and not a multiple of the keyData size, to the phone which I call the roller. Both the phone and vehicle will store the roller in their respective secure environments, SQL databases for the purposes of this project. Because both the phone and vehicle have access to the roller, key-data and original character array so we can use this information in the following calculation:

$$([KeyData\ Array\ Position]+1)*([Character\ Location\ in\ original\ character\ array]+1) \\ * [Roller] \% [KeyData\ Size]$$

This simple calculation will return a number between 0 and 250. We do the computation for each character in the key data array and derive a position for that character because it is within the size of the array. If the roller is 0 or a multiple of the keyData size then the result of this calculation

will always be 0 so it is not a valid roller. Invalid rollers lead to predictable key data which will lead to vulnerabilities within the vehicle. This calculation can lead to duplicate numbers so when a character is trying to be written in a position that is already taken then the character already in the array and all subsequent characters will be shifted down the array in order to make room for the new character. This adds further complexity to the algorithm which functions to further hide the key data as it obscures what the roller could be. In order to test the Key Rolling algorithm, I wrote a simple java class to apply my algorithm to an array of integers where the original key has '1' in position '0' and '2' in position '1' and so on for all 250 slots in the array. The reason for testing on this array instead of the alphanumeric data is two-fold. Firstly because there can be duplicate data in the alphanumeric array the tests are harder to run. Secondly running the code on integers is much quicker for millions of calculations than to do so on a char array. It should be noted that while the char array is less efficient to apply the rolling algorithm to than an integer one the difference is negligible when only running on one key at a time.

```

94 public static int[] rollKey(int[] keyData, int key_roll){
95     int[] newkeyData = new int[keyData.length];
96     if(key_roll == 0)
97         key_roll = 1;
98     else if(key_roll < 0)
99         key_roll = key_roll*-1;
100     for(int i = 0; i < keyData.length ; i++){
101         int newPos = ((i+1)*keyData[i]*key_roll) % (keyData.length-1);
102         if(newPos < 0 ){
103             newPos = newPos * -1;
104         }
105         if(newkeyData[newPos] > 0){
106             int temp = newkeyData[newPos];
107             newkeyData[newPos] = keyData[i];
108             newPos++;
109             if(newPos >= keyData.length){
110                 newPos = 0;
111             }
112             while(newkeyData[newPos] > 0){
113                 int temp2 = newkeyData[newPos];
114                 newkeyData[newPos] = temp;
115                 temp = temp2;
116                 newPos++;
117                 if(newPos >= keyData.length){
118                     newPos = 0;
119                 }
120             }
121             newkeyData[newPos] = temp;
122         } else {
123             newkeyData[newPos] = keyData[i];
124         }
125     }
126     return newkeyData;
127 }

```

The code above is used for rolling the Integer Key. The if statement on line 96 ensures the key roll is greater than 0 as it would end up just reversing the array or causing errors otherwise. Line 100 onwards is where the actual key rolling occurs. You can see that on line 105 it is checking if a number has already been written in this place in the array. If it has then it stores that value in temp and overwrites it in the array. Then in the while loop on line 112, you will find the code for moving all subsequent elements down along the array. The if statements on line 117 and 109 ensure that the array loops and doesn't go out of bounds.

9.3.2.1.1. Rolling Key Tests

In an attempt to verify the amount of variance my rolling key algorithm gives to the Key data I ran a suite of tests to check for differences between the original array of numbers, 1-250, and the rolled array. For each test, I tested all valid keys from 1-25,000,000 in a loop and printed the results of the data that I was testing for.

9.3.2.1.1.1. Check Shared locations

```
61 public static int compareKeys(int[] key1, int[] key2){
62     if(key1.length == key2.length){
63         int sameCount = 0;
64         for(int i = 0 ; i < key1.length; i++){
65             if(key1[i] == key2[i]){
66                 sameCount++;
67             }
68         }
69         return (sameCount*100)/key1.length;
70     }
71
72     else
73         return -1;
74 }
```

The code sample shows a test checking each position in the arrays to see if the number is the same between the original array and the rolled array at the nth position (i.e. `keyData[i] == roledKey[i]`). When testing for 5% shared locations the test found that 0 /25,000,000 reached that threshold. When the threshold was lowered to 2% the number of shared locations ended up at 87089 /25,000,000 which means that only about 0.35% of rolled arrays shared more than 6 numbers with the original array while 0 shared more than 12 number locations.

