An SMS-based Pet Tracking System

by

Chenwei Zhang
B.Sc., University of Victoria, 2019

A Project Report Submitted in Partial fulfillment of the
of the Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

An SMS-based Pet Tracking System

by

Chenwei Zhang
B.Sc., University of Victoria, 2019

Supervisory Committee

Dr. Xiaodai Dong, Department of Electrical and Computer Engineering, University of Victoria
(**Supervisor**)

Dr. Levi Smith, Department of Electrical and Computer Engineering, University of Victoria
(**Departmental Member**)

ABSTRACT

In this report, we propose an Internet of Things (IOT) based pet tracking solution which relies on the SMS service for communication. We introduce our proposed security solution that addresses the SMS communication in the pet tracking system and provide details for the implementation of the system on both the Android platform and the embedded hardware platform based on the defined requirements. The overall design realizes a flexible, low power, low cost, secure and privacy protected wireless pet tracking system.

Keywords: IoT (Internet of Things), pet tracking, SMS security

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Dr. Dong, who guided me throughout this project. Without her help, patience and support, I could not have completed this project. I would also like to thank my family and friends who supported and encouraged me during my study.

Chenwei Zhang

Chapter 1

Introduction

Dogs and cats are considered the most popular pets which not only bring entertainment, but also emotional attachment to the owners and other household members. According to the Canadian Animal Health Institute (CAHI), the 2018 Canadian pet population survey showed that 58% of Canadian households report that they own at least a dog or cat. The number of pet ownership continues to increase year by year, in which the number of dog ownership continued to increase from 7.6 million to 7.7 million, while the number of cat ownership remained unchanged at 8 million [1].

As more and more people decide to own a pet, the number of lost pets increased. In spite of caution, the owners fail to protect their pets from disappearing or running away unintentionally. This is because sometimes the event is beyond the control of pet owners. For example, pets instrictinly react by running away when they perceive the place is dangerous. The American Humane Association (AHA) reportedly estimates that over one-third of pets become lost at some point of their lifetime in the US. Additionally, over 80% missing pets are never found [2]. From the owners' perspective, pets are not just an animal, and are considered as the part of the family which is irreplaceable. It is devastating for them to not be able to find their pets at the moment.

Though there are plenty of pet searching organizations in place already across North America, pet owners feel reluctant to count on those organizations. The primary reason is that many pet owners find that those non-profitable organizations provide disappointing services, while the profitable ones are too costly for searching for their missing pets. Therefore, technologies are introduced to assist pet owners in finding their pets from missing. Internet of things (IoT) technologies used to track Alzheimer and dementia patients, which now, however, have been re-purposed to

track the pets. An IoT-based pet tracking system typically relies on five hardware components: a GPS Antenna used to obtain the GPS location of the tracker, a GSM antenna and a GSM sim card used to receive the network signal and connect to the network, a processing board used to run the IoT application and a battery as the power supply for the tracker.

1.1 What is IoT?

Traditionally, internet has been used by people to share the data, enhance the communication and boost the connection among people since it was invented, which is so-called Internet of People, while the Internet of Things or IoT is a new paradigm which was poised to accomplish things that the traditional Internet has been doing among not only people, but also objects, such as watches, phones, cars or even buildings. According to paper [3], the IoT is an intelligent network which connects all information sensing devices to the internet for the purpose of information exchange and communication. In a nutshell, IoT describes the interconnected devices that collect, share and process data through the wireless network.

The Internet Architecture Board has defined four technical connectivity models that are commonly used by IoT implementations, namely Device-to-Device, Device-to-Cloud, Device-to-Gateway, and Backend Data Sharing [4]. The Device-to-Device communication model is the simplest, which allows two or more interconnected devices to directly communicate with one another over the network. The model is widely used among wearable IoT devices. For example, Apple users are allowed to lock and unlock their iPhone through the paired Apple watch. Device-to-Cloud refers to one or more IoT devices directly connecting to a cloud service in which each individual device only exchanges data with the cloud. This model allows devices to download the software updates from the cloud remotely. Device-to-Gateway involves at least three parties in the single IoT system. In this model, an intermediary device is connected between an IoT device and a cloud. Gateway devices can bridge the interoperability gap between devices that communicate on different standards. Backend Data Sharing is an extended version of the Device-to-Cloud model. Under this model, users can export and combine the smart object data from a cloud database service with the data from other sources for analysis. A complicated IoT application system normally uses more than one above communication model. In this project, our system primarily relies on Device-to-Device model, although many pet tracking systems have cloud service

involved. In our opinion, the location data of a pet contains private information, which can be correlated to the owner's location. Therefore, we decided not to store the location in a remote cloud to ensure the data stays local and only accessible by the Android application.

1.2 Related Work and Motivation

First of all, we focused on the literature review, there have been numerous research on IoT-based pet tracking systems. In paper [5], the authors proposed a system that allows pet owners to track their pets in an indoor environment. The proposed system used a mote with an environment sensor board which is attached to the pet to track a pet's location, activity and surrounding environment. Additionally, the system required other multiple motes to be installed in different areas of the house to keep tracking the surrounding environmental conditions. A stationary mote which collects sensor data from other motes forward all data to an application running on PC over the wireless network. Regarding the technology in the system, they used Processor Radio Board as motes which supports TinyOS software. Also, the PC application was programmed in Java which can analyze and visualize data that came from motes.

Bokk Meow [6] is another IoT-based pet tracking system which was proposed by a software development team from Thailand. The system actually provided a platform for all pet owners who use Bokk Meow to assist each other in finding the missing pets. The proposed system comprised three actors: pet owners, pets with the tracking device and system administrators. Pet owners have to use a mobile application to track their pets. They are also able to upload photos of the missing pet and make a post on the mobile app to get help from and communicate with other users. The system administrators are responsible to maintain all the information that is posted on the mobile app. Finally, they used NodeMCU as the tracking device which is capable of receiving GPS and transmitting data over WiFi. The system uses a Device-to-Cloud communication model in which a firebase served as a intermediary object in the system to establish the data transmission between the tracker and the mobile app. Specifically, the tracking device uploads GPS location data to the firebase, then the mobile application retrieves the location from the firebase and plots on the google maps.

Furthermore, we have also studied some existing popular products in the market. However, we discovered that over 90% of products require customers to not only buy

their pet tracker devices, but also additionally charge the users a monthly or annually subscription fee to continuously use their GPS service. Although some products are subscription-free, these products either support a very limited tracking range, such as Smart Compass [7] which allows pet owners to receive their pet's GPS location within only three miles, or are costly. Garmin which costs $699 USD provides a wider GPS tracking range but still limited within nine miles [8]. Some products, such as Whistle [8], only cover connection to AT&T cellular network for the GPS location transmission, which means they can only be used across the US.

Finally, most pet tracking systems use data for Device-to-Device communication, which is not friendly to pet owners who use prepaid cellular services. These people are given unlimited SMS text in the monthly base plan, but they have to pay extra money to buy the data plan for the tracker usage. In fact, according to Cansumer [9], Canada, in fact, reportedly has the most expensive wireless data cost in the world [10]. A study published by The Markup compared how much it cost to load an hour of Netflix among 28 countries. In Canada, it costs $12.55 USD, which is the most expensive one, as opposed to Italy which is the cheapest at $0.47 USD. Therefore, in our opinions, using SMS for GPS location transmission could also potentially increase our user base to those who are reluctant to purchase expensive data plans in Canada in the future.

Therefore, we proposed an IoT-based pet tracking system which can support not only unlimited range of GPS tracking, and can transmit the GPS location data to the user over the SMS services across Canada.

1.3 System Requirements

The first step to design the system is to identify the requirements and functionality of the system. We introduced two ends into the system namely an IoT tracking device which support real-time GPS acquisition as well as SMS transmission, and a mobile application which provides pet owners a graphical interface to interact with the tracker.

Since the system involves two ends, we identified that a pairing process is required. The communication between two ends is established only after the pairing is successfully set up. Under the communication, the smart phone exchanges message with the tracker to start and terminate the GPS information acquisition. Specifically, In order to clearly identify the functional requirements of the system, we first started with a

use case scenario as follows:

1. When the pet is out of the owner's sight, the owner uses the pet tracking app on his smart phone to communicate with the tracking device.
2. The owner first has to login using email address and password in order to gain access to use the app.
3. The owner sets for how many minutes, they want to receive the GPS location.
4. The owner taps the "START" button in order to initiate GPS acquisition.
5. The app sends a GPS acquisition message to the tracker over SMS to trigger GPS acquisition service.
6. The tracker successfully obtains the real-time GPS locations, starts transmitting them to the smart phone over SMS for the time the user set.
7. Once the phone received the GPS locations, the app plots them on a google map so the owner is able to identify the whereabouts of the pet.
8. The owner finds the pet, tap "STOP" button to terminate the tracker acquiring and sending GPS location.
9. The owner has not found the pet, then goes back to step 3.

In addition, we had to consider the extension scenario which describes how the system should respond when things do not go right in some of the above steps, as follows:

2a. If either email or password is wrong, the mobile app displays a feedback message to the owner to indicate that.

4a. The phone fails to send out the message, the app displays an error message to the owner.

4b. The tracker fails to receive the message, the app displays an error message to the owner after a time out.

4c. The tracker fails to start the GPS service, then sends an error message back to the phone to indicate the user.

5a. The tracker fails to obtain the GPS location, then sends an error message back to phone to indicate the user.

Devices	Requirements
The tracker	• it should only respond to the paired smart phone. • it should start GPS service when the "START" message is received from the user. • it should transmit the GPS location for the time the owner requested once the location is acquired • it should stop GPS transmission once the "STOP" message is received from the user.
The mobile app	• it should provide an authentication mechanism for app access. • it should provide users buttons to control the tracking device to start or terminate GPS acquisition and transmission over SMS. • it should enable users to tell the tracking device how long they want the GPS location to be transmitted. • it should display the received GPS location on a Google map which enables users to know where the pets are.

Table 1.1: System functional requirements.

6a. The app fails to extract the GPS location from the SMS message, the app displays an error message to the owner.

Overall, the system has the following requirements, shown in Table 1.1, based on the use case scenario:

In addition, since the communication in our system relies on SMS, security is overwhelmingly important to our system. This is because messages that are sent through wireless communication over SMS are unencryted. In fact, mobile phone networks around the world are connected to each other through the Signaling System No 7 (SS7) protocol [11]. The SS7 system has been repeatedly attacked by hackers who have snooped on SMS messages or intercepted them [12]. Another consideration is that not just the hackers have been trying to compromise SMS messages, but

also the governments around the world. A stingray is a controversial surveillance technology that is in governments' spy kits. Stingrays impersonate a legitimate cell phone tower in order to capture mobile devices' or other wireless devices' data that are connecting to them within a limited physical range. In addition, SMS messages can also be swept up in larger surveillance systems. According to the documents released by Edward Snowden back in 2014, the NSA was, at the time, collecting over 200 million text messages a day from around the globe [13]. Therefore, to secure our system, we needed to add an encryption method into our system to make sure any message that is transmitted between two parties is encrypted if the message is intercepted by the attackers during the transmission, they are unable to read the message. This is covered in Chapter 2.

1.4 System Design

Once the system requirements were identified, in order to map the theory to the reality, we first focused on designing the system architecture which satisfies the functionality of the proposed system in the previous section. As we defined, the system is composed of a pairing process and an interaction process, so we drafted the architecture for the pairing process from a process view [14].

For the first time pairing, the user has to register an account using email and password. The IMEI of the mobile phone is also required in the first time pairing which is used in the future to identify whether the mobile phone is legitimate. The email and password are stored in the database on the cloud while the IMEI is stored in the database of the tracking device. For the signup flow, once the user fills out and tap on the submit botton, the app first sends "SIGNUP+IMEI" to the tracker. Once the message is received by the tracker, it stores the cellular number of the message and IMEI contained in that message in its database, then sends "CODE: 20002" back to the mobile phone if neither cellular number nor IMEI exist in the database. Otherwise, the tracker sends back "CODE: 40001" to the mobile. When the mobile application receives "CODE: 20002" from the tracker, it eventually submits the user's email and password to the cloud database. When the first time pairing is successful, the user is required to provide email address and password for future pairing which are submitted to the cloud database for verification first. Once the credential is verified by the cloud database, the app sends "LOGIN+IMEI" to the tracker. The tracker will check whether the cellular number and IMEI exist or not, it will send "CODE:

20000" to indicate they are found in the database or "CODE: 40004" for not found. It is worthwhile to also note that the communication between the mobile application and the cloud database is over the network instead of SMS. Fig 1.1 shows the overview of the first-time pairing process.

Once the pairing is setup, the system enables the user to interact with the tracking device through the mobile app. To start acquiring the GPS location from the tracker, the app sends "START+X" where X is a placeholder for how many minutes the user would like to keep receiving the GPS, while to terminate the GPS acquisition and transmission, the app sends "STOP" to the tracker.

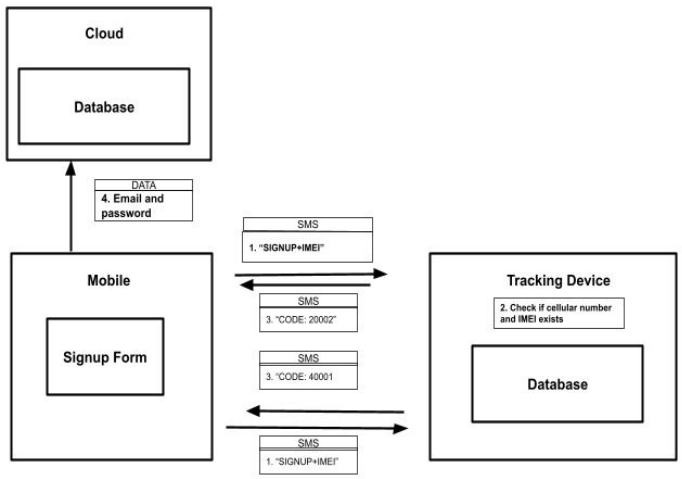

Figure 1.1: First-time pairing process

From the development view [14], we identified major components of the proposed system which can get the ball rolling as illustrated in Fig 1.2. On the mobile side, the SMS Receiver and SMS Sender components are responsible for managing incoming messages from the tracker and outgoing messages to the tracker. The GPS Extractor extracts the GPS location from the received message and then passes the location to the Map component to plot it on a digital map. The Cloud Database API allows the application to connect to the cloud database to submit and verify user credentials. Similarly, the SMS Receiver and SMS Sender components are also needed on the tracker side. In addition, the SMS Processor component interprets the incoming messages from the mobile. If it is a pairing message, the cell number and IMEI are passed to the Database component. The GPS Service component manages starting and terminating GPS acquisition.

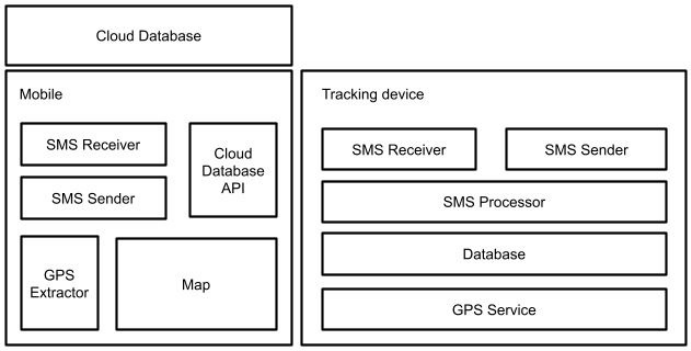

Figure 1.2: Architecture of the system

1.5 Project Contributions

The idea of the project was initially proposed by Dr. Xiaodai Dong. Prior to the development, Dr. Dong and Yiming Huo concentrated on the IoT tracking device selection. I participated in system design and requirement analysis, especially for the Android application, with both of them. Then, I primarily focused on the development on the software and firmware alongside the system requirements that we created. Dr. Dong helped us resolve any technical issue regarding the IoT communication and supervised us throughout the project to ensure that the implementation functionally meets the requirements.

Chapter 2

System Security

A single authentication mechanism is not enough to secure the system. As explained in the system requirements section, the SMS messages which are transmitted around the world are unencrypted so not only the cellular service provider can read the message without any efforts, but also malicious attackers as long as they compromise the cellular tower. In this chapter, we review some literature on how they secure the SMS communication, then introduce some famous encryption methods in both symmetric and asymmetric cryptography. Finally, we describe the encryption we chose that can address the SMS security issue in our system.

2.1 Related work of Securing SMS

Securing the SMS message is not a new-fashioned topic. Many researchers have attempted to used various encryption techniques to provide confidentiality to SMS transmitted messages.

In paper [15], the author developed a "Safe SMS" application which can secure the SMS messages that are transmitted between mobile phones. As the author described, the solution is an end-to-end encryption which provides confidentiality, integrity and authentication in the SMS communication without requiring any additional hardware tokens. Notably, to achieve authentication, the system adds a database which is used to store some keywords, such as Personal Identification Number (PIN) or phone numbers. The application checks whether the keyword exists in the database once the message is received.

In [16], Tarek M. Mahmoud proposed a solution using RSA to encrypt SMS over

Symbian OS. Since using RSA encryption method usually generates ciphertexts that are longer than 160 bytes which is the SMS character limit, he added a compression technique to shorten the SMS length within 160 characters. In general, when the system gets the message, it determines the SMS recipient, then compresses the message. If the compressed message is still longer than maximum limit, the system divides the message into segments. Subsequently, the system encrypts the compressed SMS using the RSA algorithm and add signature before the message is sent. The system provides both confidentiality and non-repudiation. However, the auther did not describe the compression technique in the papaer.

In [17], authors proposed a security solution for SMS in GSM Network in 2012. They compared the symmetric encryption techniques, such as Blowfish algorithm, DES, Triple-DES, AES with the Aymmetric cryptography algorithms, such as DSA and RSA. They conducted studies of these encryption technique against some common attacks on SMS such as man-in-the-middle (MITM) attack and replay attacks. They concluded in the paper that the AES algorithm is the most efficient technique to secure SMS over GSM network.

2.2 Symmetric Cryptography

Symmetric key encryption or private key encryption uses the same single key that is only known by the sender and the receiver to encrypt data. In private key encryption, both parties have to keep the key secret in order to protect the encrypted message against attackers. Here is a scenario how symmetric key encryption generally works: Bob is about to send his health information to Dr. Kathlee and he does not want anyone else to see it. Both Bob and Dr. Kathlee have the same private key stored in their own hard disk, so Bob uses his key to encrypt his message, and then the algorithm generates a string of scrambled data which is not readable by humans. When Dr. Kathlee receives the encrypted message, she uses the same key stored on her hard disk and decrypts the message, and now the scrambled message becomes back to human-readable texts. Symmetric algorithms generally have two categories: 1) Stream cipher 2) Block cipher.

A stream cipher typically operates on one bit of plaintext at a time. A single bit of the message is combined with a single bit from a pseudorandom cipher digit stream (a keystream) using XOR operation to produce a single bit of cipher. Stream ciphers are typically less secure than block ciphers and less used in the real world. One of

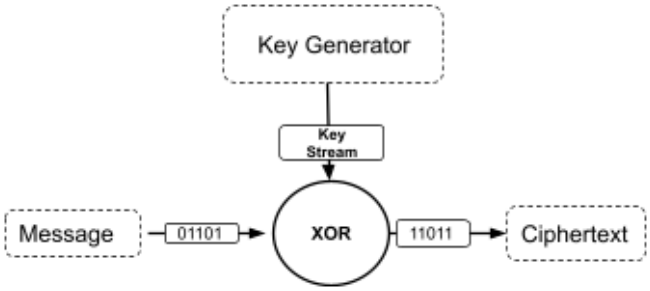

Figure 2.1: Stream cipher

the famous implementations of stream cipher is RC4 which was developed by Rivest-Shamir-Adleman (RSA) Data Security. Wired Equivalent Privacy (WEP), a security algorithm for IEEE 802.11 wireless communication, uses RC4 which is considered the least secure wireless communication protocol.

A Block cipher operates on "chunks" or blocks of a message. In addition, in block cipher, it accepts a key length parameter and produces a single fixed-length randomly chosen secret key. For example, assume that there is a plaintext "encrypt me", the block cipher would first break the plaintext into multiple equal size blocks with presumably 64 bits. Once all blocks are generated, the secret key is used to encrypt each block individually to generate the ciphertext. When the block size is 64 bits, then the ciphertext will also be 64 bits. Finally, the algorithm will concatenate each block of ciphertext by using a technique called cipher block chaining. The idea of cipher block chaining is to use ciphertext from the previous block to impact the next block. One of the most famous block cipher algorithms is Data Encryption Standard (DES). DES uses a 56 bits key to drive the encryption and decryption process, and it is a 64 bits block cipher that has five modes of chaining operations.