9.3.2.1.1.2. Shared Partners Test

```
130 public static void compareKeys(int[] arr){
131     int count = 0;
132     for(int i = 1; i < arr.length - 1; i++){
133         if(arr[i] == arr[i-1]+1 || arr[i] == arr[i+1]-1 || arr[i]
134             == arr[i-1]-1 || arr[i] == arr[i+1]+1){
135             count++;
136         }
137     }
138     System.out.println("Paired Data : "+count+" / "+arr.length );
139 }
```

This test checks if the characters in the rolled array have the same adjacent characters as they had in the original array even if the partnership is flipped i.e if it was originally “a,b” and was rolled to “b, a” it would be enough to trigger this test. The point of this test is to check if data has been universally shifted along the array or if the data has simply been reversed in the array. Again this test shows promising results, at a threshold of 5% shared neighbours per rolled key resulted in

26,1072/25,000,000 which equates to about 1.04% of the rolled arrays triggered whereas at the lower threshold of 2% the result is 4,066,944/25,000,000 or 17.8% sharing neighbours. While this number may seem large it is still an incredibly small number considering the threshold and this means that there is enough variance between the keys that they should be sufficiently unpredictable. It should also be noted that as mentioned earlier each set of shared neighbours would trigger the test twice, once for each character so it isn't as large as it may seem.

9.3.2.2. Specific Element Request

This, the final portion of the key is another step to safeguard the key data itself. The general idea of this step is that the vehicle will request specific sections of the key data as opposed to the entire key. For this project, 5 requests with different positions between 0 and 250 and will be expecting the key-data at the listed positions along with the additional 10 items in the array. This decision was inspired by online banking where you are requested to only enter a few select digits of an identifier code as opposed to the entire thing. Pairing this method with the rolling key preserves the privacy of both pieces of data. This does however drastically lower the potential for brute force the key data to

416,470,721,148,177,771,532,676,369,918,117,947,181,737,776,340,608,755,765,950,803,371,888,848,050,241,916,448,538,624

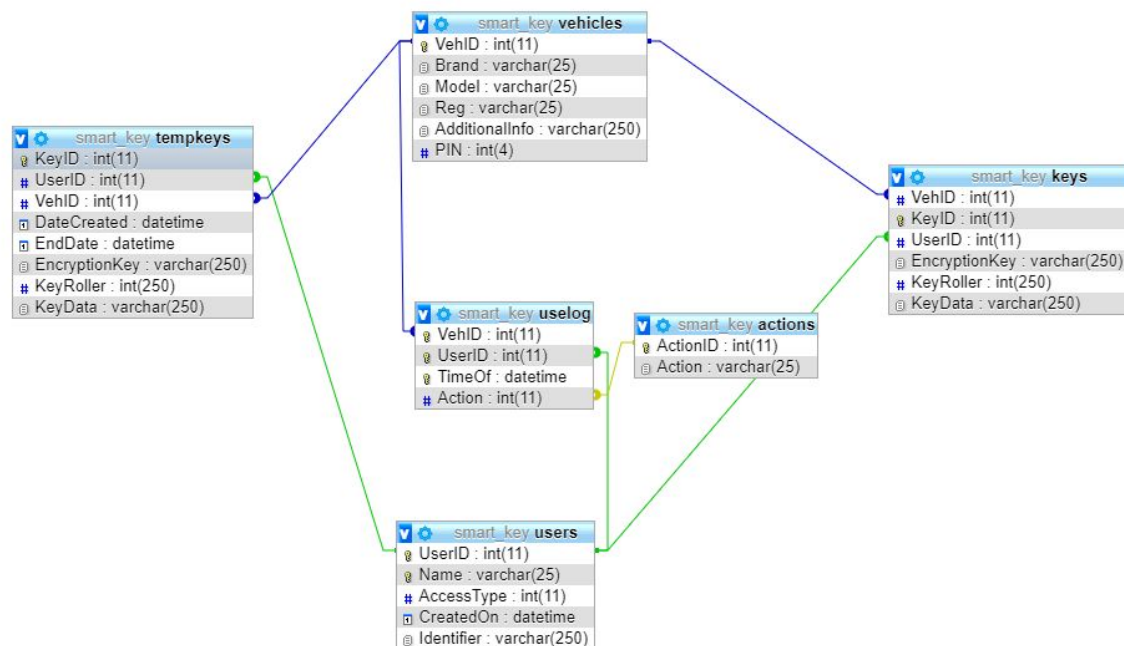
This is a 90 digit number and much more possible to brute force but there could be procedures built into the vehicle to prevent this kind of attack. A waiting system once an attack of this kind is detected, based on a large number of incorrect attempts in a short period of time. This isn't something I wanted to focus on as it would take up too much of my time and not add to the core functionality of the system. But every request from the vehicle should be unique meaning that every time the expected key data will change and when paired with a rolling key every successful entry will cause the key to roll leading to further data obscuring from potentially malicious users. Thanks to this the full key data will never be exposed after it is initialised so it