Electronic Code Book (ECB) is the least secure mode. In ECB mode, the given plaintext is divided into blocks of 64 bits and the algorithm encrypts each block independently. The ECB mode introduces patterns of ciphertext, which means given the plaintext whose occurrence is more than one in the input can generate the same ciphertext in the output, which can give clues to the attacker [24].

Cipher Block chaining (CBC) is a more popular mode and slightly more secure, as opposed to ECB. CBC computes the bitwise EXCLUSIVE OR(XOR) of every plaintext with the ciphertext from the previous block, which can introduce errors to the next encrypted block. For example, if one block is corrupted while encrypting, then the corrupted block/error will be chained to the next block for encryption,

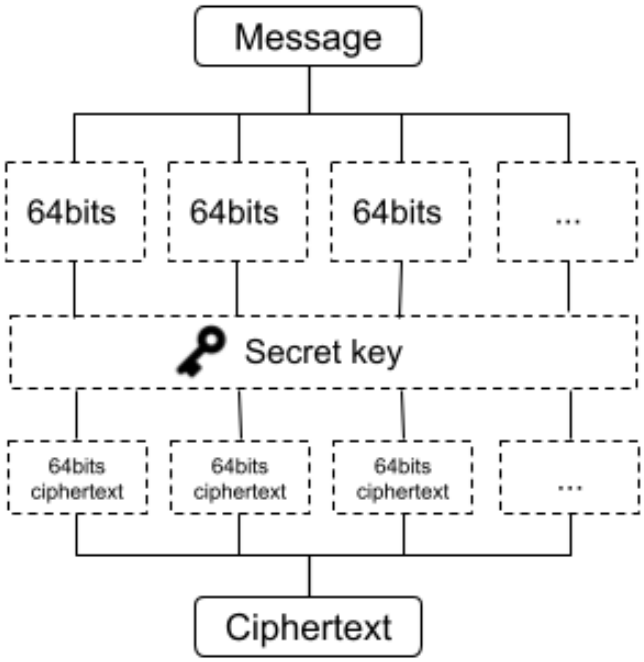

Figure 2.2: Block cipher

therefore the remaining blocks all have the error and it is impossible to decrypt them [18].

Cipher Feedback Mode (CFB) is a block cipher algorithm that emulates a stream cipher. The data in CFB mode is encrypted in smaller units of block 8 bits instead of predefined 64 bits. CFB uses an Initialization Vector (IV) also, and the ciphertext block of the previous block is encrypted first and then XOR with the current block. The use of IV eliminates CFB to generate any patterns the plaintext may have. Regarding encryption and decryption speed, CFB outran CBC. However, CFB also suffers from Error Propagation as CBC does [18].

Output Feedback Mode (OFB) operates in a similar fashion as CFB mode. The only difference is rather than XOR-ing the ciphertext of the previous block with the current block, a seed value is created and added to every XOR operation. As opposed to CBC and CFB, error propagation is eliminated in OFB because there is no block chaining [18].

Counter Mode (CTR) is considered to have the best balance of speed and security. The algorithm uses a counter that increments for each and every encryption process. The unique counter ensures that each cipher blocks XORs with a unique key stream value. Additionally, there is no chaining operation involved in CTR [18].

DES was first published by the US government in the 1970s, and had been widely embedded in thousands of products. However, DES was replaced by the Advanced Encryption Standard (AES) in December 2001 because of its relatively short 56-bit key size.

Advanced Encryption Standard Advanced Encryption Standard (AES) or Rijndael was introduced by the National Institute of Standards and Technology in October 2000 to replace DES. One year later, the National Institute of Standards and Technology (NIST) officially mandated to use AES to encrypt all sensitive but unclassified data [19]. As opposed to DES, AES uses subkeys operation which derive from the main key with the Rijindael key schedule [20]. Then the Addround operation is applied to data in which the data is XORed with the current round of the subkey. The key size will determine the number of rounds of the XOR of data and the subkey. The longer key size will add more rounds which brings more security. AES has three variants: AES-128, AES-192, AES-256. The digits indicate the key size of each AES variant. Different key length comes with different rounds of transformation on the messages, thereby adding different level of security for encryption. In a nutshell, the longer, the stronger. AES has never been cracked since it was published, and AES-256 is considered one of the most secure symmetric cryptography algorithms in the world.

In the block cipher family, besides DES and AES, there are also many other encryptions that are well-known. The Skipjack algorithm operates on 64-bit blocks of messages [21]. The key length for Skipjack is 80 bits; the algorithm also supports the same four modes of operation supported by DES [21]. However, Skipjack is rarely used for data encryption, it provides the cryptographic routines supporting the Clipper and Capstone encryption chips [21]. International Data Encryption Algorithm (IDEA) was invented to become the solution to the insufficient key length of the DES. Likewise, IDEA has a block size of 64 bits and 128 bits size of key length. IDEA now is widely used for secure email encryption. In addition, RC2 and RC5, which are patented by Rivest-Shamir-Adleman (RSA) Data Security, are no longer considered secure.

In symmetric-key cryptography, key distribution is a major challenge. Entities that share the same private key must have a secure method of exchanging the key before establishing the communication. It is recommended to use offline key distribution, if possible, in which one party provides the other a medium (i.e., Hard disk,

USB drive) that contains the secret key. The security of the symmetric encryption method is completely dependent on how well clients protect the key.

2.3 Asymmetric Cryptography

Asymmetric cryptography or public-key cryptography came out after symmetric encryption as a solution to the weaknesses of symmetric key cryptosystem. In an asymmetric cryptosystem, rather than sharing the same secret key, each client has a key pair: 1) a public key, 2) a private/secret key, which are mathematically generated. The public key is distributed to everyone in the system while the private key has to be protected by the system who creates the key pair. Here is a scenario: Bob would like to send his bank account information to Mary. Mary first has to generate a pair of keys: one public key and one private key. Mary keeps the private key in a secure place and gives the public key to Bob. Once Bob receives Mary's public key, he encrypts the message with Mary's public key, then sends the encrypted message back to Mary. When Mary receives the message from Bob, she uses the private key to decrypt the message.

Rivest-Shamir-Adleman RSA (Rivest-Shamir-Adleman) is one of the well-known asymmetric cryptography which is widely used for secure communication. RSA cryptography is the underlying integer factorization problem [22]. The scheme of RSA is as follows:

1. We pick two pairs of numbers (5, 14) and (11, 14), of which (5, 14) will be used for encryption and published to everyone, (11, 14) for decryption.

2. We select the single character '2' as the message we want to encrypt, and assume '2' can be represented numerically as 2

3. To encrypt the message, we compute the message 2 to the power of the first number, mod the second number:

$$2^5 mod 14 = 4,$$

   and 4 is our ciphertext produced by RSA.

4. To decrypt the message, we compute the ciphertext 4 to the power of 11, mod 14, get:

$$4^{11} mod 14 = 2,$$

which equals our original message '2'.

In fact, these two pairs of numbers are not chosen by us. In order to generate the key pair in RSA, the algorithm first has to pick two prime numbers, p and q. Assuming p is 2 and q is 7, then the algorithm calculates the product N of p, q which is 14. So 14 is the modulus or second number is each key pair. The RSA algorithm introduced a $\phi(N)$ function, $\phi(N) = (p-1)(q-1)$. Then the algorithm has to choose a number ε which satisfies the following conditions [22]:

1. $1 < \varepsilon < \phi(N)$,
2. coprime with N, $\phi(N)$.

Given $\phi(N)$ is 6, the only number that has co-prime with 14 and 6 is 5, so ε is 5. So far, the public key is generated (5, 14). Subsequently, the algorithm chooses another number d, in which $d * \varepsilon(\text{mod}\phi(N)) = 1$. Any number that holds the equation can be chosen, so it can be 1 or 11. If we pick 11, we get the key pairs as we had in the example. Unlike what we did in the above example, in the real world, p and q are typically greater than 2^{512}. There are no published methods to crack the RSA algorithm when a large enough key is used [23].

Diffie-Hellman Another famous asymmetric key cryptography is the Diffie-Hellman algorithm. The Diffie-Hellman algorithm is widely used as symmetric-key exchange algorithm since it was developed in 1976. The algorithm is to enable two clients to exchange a symmetric key, in a secure way, that can be used for the message encryption . The algorithm itself has never been used for transmitting secret messages.

Diifie-Hellman key exchange algorithm works in a similar mechanism as RSA key generation. However, rather than only one party generating the key, in Diffie-Hellman, two parties are involved in key generation. First, the algorithm selects a prime number q and α which is the primitive root of q. Then client A generate X_A where $X_A < q$, client A calculate public Y_A,where $Y_A = \alpha^{X_A} \bmod q$. Subsequently, client B does the same thing, selecting $X_B$where $X_B < q$, generating $Y_B = \alpha^{X_B} \bmod q$. Afterwards, client A and client B exchange Y_A and Y_B with each other to derive the symmetric secret key separately, where for A, $Key = Y_B^{X_A} \bmod q$, for B, $Key = Y_A^{X_B} \bmod q$.

However, asymmetric cryptography has a major problem. Look at the Bob and Mary scenario again, although Mary's private key can verify the message is secure when it is in transit, it cannot verify who the sender actually is. Since Mary's public key is public to anyone, anyone is able to get her public key, and send the encrypted message to Mary. Imagining Bob and Mary discussing the date of the business appointment, the hacker grabs Mary's public key and sends a wrong date to Mary before Bob does so, Mary can miss the appointment with the real Bob. Even in the Diffie-Hellman key exchange algorithm, imagining, without a trusted third party involved for identity verification, the hacker can easily pretend to be another party in the communication, and exchange the symmetric key with the legitimate user. In addition, the asymmetric encryption algorithm is relatively slow as opposed to symmetric cryptosystems.

2.4 Security Solution

For our project, we first conducted some experiments to use the public key cryptography to provide an end-to-end encryption. Ideally, we would like to use phone numbers as the public key for encryption. In fact, we found that, as mentioned in the related work section, RSA has a minimum of 2048-bit key length [24], but the character limit for a single SMS message is 1280-bit, which means in order to exchange public keys initially, both ends have to transmit one single public key in at least two separate SMS messages, which could be troublesome especially when the SMS are even received in the wrong order. Although we can apply the message compression technique as the authors of paper [16] did, we are unable to guarantee the encrypted messages are limited within 160 characters. Moreover, cell phone numbers can be counterfeited. If the hacker fakes the tracker's cell number and sends the user wrong GPS coordinates of their pet. The user would be unable to figure out whether the location was sent from the legitimate tracker or hacker, the lost pet would never be found. Finally, we conducted experiments to run RSA encryption and decryption on the selected tracking device particularly, the results showed that RSA encryption and decryption is relatively slower than any types of symmetric encryption methods. The following table shows the comparion of the time of running symmetric and asymmetric encryption on MangOH in milliseconds.

Therefore, we proposed to use AES for end-to-end encryption based on our experiments and conclusion from paper [17]. Unlike the traditional key sharing in sym-

Cryptography method	Encryption	Decryption
RSA	5362	4872
AES	3646	3217

Table 2.1: Cryptography performance on Legato

metric cryptography systems, our security system uses an offline secret key sharing method rather than distributing the key over the network. Regarding key generation, we decided to use tracker's IMEI as the secret key instead of generating a random one programmatically because, firstly, IMEI is unique enough to become the secret key, secondly, IMEI is only accessible by login the MangOH board via USB and the board is protected by Secure Shell Protocol against unauthorized access. Drawing on the approach in the literature, we combine the AES encryption with the proposed authentication mechanism in Chapter 1 to provide both confidentiality and authentication. In a nutshell, the sender sends the AES-encrypted message, the recipient decrypts the message, then uses the phone number of the sender as an personal identifier and checks whether the number exists in the database to determine the phone number has paired with the tracker or not. Clients who purchase the tracker will be given a hardcopy of the symmetric key, and the Android application requires them to manually enter the key when they sign up on the app. On the tracker side, the device programmatically fetches its IMEI and convert it to the key. As long as the user keeps the key secret, the security is guaranteed. By this manner, we ensure that both confidentiality and authenticity are in place. Specifically, End-to-End encryption provides confidentiality protecting messages transmitted between two platforms against Man-In-The-Middle(MITM) attacks. For authenticity, exchanging the secret key offline guarantees that the key is only known by the legitimate users and the legitimate encrypted message can be only sent from these users. Finally, we designed the secured pairing process and authentication process between MangOH and Android app, based on the end-to-end encryption and authentication mechanism that we proposed previously, as shown in Fig 2.3 and Fig 2.4

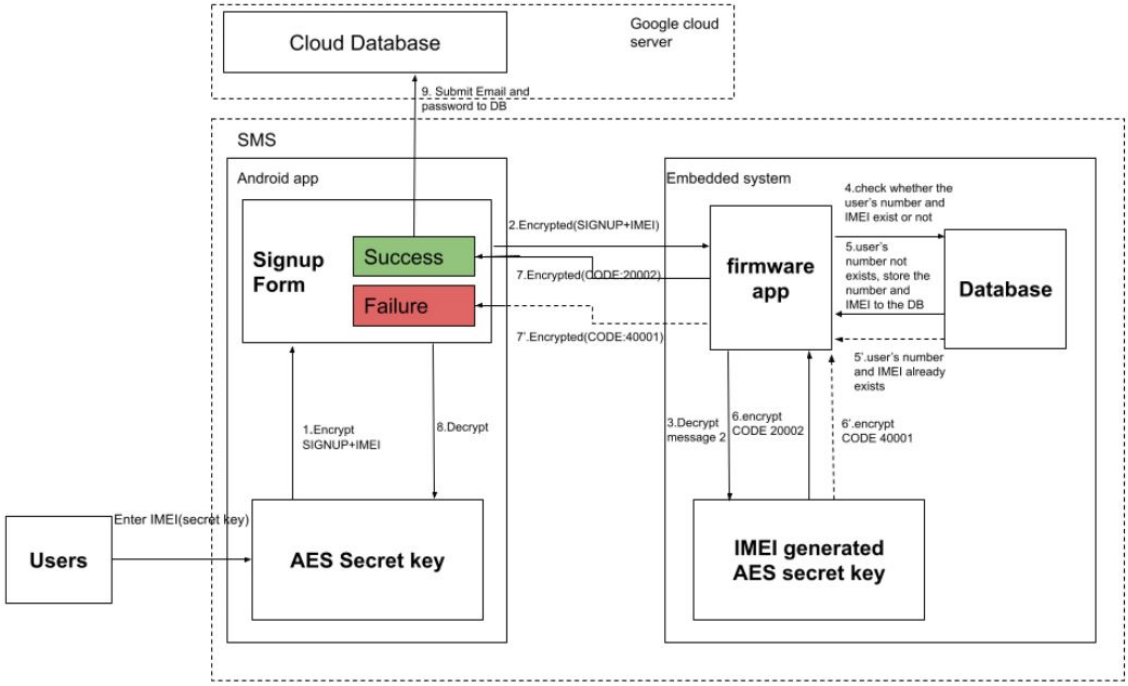

Figure 2.3: First-pairing/Signup process with security solution

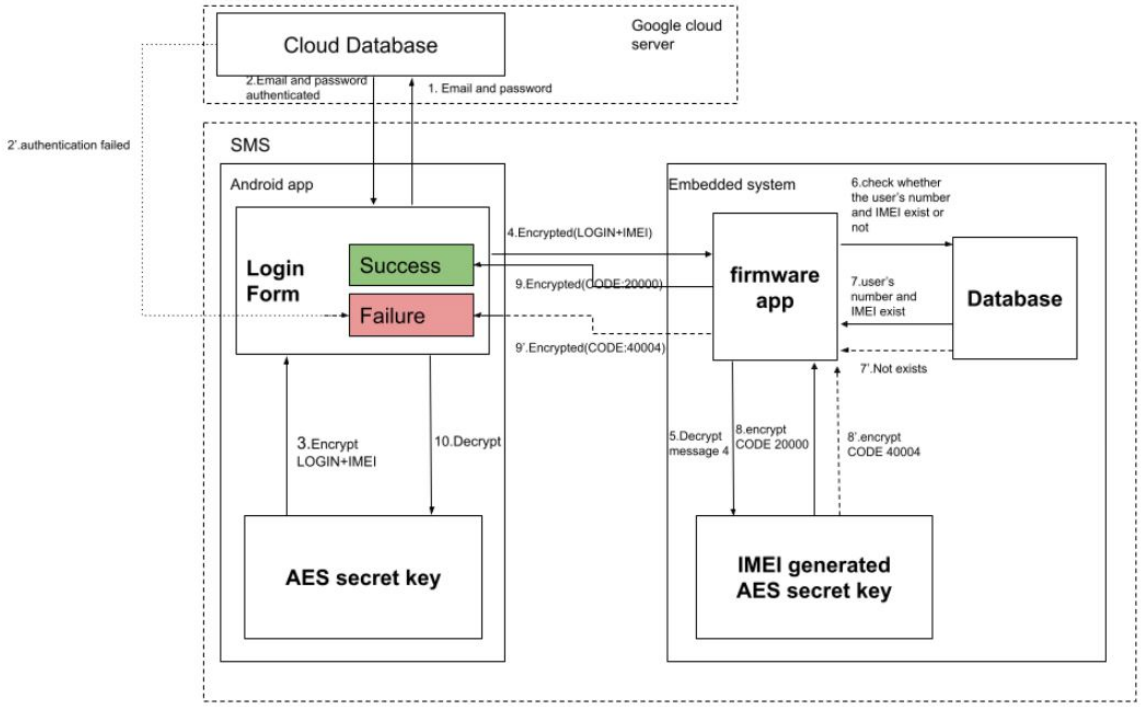

Figure 2.4: Login/Authentication process with security solution

Chapter 3

Android Application

Since the system is composed of two platforms, Android application and tracker platform, they will be discussed individually in two separate chapters. In this chapter, we introduce the architecture and main components of the Android system, and we describe the implementation of the application.

There are primarily two platforms for building mobile apps, namely, Android and iOS. However, we decided to focus on developing an app for one platform first rather than building the app for iOS and Android simultaneously, because cross-platform application development is time-consuming and requires a lot of effort to manage the code base for a small development team. Hence, when considering which platform we would build our app on, we compared the following differences. Firstly, to build an iOS application, developers are required to install an Integrated Development Environment (IDE) called XCode. XCode is the macOS-only software program, which is not friendly to Windows users. Android application development relies on Android Studio, which is capable of operating on both Windows and MacOS. Additionally, regarding programming languages, Android apps are written in Java, while iOS developers use Swift to build apps on Apple products. As we have experience in building softwares with Java, building apps in Android would require a less steep learning curve for the team. Lastly, we found that developers have to pay Apple $99 USD annually for deploying the app onto the iOS devices or distributing them in the Apple store, while Android developers are required to pay $25 USD just once for publishing the app onto Google play. Therefore, we decided to choose Android for building the mobile app for the present, but we will build an iOS version in the future when the iOS user base grows.

3.1 Android System Architecture

Android is an operating system created by Google which is initially designed for touchscreen mobile devices [25].