becomes nigh impossible for someone to eavesdrop especially when the key is immediately updated even if some of the elements are temporarily exposed.

9.4. Data Storage

For this project, I am using a MySQL database to store information on the Pi side coupled with an internal SQLite database for secure storage on Android. I am not too concerned with the security of the SQL database on the Pi as for someone to gain access to this information they would need access to the vehicle internals which if already acquired means that they have bypassed the lock mechanism anyway.

9.4.1. Vehicle Side



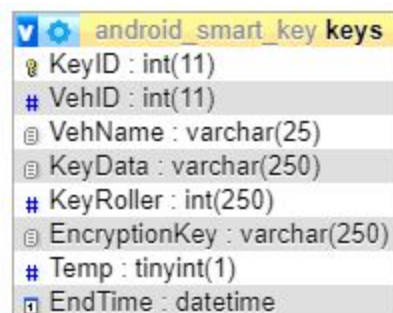
The schema for my database is documented above. The vehicles table is only for demo purposes so that the Pi could emulate many vehicles and only be accessible by certain keys as I only have one set of hardware but in an actual vehicle, this table would be mostly useless.

The keys and tempKeys tables are mostly the same, they both store the Key Roller as an int and they store the encryption key as a string which can be converted to byte form to encrypt and decrypt the information. The only difference between these two tables is that the tempkeys stores DateTime for the creation and intended destruction of the key. This ensures that the key will remain stored but can be ignored when required.

The users' table contains a few fields for identification but the access type field is in there for future use to allow for different user ranks to enable or disable certain features such as admin access.

The logging table tells when the car has had an interaction and logs it and I've created a separate actions table so the specificity with which the action is described can be changed and this could log all variations of interactions and try track users where possible however when the user isn't recognised it could be left null just to show that someone tried to gain access to the vehicle be it nefarious or benign.

9.4.2. Android Database



	android_smart_key keys
🔑	KeyID : int(11)
#	VehID : int(11)
📄	VehName : varchar(25)
📄	KeyData : varchar(250)
#	KeyRoller : int(250)
📄	EncryptionKey : varchar(250)
#	Temp : tinyint(1)
📅	EndTime : datetime

Here I decided to put all the data in one database so that the application will only be required to search a single database for their key. This was done for speed reasons in case Android takes too long to respond via NFC the connection could timeout.

9.5. Temporary Keys

These work similar to the normal key except when originally created they have a minutes parameter which is sent to the Android device. On both the pi and Android device, the minutes parameter is added to the current timestamp and gets stored in the respective databases. Every time the Android application is opened it clears all keys before the current timestamp and even if you were to change the time on your device then the pi side still checks for keys after the current timestamp so you couldn't gain access to the vehicle outside your designated time span.

The only potential flaw in this system is it relies upon system time of the Pi which could be rolled back to extend access however there is a solution to this. Separating the system for key storage from the main dashboard computer in the vehicle would mean that a low power mode would be more easily implemented as it is not tied into the entertainment system and other full power features but it would mean that any changes to system time would not affect the key system.