Android has an architecture which was designed and segmented into layers. Fig 3.1 shows that four layers compose the Android architecture. Android architecture is a complete stack that includes everything from the operating system through middleware and up to applications. The architecture is built on the linux 2.6 kernel. The linux kernel is used as the hardware abstraction layer. In other words, for an Original Equipment Manufacturer (OEM) who is trying to build Android on a new device, the first thing has to be done is bring up Linux and get all drivers in place. The Linux kernel provides memory management, process management, a security model, networking, and a lot of core operating system infrastructures that are robust and have been proven over time. On the top of the kernel is the Libraries layer. It is

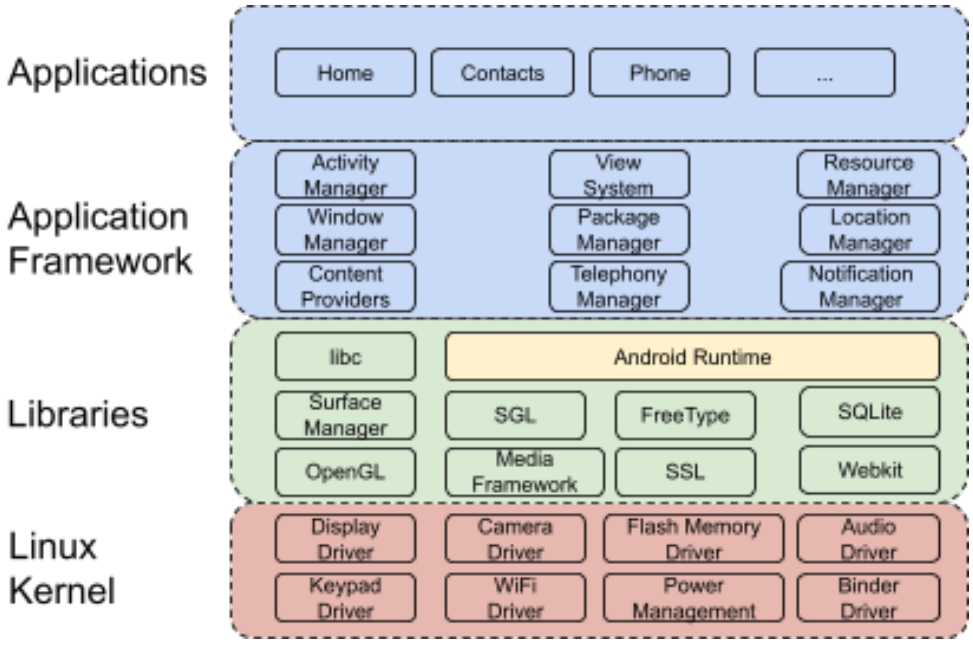

Figure 3.1: Android architecture

written in C/C++. The layer provides a lot of core power of the Android system. The native libraries include nine components. The surface manager is responsible for combining different drawing surfaces onto the screen. OPENGLES and SGL make up the core of the android graphic libraries. OPENGLES is a 3D library. Android has a software implementation that is hardware accelerate-able if there is a 3D chip on

the device. The SGL graphics are for 2D graphics, which is commonly used for most of the application drawing. Android allows the combination of 3D and 2D graphics in the same application. Media framework was provided by PacketVideo, which is one of the members of Open Handset Alliance that contains all of the codecs, such as MP3, JPG, PNG, that make up the core of media experience. Freetype is designed for rendering the font. SQLite is introduced as the core of the data storage on Android. Webkit is the open source browser engine which is used as the core of Chrome. The Android runtime is an application runtime environment used by the Android operating system [26]. The Android runtime allows the Android operating system to run in an embedded environment which comes with limited battery, limited memory, limited CPU.

The third layer from the bottom is the Android Application Framework layer. As opposed to the Libraries layer, the framework layer is all written in the Java programming language. The framework provides the toolkits that all applications would use. The Activity Manager is responsible for managing the lifecycle of the applications. It also maintains the common back stack of the application. The Package manager is responsible for keeping track of which applications have been installed on a smartphone device. The Window Manager is responsible for window management. It is mostly a Java programming language abstraction built on the top of services in the lower level that are provided by Surface Manager that is in the Libraries layer.

The Telephony Manager contains APIs that we would use to build phone applications, for example, in our Pet Tracker app we want to use SMS transmission in the application, so the Telephony Manager provides APIs such as SmsManager, which allows us to send and receive SMS messages specifically in our application. The Content Providers is one of the four main components in the Android application that is introduced thoroughly in section 3.2. In a nutshell, it allows applications to share data with other applications. The Resource Manager is used for the storage of localized string, bitmaps, layout file description, all of the external parts of the application. The View System contains all the front-end widgets such as buttons, textView, radio buttons, text inputs so on and so forth. It is also responsible for layout dispatching, layout, drawing. The Location Manager handles GNSS connection with the satellite and APIs that allow users to use the GPS positioning service of the device. Finally, the Notification Manager provides notification services.

Lastly, the Application layer on the top is responsible for displaying the application to Android users and passing user's interaction with the application down to the

bottom layers. It contains default applications such as SMS inbox, contacts, browser, as well as users applications. Everything in this layer uses the same application framework provided by layers below.

3.2 Android Application Components

An Android application is empowered by components. These components are managed by the application manifest file AndroidManifest.xml which describes what component the application has used and how the components interact. There are four main components in the Android application, as follows [39]:

- **Activity** An Activity [39] is essentially a piece of the Graphical User Interface (GUI), corresponding to one screen. If we think of a mail application, it can be decomposed to three major Activities: something lists the mail, something displays an individual mail message and a screen to put together an outgoing mail. In addition, an activity can be started by a different app if the permission is given, although it belongs to one application. For example, a chat app can start the activity in an album app which lists all pictures in the phone to allow users to upload and send a picture in the chat. The activity could also enhance user experience while using the Android phone by doing the followings [39]:

  1. tracking which activity the user is current at so enables the system to keep holding the process that contains that activity.
  2. remembering the state of the stopped process so when the user gets the process back on, the activity can restore the state of the previous process. For example, reading an e-book on the phone, the user restarts the e-book app next time, the phone displays the page where the user left at.

  In essence, the Android system defines a lifecycle of the activity. As the user navigates through different activities in the app, the activity transitions through different states in its lifecycle and each state has been abstracted to six callback functions namely `onCreate()`, `onStart()`, `onResume()`, `onPause`, `onStop()`, and `onDestroy()`. The system invokes each of these callbacks as an activity enters a new state.

  As the user launches an activity, three methods are invoked successively, `onCreate()`, `onStart()` and `onResume()`. In the `onCreate()` method, the system first checks

whether there is a saved state instance for the activity. If there is a saved state instance of the activity, the system loads everything from the saved state while initializing the activity and displays what it is supposed to be to users. Otherwise, the system just creates a new instance of the activity from scratch such as initializing variables that will be used, layout widget that need to be displayed. Subsequently, the `onStart()` method is called. The method is responsible for rendering everything that is defined in the layout file on the screen, which means the activity is visible to users on the screen after `onStart()`. Then the `onResume()` enables activity to start interacting with the user and the activity is at the top of the back stack. What follows is `onPause()`, the method is called to indicate that the user is leaving the activity and the activity is no longer on the foreground. Once the activity is no longer visible to the user, it means that it enters the Stop state thus `onStop()` is called. From the Stop state, the activity would either come back to interact with the user or the activity finishes running and goes away. If the activity comes back, the system invokes `onRestart()`. Activity might be destroyed due to lack of usable memory, which means it will be killed by the system or manually terminated by the user. It is worth noting that all the interaction that we have with the application happens between `onResume()` and `onPause()`. Fig 3.2 presents a visual representation of the activity lifecycle.

- **Service** As opposed to the activity components, the service component [39] runs in the background in the Android system and handles processes that run in the background. The component ensures the background processes to keep performing the proper function to users and not be killed by the system. For example, the user may listening to music in the background while using a different app. In fact, the service component manages the system to play the music until the task is terminated by the user or the system as necessary to resource reclaim. Also, the service component might fetch the data over the network while the user is on a different app. For example, we can receive messages from a chat app in the background.
- **Broadcast receiver** A broadcast receiver component [39] is an event listener that responds to system-wide broadcast announcements. Android system has defined a set of system-wide event, such as when the charger is connected, new SMS message is received etc. In Android, the broadcast receiver allows appli-

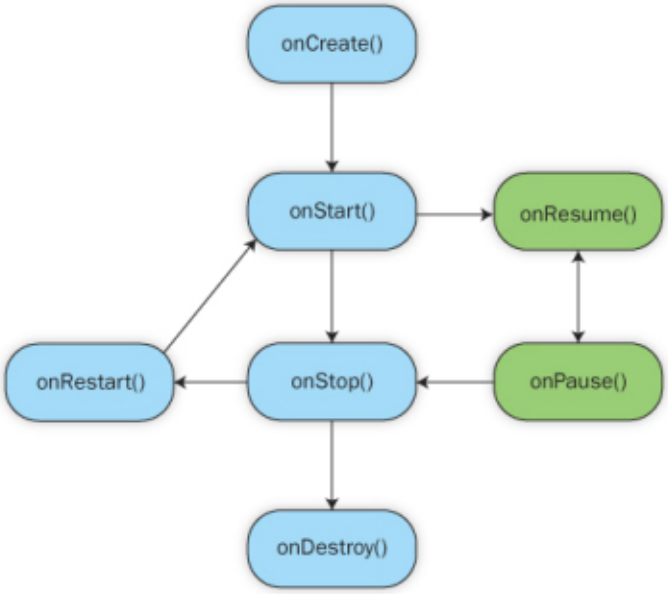

Figure 3.2: Android activity lifecycle

cations to perform a specific task when a certain system-wide event occurs. For example, a game application can automatically set the graphical performance to low when the low power notification is received thereby saving more powers, then set the graphical performance back to high when the system notifies that the charger is connected.

- **Content provider** The Content provider component [39] serves the purpose of a relational database. The component manages a centralized repository of data in the Android filesystem, a SQLite database, on the network or any other form of persistent storage location that any app can access or modify the data if the permission is given through the content provider component. For example, there is a content provider which manages the contact information in the Android system. Any app can query the content provider to read and write any data on the contact list as long as the permissions are granted. The Android system defined four fundamental operations in Content provider namely create, read, update and delete. Similar to database operations, the create operation creates or adds data in a content provider. The Read operation is used to fetch data from a content provider. The update operation modified existing data in a content provider. The delete removes existing data from the storage. Another key concept of content provider is the content URI(Unified Resource Identifier).

A URI is used to query a content provider to get the required data.

3.3 Android Application Lifecycle

In Android, every application has a lifecycle. The Android operating system is responsible for managing applications alongside the application lifecycle to start as Android users intend to, or terminate applications, as necessary to reclaim resources.

The following scenario illustrates the concept of the Android application lifecycle management by Android system from users' perspective. The user is currently at the home screen of a smart phone, and the user wants to navigate from the home screen to the chat application that is installed on the phone. Subsequently, the user would like to select a chat with someone in the application, and from the chat, the user clicks on a website link that is sent by that person to open up the website in the browser application. Finally, in the browser application, the user clicks on some address which starts the Google map application. The sequence of action actually taken through four different applications which are running in four different processes that are possibly written by many different developers. From the user's point of view, the transition from one application to another is supposed to be seamless. When users use the phone, they navigate forward or backward without noticing any system operations. Moreover, smartphone users never worry about what application takes how much memory, or how much memory we have consumed, how much memory is left on the device; nor whether our device is able to launch the new process because all resources have been taken up, because all of these are Android system's job.

With the above scenario, the Android system works as follows: at the beginning, the user is on the home page. Meanwhile, there is always a system process running in the background, which contains the Activity Manager and a back stack that is used to keep track of activities. Prior to starting the chat application, the system saves the state of the home application, then creates the chat process and starts the chat user list activity into that process. Once the user taps on a particular chat user displayed on the user list, a request is created to launch the next activity that goes into the system process. The system saves the state of the last activity before the new activity launches. It is worthwhile noting that "save the state" does not mean save every entity on the user list, it is simply the information about the current state of the activity such as where the user scrolled to in the list, and which item the user selected. Once the state of information is saved, the system is able to launch the next

activity that shows a particular chat history with a particular user. From the chat history message screen, the user clicks on a link to go out to the web which creates another request and saves the current running activity saved in the system process. Subsequently, a process is created to run the browser and launch the browser activity into that process, and now the browser is currently displayed to the user on the screen. Afterwards, the user clicks on the address in the website which launched a Google map. However, the system discovers that it is out of memory to run the Google map. So the system has to find a process to kill for running the map application. Typically, the home application is not going to be killed, because it is used for a navigation hub; nor the browser application, because that is the activity where the user just came from. Therefore, assume the system kills the chat application. Once the chat application is gone, the system immediately creates the maps process and creates a map activity in the process. And the Google map is currently displayed on the screen. The reason we want to save a little piece of state information in the system process is to improve user experience, so that users can go backwards as seamlessly as they navigated forward.

Once the user hits the "back" button from the current state, the system will remove the map application from the activity stack and restore the browser from its saved state. Now the browser is displayed to user on the screen and it is the foreground activity. If the user hits "back" again, the system is supposed to display the chat message view where the user came from. Unfortunately, the chat process has already gone, so the first thing the system would do is to make space to run the chat application by killing the map process, and then starting the chat process. Android launches a fresh copy of the message activity which does not include any stored state. Subsequently, the system fetches the saved information from the system process to restore the state of the activity in order to display the previous chat message to the user, meanwhile, remove the browser application from the activity stack. If the user hits the "back" button to go back to the user list screen, the system does not need to create a new process because it is still running in the chat process. At this time, the system simply creates a new instance of the user list activity and take the saved state from the system process to restore the view of where the user left. This is the Android application lifecycle in which the Android framework does the hard work of launching and shutting down the processes to manage recourse, making sure the state is preserved as users expect, therefore, giving users a seamless user experience when they switch between applications from the back and forth.

3.4 Android Fragment Lifecycle

As opposed to the Activity component in Android, a Fragment represents a portion of one screen [40]. A Fragment has to be created as a part of an activity, and an Activity can contains as many fragments as needed. Fragments can not exists independently, they are hosted by an Activity. Fragments are introduced when the developer wants to divide a single screen to multiple chunks. Additionally, Fragments can also be used when the app needs to navigate through multiple screens in a single activity. Although this is can be achieved with the Android Activity component as we introduced in the previous section, transitioning between Fragments in fact is more seamless and require less system resource because Fragments are managed by their parent Activity, while Activities are managed by the Android system. Considering a telephone app that contains a couple of tabs at the bottom navigation namely Keypad, Contact and Recent. When tapping on a different tab, a different UI layout is displayed on the screen. For example, the Keypad tab shows a miniature keyboard of digits which allows the user to dial phone numbers; the Contact tab lists the names and phone numbers of people; the Recent tab displays a list of call history that the user has made. In this scenario, the telephone app has a single Activity which contains three Fragments. The host Activity manages to show or hide which Fragment based on which tab is selected.

Similarly, Fragments in Android has its own lifecyle. However, because the Activity is the container of Fragments, when the Activity is destroyed, there is no reason that Fragments contained in the Activity still stay active in the system. So the Fragment lifecycle is directly affected by the lifecycle of the Activity which contains it. Each state of Fragment's lifecyle associates with callback functions, and when the Activity transitions from one state to another, it triggers callbacks for Fragments, as follows [27]

- When the Activity is created, Fragments are added and its layout is inflated. The `onAttach()`, `onCreate()`, `onCreateView()` and `onActivityCreated()` callbacks are triggered for Fragments.
- When the Activity is started, Fragments becomes active and visible on the screen. The `onStart()` callback is called.
- When the Activity is resumed, Fragments are back on the screen and ready for user interaction. The `onResume()` callback is triggered.

- When the Activity is paused, Fragments are paused which triggers `onPause()`.
- When the Activity is stopped, Fragments are stopped and invisible to users. Meanwhile, the `onStopped()` function is called.
- When the Activity is destroyed, Fragments are destroyed and `onDestroyed` is called.

3.5 Implementation

We first focused on mapping the defined architecture components to the Android components. We decided to create three activities in total, namely a signup Activity, a login Activity and a main Activity. Particularly, the signup Activity contains four fragments as illustrated in Fig 3.4, each of which provides an individual screen for users to enter signup informations. The first fragment provides the user text fields to enter emails and password. Then the next requires user to enter the tracker's cellular number. In addition to our designed architecture in Fig 1.2, the third fragment enables users to enter the secret key for AES encryption and decryption. Last but not least, the last fragment indicates the user the signup is successfully completed. It is worthwhile noting that when the user signs up successfully, the secret key will be saved persistently through the SharedPreferences APIs [28] in Android. The SharedPreferences APIs save data as plaintext by default, if the malicious people obtain the Android phone by some means, they will be able to easily compromise the secret key by navigating to the file where the key is stored. Hence, it is recommended to use the EncryptedSharedPreferences APIs [29] which is an implementation of the SharedPreferences. Unlike the SharedPreferences, EncryptedSharedPreference encrypts the data using AES, then saves the ciphertext on the disk. The login Activity is the entry point of the application which is a single screen as shown in Fig 3.3. The login Activity provides text fields for users to enter emails and passwords. We also added a "remember me" checkbox to enables the user to save the credential persistently thereby avoiding repeatedly entering credentials. When users pass the authentication process, the login activity is killed, the main Activity starts and becomes visible to users. The main Activity is a single screen which is divided into two portions, of which the top half portion contains buttons and the bottom display the google map.

Afterwards, we concentrated on developing components according to our designed

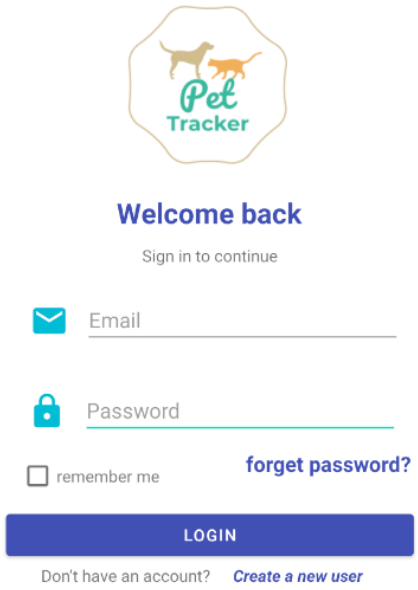

Figure 3.3: Login screen

Figure 3.4: Signup screens

architecture in Fig 1.2. To implement the authentication process between the mobile and the cloud base, we used Firebase which is widely used to authenticate the users in the Android application. The database provides APIs to integrate the authentication into the Android application in an easy manner. To authenticate with Firebase, the application is required to pass the credential to Firebase through provisioned APIs, then Firebase will verify the credentials and return a response to indicate if the authentication is successful or not. The createUserWithEmailAndPassword() method allows takes email and password as a parameter, validates them and then create a new user. The signInWithEmailAndPassword() method submits email and password to Firebase, the database validates them and in return, the application receive the message to indicate whether the validation is successful. To implement the SMS transmission components, the SmsManager class contained in the Telephony manager component manages operations for sending SMS's in the application, see Appendix B. We created a subclass of BroadcastReceiver [30], then implemented the onReceive method for reading the incoming messages in the application, see Appendix A.

GPS Extractor simply takes a Java string as a parameter, uses regular expression to extract the coordinates in a format of "latitude, longitude" and passes it to the google map object. For implementing the digital maps, Android, in fact, provides facilities to integrate Google maps into the application rather than requiring the developers to write the maps programmatically. The MapView object allows to display a static Google maps on the screen. Users are unable to interact with the map until a GoogleMap object is created. A GoogleMap object automatically performs the following operations [31]:

- Connecting the Android app to the Google Maps service.
- Adding and enabling various controls on the map, and responding to user's gestures by moving the map or Zooming in and out.

In addition, we added a button at the top left corner of the map for users to expand the map to full screen or collapse the map back to half screen. Fig 3.5 shows the screenshot of displaying the map in default size and full size. To enable the users to specify how many minutes they intend the GPS location to be transmitted from the tracker, we added a NumberPicker widget in Android UI library, from which users can scroll up and down to select a integer value from 1 to 10. The application appends the selected number to the request location message which will be sent to the tracker. Last but not least, considering users would accidentally touch the button

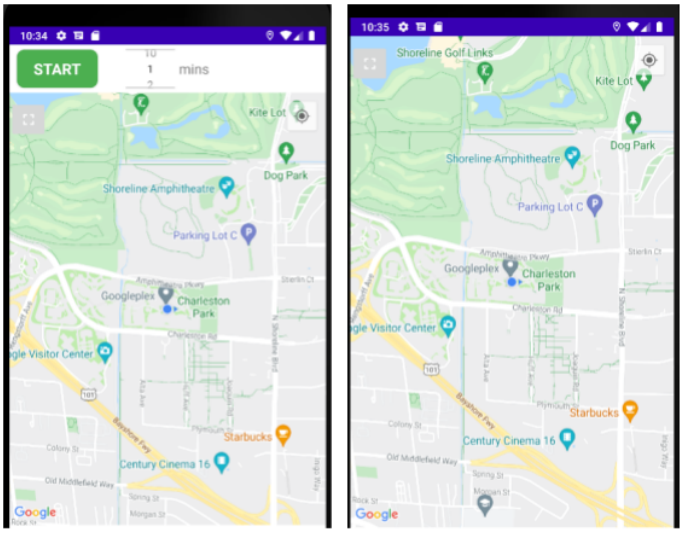

Figure 3.5: The google map

when holding the mobile phone in their hands, we used long-press buttons that require users to long hold the button to trigger the start and the stop action, rather than one tapping on them.

Regarding the implementation of AES cryptography, the Java Cryptography Architecture (JCA) provides a set of APIs which implement concepts of modern cryptography such as symmetric and asymmetric cryptographic algorithms. The core feature of JCA is the engine classes, which provides an interface to a specific type of cryptographic operation. The cipher class is one of the engine classes that we used to implement the AES cryptography, which encapsulates underlying mathematically cryptographic operation for data encryption and decryption. With the means of the cipher class, developers are able to implement a certain cryptography in a couple of lines of codes.