9.6. UI

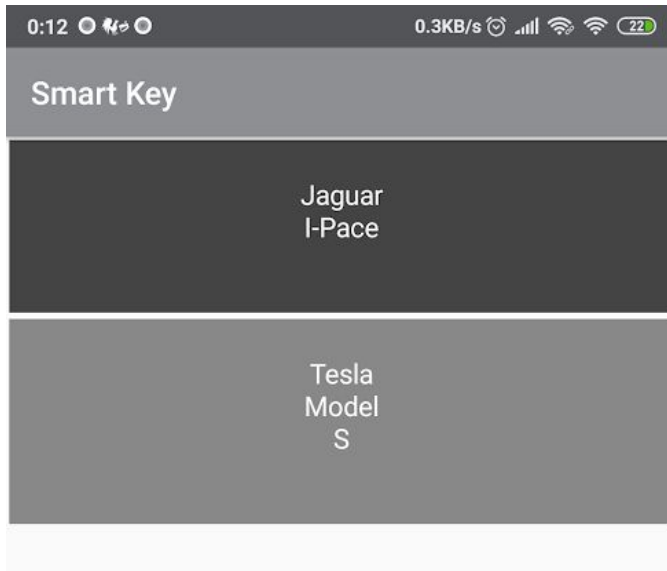
9.6.1. Pi UI

Due to the limited power of the Pi, I decided to use basic command inputs but this added the complication of thread management since the inputs had to run on a separate thread to the NFC reader. This issue was mitigated by using the command design pattern to check user inputs and setting a global variable to signal to the reader portion what actions it should be carrying out. I was sure to run only the user inputs on a separate thread as the threads weren't capable of anything more intensive.

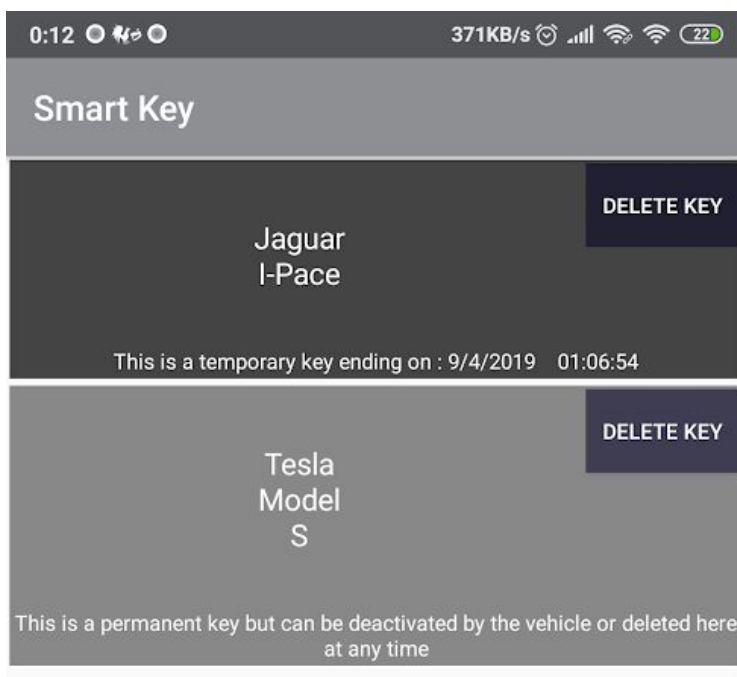
```
Please enter a command or type "-help" to view options:
Enter a valid id to emulate:
3
Please enter a command or type "-help" to view options:
-help

Enter one of the following commands:
-- "-c <PIN> <UserName>"          to create and add a new user key
-- "-t <PIN> <UserName> <No. of Minutes>"  to create and add a new temporary user key
-- "-q"                          to end the program
-- "-help"                       to view options
Please enter a command or type "-help" to view options:
█
```

9.6.2. Android UI



For the UI here I wanted the system to be foolproof and as simple as possible and I realised that my original design required the user to input some information already available on the Pi and could be sent over automatically upon key creation to automate the entire process. So now you simply tap your phone and it automatically stores and displays the key. I use a different colour to differentiate between a temporary key and a permanent one. These keys can be selected to be expanded for further information as can be seen in the below image



10. Retrospective

10.1. Overview

While working on this project I faced many issues but also learned a lot that I will be taking forward into the workplace and my future projects. The first and I think the main learning outcome is to try to maintain scope. In the beginning, I had grand broad reaching ideas for what this project could and should be. While I still think those ideas are great and should exist in this system I just didn't have the time or resources to implement all of them within the time frame of this project.