Once the encryption was implemented, we found that the ciphertext that was generated by the implementation was represented in the form of the garbled text. Initially, we tried to transmit the ciphertext directly over SMS. However, it turned out that the ciphertext, which was received by the other end, ended up becoming corrupted which resulted in the failure of decryption. After looking into it, we figured that these garbled characters are representations of the binary. In fact, binary ciphertext contains any byte values which are incompatible with character encoding, such as ISO/IEC 8859-1 [32] which does not allow all bytes to be used. Additionally,

some characters are called “control characters”, which means they are not printable in text. Hence, when the binary ciphertext is being transmitted over the SMS, if the cellular towers or the receiver devices failed to interpret some of the characters in the binary ciphertext, they use the generic replacement character (“”) in places instead, which results in the corruption of the ciphertext. Therefore, it is recommended to encode the ciphertext to Base64 [33] format when we want to store or transmit it as text. The entire encryption/decryption operation flow between two ends is shown in Fig 3.6. The implementation of the encryption and decryption in Android is shown in Appendix D.

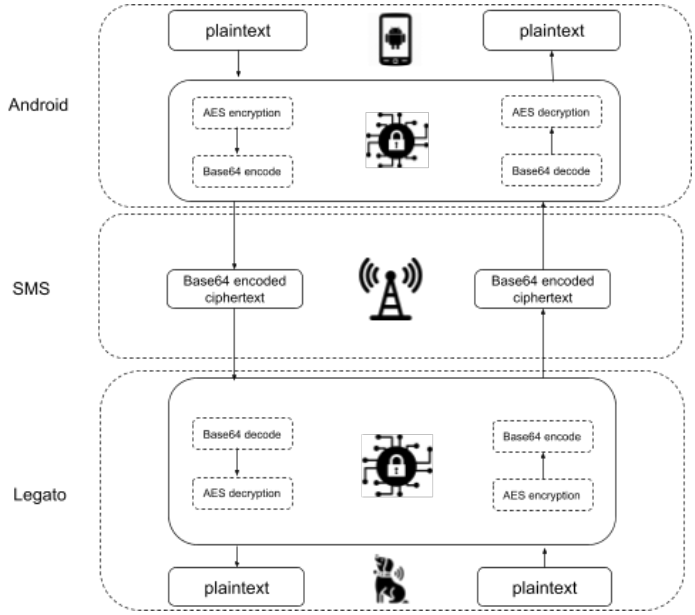

Figure 3.6: encryption/decryption operation flow

Fig 3.7 illustrates the software architecture of the Android application. We added AES encryption in the SMS Sender component to ensure any outgoing SMS is automatically encrypted and AES decryption in the SMS Receiver component to decrypt the incoming SMS before passing it to the activity. All activities have a SMS Receiver and SMS Sender installed in order to receive and send the SMS messages.

3.6 Testing and Validation

Once the Android application was implemented, I ran the app on virtual devices with different screen size to ensure the UI components scale correctly. Fig 3.8 shows how

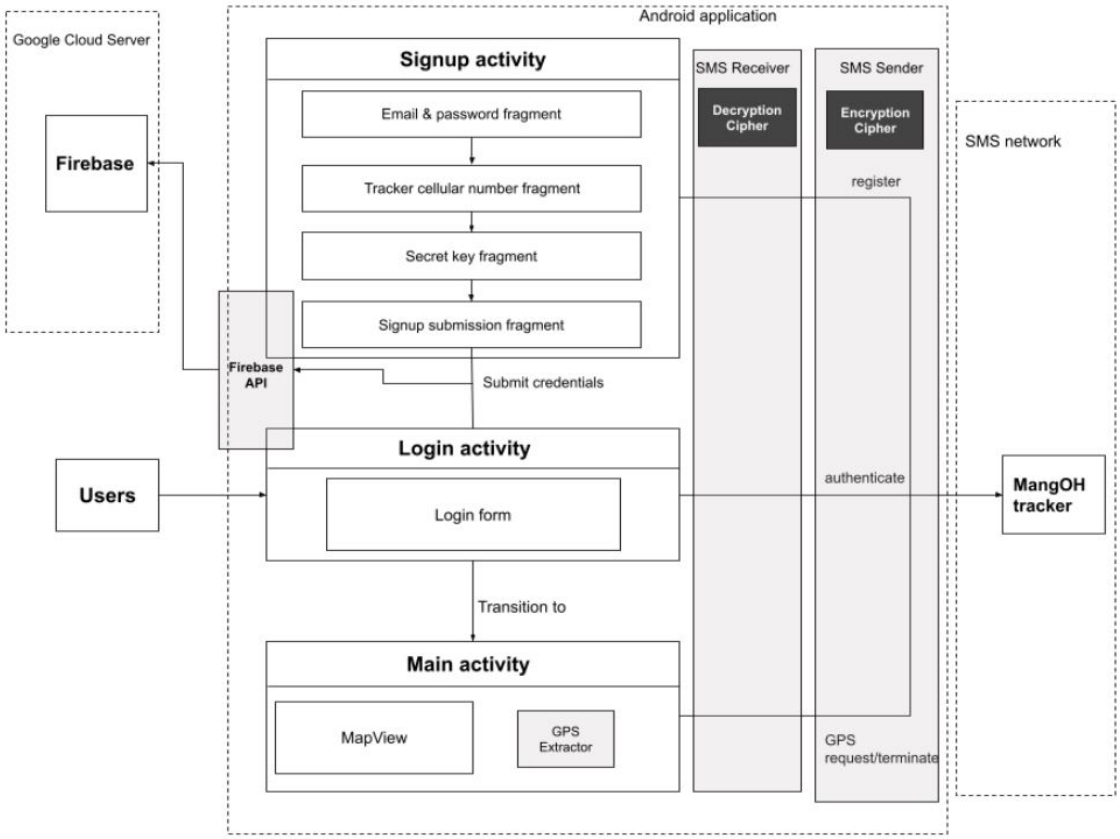

Figure 3.7: Android application architecture

the login activity displays on different screen resolutions of 1440x2560, 1080x2220, 1080x1920, from left to right. Afterwards, we set up a meeting with Dr. Xiaodai Dong to perform end user evaluation. We primarily focused on testing usability and learnabiltiy through asking the end user to perform a series of tasks to interact with the app. We conducted the evaluation on the Android emulator within Android Studio, because the emulator allows us to simulate incoming SMS messages. Initially, the users are given a made-up tracker cellular number and secret key for testing purposes. We asked the users to go through the signup process to set up an account for the app. We observed that the users were able to smoothly transition from one fragment to another to create an account. Meanwhile, since the firebase was in place at the time, we kept monitoring the database console to make sure whether the signup process succeeds or not. When a user successfully creates an account, we could see that a new email address and encrypted password were added in the database.

As we moved to the login activity, the participants became more comfortable with

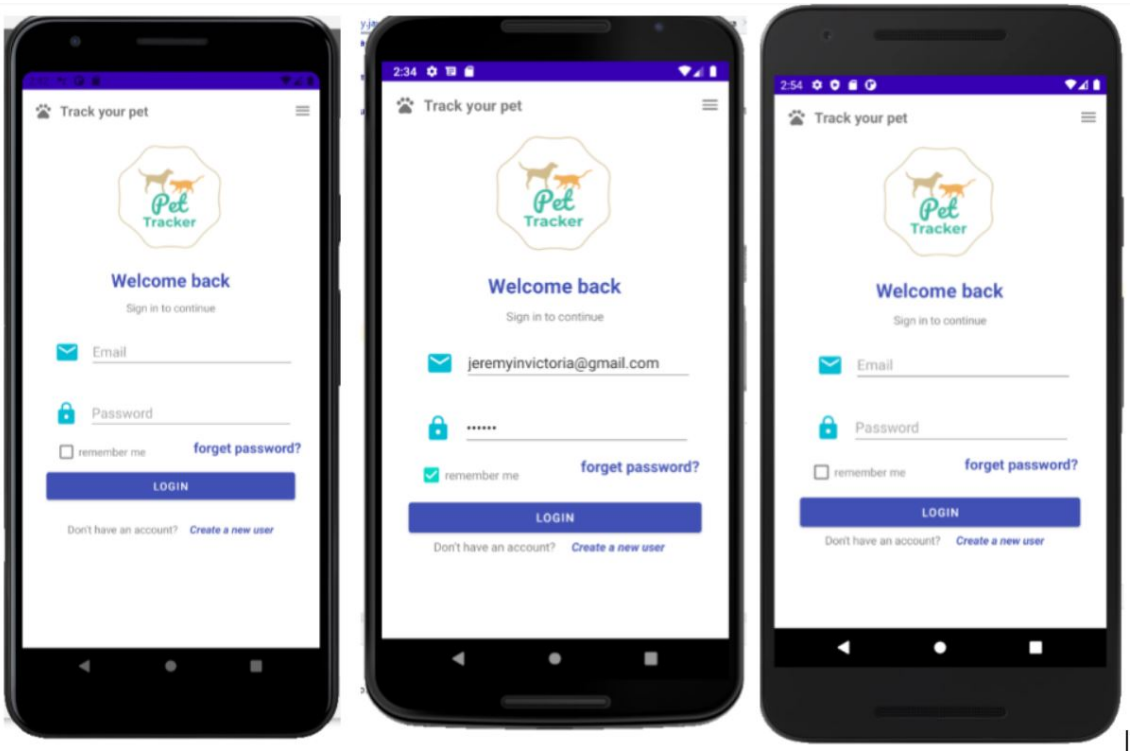

Figure 3.8: UI displayed on different screen resolution

the user interface. However, all users failed to login in the first place, although we were sure their accounts had been added to the firebase through the previous signup process. We checked the source code, and found out that users have to verify their email addresses prior to the first-time login. The participants failed to verify the emails because the app did not notify them to do so. Therefore, we decided to add a toast which displays a small message on the screen to tell users verify their emails if they failed to do so. We also added a notification for email verification on the submission fragment in the signup activity.

While testing the main activity, the users got confused with the long-pressed buttons in the first place, but we observed that they could get used to them after a couple of attempts. Therefore, we decided to leave the buttons as they are.