Also, I think I would have benefitted from starting the implementation earlier. In the timeline followed for this project I started out with the bulk of my research and then later started the implementation. However, I ended up requiring further research during the implementation and learning at a much faster rate as I had the context of my current task to help guide my research. I think had I begun implementation sooner I could have encountered issues and adapted to counter them more effectively. I'm not suggesting that my research was fruitless but rather that if I balanced the workload more evenly throughout the project I think I would have found greater success.

10.1.1. Implementation Issues

While developing this project several issues were encountered which altered the timeline and initial plan for the project. These weren't necessarily a negative experience as it led to me doing further research and learning more than I had initially thought required. One such issue was mentioned in NFC implementation and that was the lack of support for Android HCE/ISO 7816 in the NFC chip I had purchased. While this was a big hurdle for me to overcome and altered the time frame of the project I learned much more about NFC and the readers' requirements and capabilities in doing so and also added functionality not otherwise available in the MFRC522

chip used in this project. There is no publically available code enabling ISO 7816 requirements on this chipset but perhaps this document and my project could help others who encounter the same issues with the chipset.

One major revelation that I had while working on the project was that I hadn't initially focused as heavily on security as I should have. Since then the scope of the project was altered somewhat to include adding security elements. Security, in line with the rest of the project, took more time than initially anticipated. The key system and key rolling required multiple iterations and a lot of testing until it was satisfactory. These elements were then further complicated by the fact they are unique to this project so there aren't any online resources to help test how secure they are so I had to figure out my own testing to verify the security of the key.

The other major issue which I'll be writing about but arguably the largest is that of BLE and its limitations. I have already mentioned this earlier in the [BLE implementation section](#) so won't be going into too much detail here. This was a major hindrance for this project as I was effectively losing half of the initial plan but it ended up being a benefit as the project scope had grown to a scale unmanageable for me and had I continued to implement the entire plan including the BLE elements I may not have been able to make a stable system work as smoothly as the end product does. As with the other major issues, it did provide a great learning experience as I found out much more about how BLE GATT profiles work and BLE's capabilities.

11. Conclusions

11.1. Delivered product

As stated previously due to issues encountered the end product does not contain every feature that was mentioned in the initial project outline. I was not expecting to be able to fully build the system and I am satisfied that I have managed to implement the core functionality of the smart key. This includes the added extra of temporary keys that automatically become inactive after the set time frame. I was disappointed that I was unable to implement a BLE prototype but I am happy with the added research my trials provided.

11.2. Which is better, NFC or BLE?

The title of this project is “Researching BLE (Bluetooth Low Energy) and NFC (Near Field Communication) for use as a personal identification service to unlock a vehicle.” and I wanted to find which was best suited to both general personal identification and then more specifically as a smart key. From my research and while implementing the project I have arrived at the conclusion that all digital identifiers using a smartphone should use NFC over BLE which is actually contrary to my initial inclination, [mentioned earlier in the project](#).

From the research, BLE boasted greater versatility and capabilities. The main advantage it appeared to have over NFC was its much greater range, up to 100 metres. This range can still be seen as a benefit for example in a smart lock it could unlock upon approach to save you time but this, in my opinion, is not secure enough as the range detection methods currently used for BLE aren't accurate enough and may leave you or your property vulnerable.

The main reason however for selecting NFC over BLE is simply reliability. A BLE system could suffer from interruption from being in a busy environment and is also much more susceptible to

malicious attacks limiting its use. On the other hand, NFC is much less vulnerable to attacks and even when attacked the attacking device will be visible thanks to its short range so it can be detected and removed more easily.

This said I don't think that BLE should be ruled out of a smart key system by any means. I think that for the time being there should be a mandatory NFC key and an optional, potentially less secure, BLE one. The reason for this is a key above all else must be secure and reliable and if it were to fail either of these then the entire system is useless. In this case, even if the BLE key fails for reason of interruption, attack or even a simple connection error then you can still tap your phone to the NFC reader and be able to access your vehicle. In a system making use of both BLE and NFC you also have the added benefit of bypassing some of the awkwardness of initial BLE pairing by using NFC to automatically do this for you. I also think that were there to be further support from the Bluetooth SIG, even a dedicated smart key GATT profile, then perhaps the smart key could exist using only BLE but as of writing this report, I feel like it does not suffice on its own.

12. References

- [1.] About Smart Cards : Frequently Asked Questions [online] (2019) *Secure Technology Alliance*, available: <https://www.securetechalliance.org/smart-cards-faq/> [accessed 8 Apr 2019].
- [2.] Avigezer, S. (Ed.) (2017) How To Communicate with Bluetooth Low Energy Devices on Android [online], *Medium*, available: <https://medium.com/@avigezerit/bluetooth-low-energy-on-android-22bc7310387a> [accessed 8 Apr 2019].
- [3.] Bluetooth Pairing Part 1 Pairing Feature Exchange | Bluetooth Technology Website [online] (2016) *Bluetooth.com*, available: <https://blog.bluetooth.com/bluetooth-pairing-part-1-pairing-feature-exchange> [accessed 8 Apr 2019].
- [4.] *Building Digital Key Solution for Automotive Content* (n.d.) available: <https://carconnectivity.org/wp-content/uploads/CCC-Digital-Key-White-Paper.pdf> [accessed 8 Apr 2019].
- [5.] Coskun, V., Ozdenizci, B., Ok, K. (2013) “A Survey on Near Field Communication (NFC) Technology. *Wireless personal communications*, 71(3),” 2259–2294.
- [6.] D’Angelo, M. (2017) Tesla Model 3 Mobile App Adds “Phone Key” and Frunk Unlock Feature (Screenshots) [online], *TESLARATI*, available: <https://www.teslarati.com/tesla-model-3-app-phone-key-features-screenshots/> [accessed 8 Apr 2019].
- [7.] Deordica, B., Alexandru, M. (2014) “ADVERTISEMENT USING BLUETOOTH LOW ENERGY,” *Review of the Air Force Academy*, (2), 2014, available: http://213.177.9.66/ro/revista/Nr_2_2014/65_DEORDICA_ALEXANDRU.pdf [accessed 8 Apr 2019].
- [8.] DunderGifflin, Imgur (2018) *Concept Key for a Mercedes Benz Maybach 6* [online], Imgur, available: <https://imgur.com/TptQAQU> [accessed 8 Apr 2019].
- [9.] Encarnacion, Y. (2016) Raspberry Pi 3 as an Eddystone URL Beacon [online], *Hackaday.io*, available: <https://hackaday.io/project/10314-raspberry-pi-3-as-an-eddystone-url-beacon> [accessed 8 Apr 2019].
- [10.] Getting Started with Bluetooth Low Energy [online] (2019) *O’Reilly | Safari*, available:

<https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch04.html>
[accessed 8 Apr 2019].

- [11.] Gomez, C., Oller, J., Paradells, J. (2012) “Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology,” *Sensors*, 12(9), 11734–11753, available: <https://www.mdpi.com/1424-8220/12/9/11734> [accessed 8 Apr 2019].
- [12.] History of Near Field Communication - NearFieldCommunication.Org [online] (2010) *Nearfieldcommunication.org*, available: <http://nearfieldcommunication.org/history-nfc.html> [accessed 8 Apr 2019].
- [13.] Host-Based Card Emulation Overview | Android Developers [online] (2019) *Android Developers*, available: <https://developer.android.com/guide/topics/connectivity/nfc/hce> [accessed 8 Apr 2019].
- [14.] howstuffworks.com (2001) How Remote Entry Works [online], *HowStuffWorks*, available: <https://auto.howstuffworks.com/remote-entry.htm> [accessed 9 Apr 2019].
- [15.] iotbymukund (2016) How to Calculate Distance from the RSSI Value of the BLE Beacon [online], *IOT and Electronics*, available: <https://iotandelectronics.wordpress.com/2016/10/07/how-to-calculate-distance-from-the-rssi-value-of-the-ble-beacon> [accessed 8 Apr 2019].
- [16.] *ISO/IEC 7816-4 Second Edition Identification Cards -Integrated Circuit Cards - Part 4: Organization, Security and Commands for Interchange Cartes d’identification -Cartes à Circuit Intégré - Partie 4: Organisation, Sécurité et Commandes Pour Les Échanges* Reference Number *ISO/IEC 7816-4:2005(E)* (2005) available: http://www.embedx.com/pdfs/ISO_STD_7816/info_isoiec7816-4%7Bed21.0%7Den.pdf [accessed 8 Apr 2019].
- [17.] Kamkar, S. (2015) *Drive It like You Hacked It.*, available: <https://samy.pl/defcon2015/2015-defcon.pdf> [accessed 8 Apr 2019].
- [18.] Leszczyński, M. (2018) Communication between Android Host-Based Card Emulation and a Microprocessor System with NFC Transceiver Frontend | IceDev [online], *Icedev.pl*, available: <https://icedev.pl/blog/nfc-frontend-android-hce-apdu/> [accessed 8 Apr 2019].
- [19.] Mohd, S. (2019) “YouTube,” *YouTube*, available: <https://www.youtube.com/watch?v=3hzrp-3JIQQ> [accessed 9 Apr 2019].
- [20.] Mostafa, M., Allah, A. (2011) “Strengths and Weaknesses of Near Field Communication (NFC) Technology Strengths and Weaknesses of Near Field Communication NFC Technology,” *Type: Double Blind Peer Reviewed International Research Journal Publisher: Global Journals Inc*, 11, available: https://globaljournals.org/GJCST_Volume11/7-Strengths-and-Weaknesses-of-Near-Field-Communication.pdf [accessed 8 Apr 2019].

- [21.] Nash (2018) What Is AES-256 Encryption? - Cyclonis [online], *Cyclonis*, available: <https://www.cyclonis.com/what-is-aes-256-encryption/> [accessed 8 Apr 2019].
- [22.] NFC Basics | Android Developers [online] (2019) *Android Developers*, available: <https://developer.android.com/guide/topics/connectivity/nfc/nfc#p2p> [accessed 8 Apr 2019].
- [23.] NXP ® NFC and Contactless Reader Solutions NFC TAG IC SOLUTIONS (n.d.) available: <https://www.nxp.com/docs/en/brochure/939775017564.pdf> [accessed 8 Apr 2019].
- [24.] Pimbblet, K. (2019) The Fate of the Universe: Heat Death, Big Rip or Cosmic Consciousness? [online], *The Conversation*, available: <http://theconversation.com/the-fate-of-the-universe-heat-death-big-rip-or-cosmic-consciousness-46157> [accessed 7 Apr 2019].
- [25.] Rahul, A., Krishnan G, G., Krishnan H, U., Rao, S. (2015) “NEAR FIELD COMMUNICATION (NFC) TECHNOLOGY: A SURVEY,” in *International Journal on Cybernetics & Informatics (IJCI) Vol. 4, No. 2*, available: https://www.researchgate.net/publication/276534674_Near_Field_Communication_NFC_Technology_A_Survey [accessed 9 Apr 2019].
- [26.] Simon, J. (2012) What Is an APDU? [online], *Jguru.com*, available: <http://www.jguru.com/faq/view.jsp?EID=470744> [accessed 8 Apr 2019].
- [27.] Thrasher, J. (2018) RFID versus NFC: What’s the Difference between NFC and RFID? [online], *RFID Insider*, available: <https://blog.atlasrfidstore.com/rfid-vs-nfc> [accessed 8 Apr 2019].
- [28.] Viewer | Bluetooth Technology Website [online] (2019) *Bluetooth.com*, available: https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.heart_rate.xml [accessed 8 Apr 2019].
- [29.] Zegeye, W., Dean, R., Moazzami, F., Astatke, Y. (n.d.) *Exploiting Bluetooth Low Energy Pairing Vulnerability in Telemedicine*, available: https://www.morgan.edu/Documents/ACADEMICS/DEPTS/ElectricEng/Exploiting_BluetoothLE_final.pdf [accessed 8 Apr 2019].