Chapter 4

Tracker and Firmware

MangOH Red designed and manufactured by Sierrawireless is a credit card-sized, industrial-grade IoT device which can connect to any 2G, 3G or 4G LTE mobile network with Common Flexible Form Factor (CF3) wireless based modules. In addition, the device includes Global Navigation Satellite System (GNSS) capabilities. The GNSS implementation supports GPS L1 [34] operation. In essence, the device features an integrated Linux-based processor configured with the Legato Application Framework. The Framework enables Legato developers to build and install their Legato applications on the tracking device. The Legato Application Framework also provides APIs for developers to bind to their applications and interface directly with the hardware services.

4.1 MangOH Hardware Architecture

The MangOH Red hardware platform is founded upon a Common Flexible Form Factor (CF3) module. The CF3 module not only provides the hardware the essential connectivity between Wi-Fi/Bluetooth, LTE-M, NB-IoT, 4G, 3G and 2G modules, but also supports bit-pipe and value-add solutions [35]. There are four major components in MangOH, as follows:

- Processor
- Card slot
- Connector
- Antenna

The processor components are CF3 module and Wi-Fi/Bluetooth chipset which supports Wi-Fi/Bluetooth/WLAN connections via an integrated 2.4 GHZ antenna. The chipset also provides access to the accelerometer and temperature sensor as long as these are attached to the device. Under the card slot category, both Micro-SIM holder and USIM are used to establish mobile network connectivity and allow the device to use the internet. MicroSD holder provides access to microSD card. The MangOH red hardware supports a few USB-type connectors and a Raspberry Pi connector. The USB connectors are primarily used by firmware developers to connect to CF3 module console via their computers, while the Raspberry connector connects the mangOH hardware components to a Raspberry Pi. In terms of antenna, the hardware contains a main antenna for cellular networks, a GNSS (Global Navigation Satellite System) antenna for GPS acquisition, a diversity antenna to increase the quality and reliability of wireless connectivity and a Wi-Fi/Bluetooth shared antenna. Finally, MangOH red supports different sensors, such as an accelerometer, a pressure/temperature sensor and a light sensor. Fig 4.1 shows an architecture of the mangOH Red's hardware components relative to the CF3 module.

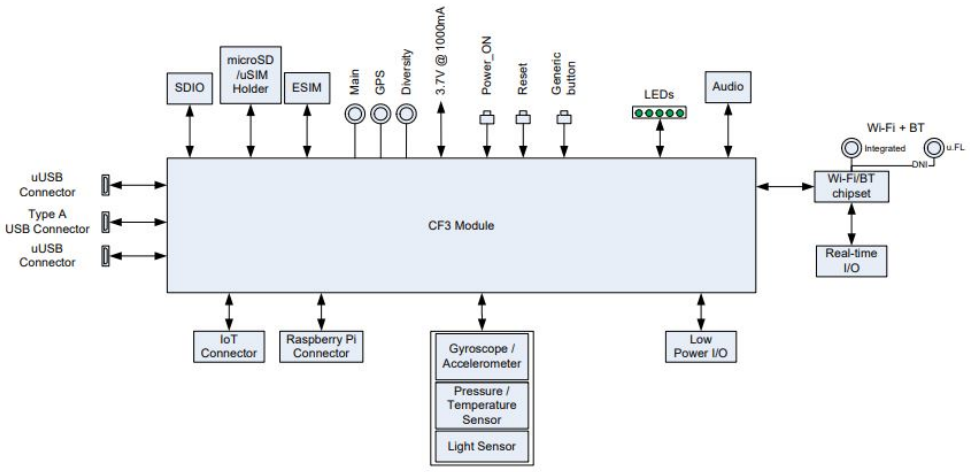

Figure 4.1: MangOH hardware architecture

4.2 MangOH Firmware Architecture

MangOH features an integrated Linux-based processor configured with the Legato Application Framework. The Framework enables Legato developers to build and install their Legato applications on the tracking device. The Legato Application Framework

also provides APIs for developers to bind to their applications and interface directly with the hardware services.

Prior to unveiling Legato Application Framework, we have to briefly introduce Legato architecture. Similar to the Android architecture, the Legato architecture is also a software stack that contains, as shown in Fig 4.2 from bottom to top, hardware, operating system, application framework, platform services and applications. The hardware refers to the physical MangOH board including major hardware infrastructures, such as GNSS, modem and power. The operating system is an embedded linux distribution called Yocto [36]. The application framework and the platform services form the Legato Application Framework which is used to build the application and Legato application layer contains all firmware applications on the device. It is worthwhile to note that programmers who intend to build applications on Legato are required to focus on the top three layers.

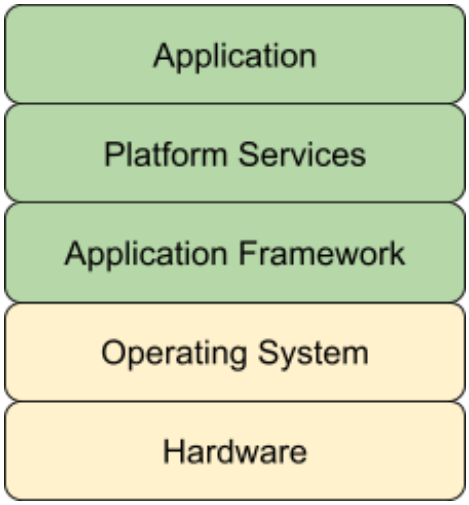

Figure 4.2: Legato architecture

In general, an application framework is a set of software libraries which is implemented to provide a fundamental structure to build softwares in a specific environment. An application framework helps programmers to avoid creating applications from scratch and thereby greatly facilitate the design and implementation of the functionality of the application. Similarly, Legato Application Framework was designed to simplify IoT embedded system development on MangOH devices. The Legato AF is responsible for connecting applications that are installed on the target to the underlying service of the device.

As described above, Legato Application Framework comprises of the Application

Framework and Platform Services, which can be broken down into Platform Services and Platform Adapters. Platform Services are a set of default services such as location sensing, audio connections, cellular and data network connections. Platform Adapters provide specific communication functions to trigger the specific service on the hardware when the function is called in Legato application through APIs. So Platform Adapters serve as an interpreter between the hardware infrastructures and Platform Services. Whenever users' application requests to use underlying services of the target through APIs, Platform Adapter receives the request from the certain application in Platform Service, then it performs the operation on the target. It contains the target specific code, in other words, developers do not have to worry about coding in different low level languages such as assembly languages or AT command, while working on Sierra Wireless devices, because Platform Adapter is able to interpret high level languages from Legato framework to low level languages accordingly to trigger hardware functionalities. Platform Services and Platform Adapter function dependently.

Finally, Application Framework is the top layer which is exposed to Legato developers. Application Framework provides APIs of the services in Platform Services, so Legato developers are able to add these services into their Legato applications via the APIs. In addition, it includes a core daemon which is responsible for management of the Legato applications, such as start, kill and pause applications. The Framework also provides a set of command line tools, which not only includes the configuration and administration of deploying applications and configuring the target device, but also the interaction with Platform Services over the Shell.

4.3 Implementation

Similar to develop the Android application, we implemented the firmware application on MangOH alongside the defined requirements. Prior to writing codes, I visualized the firmware application architecture to help the team understand how different components interact, shown in Fig 4.3. The arrows represent the data flow between components in the firmware system. Firstly, we focused on creating the SMS Receiver and SMS Sender components. The Legato platform provides `le_sms_Send()` method to enable applications to send the SMS messages. In order to read the incoming SMS messages in the Legato application, we had to register an incoming messages handler using Legato API `le_sms_AddRxMessageHandler()`. The API provides and

stores all information of the incoming messages on a Message handler object, such as content of the message and cellular number that the message was sent from. To get both message content and the sender's number, using `le_sms_GetSenderTel()` and `le_sms_GetText()` enables the application to get message content and the sender's number respectively.

Once the message is successfully received, the message is passed to the Message processor component in which the application interprets the message to determine whether the mobile user wants to pair, start or terminate GPS acquisition. Since the messages are already defined in the system design phase, the application actually compares the message with the pre-defined messages to check if they match in the Message processor component. The GPS service component involves the interaction with the GNSS service on MangOH. The GNSS service consists of three states: GNSS service initiation, GNSS acquisition, GNSS service termination/release. The application requests to acquire GNSS information every time, the GNSS service will loop through three states to complete the task. To start GNSS service in the application, Legato provides `le_posCtrl_Request` API to request activation of the positioning service. The API returns either a reference to the service activation request which will be used later for releasing the request or NULL when the tracker fails to activate the service. When the service is successfully activated, the `le_pos_Get2DLocation` API allows the application to start obtaining GPS coordinates. For each acquired location, the application forwards it to the user through `le_sms_Send` API. In addition, we added a timeout mechanism to handle the exceptions. The timeout defines the length of time, in seconds, in which the application keeps attempting to establish connections with the satellite to get the first fix of the location. The timeout value is 60 seconds, which means the application will release the resources that were used for GNSS service if it fails to obtain the location within that amount of time. The application would also send an error message to the user to indicate the failure.

Regarding the database that the system needs as part of the authentication mechanism, Legato does not have any built-in relational database. So we installed SQLite on the Legato platform instead. Unlike other famous SQL databases such as MySQL and Oracle, SQLite is a lightweight relational database that is widely used on embedded Linux. SQLite implements a serverless engine which reads and writes directly to disk files. In terms of memory utilization, SQLite uses small memory space but still has good performance on speed. As stated in the Proposed System section, when the tracker retrieves data in the database, it only looks for two pieces of information:

phone number and IMEI. Therefore, only one table exists in the database which was defined as follows:

Attributes	Definition
ID	Primary, Unique ID
Phone	Phone number of the authorized users
Fingerprint	IMEI of the authorized user's phone

To implement cryptography on the tracker end, Legato AF has the built-in OpenSSL crypto-library which implements a wide range of cryptographic algorithms. OpenSSL is a popular open source toolkit which provides cryptographic encryption and decryption APIs that the developers can use in their C/C++ program. For implementing AES encryption and decryption, OpenSSL provides `AES_encrypt()` and `AES_decrypt()` functions, but it is discouraged to use these functions for AES implementation in fact. Firstly, they are a software-only routine which does not use hardware acceleration, such as AES-NI [37], at all, so the implemented AES cryptography using this method is vulnerable to side channel attacks [38]. Secondly, they could suffer from endianness issues on some obscure platforms [39]. Instead, the `EVP_*` interfaces, which have the hardware acceleration and endianness issues resolved, are recommended to use for AES implementation. In addition, the `EVP_*` interfaces also provide functions for Base64 encoding/decoding. The implementation of AES in Legato shown in Appendix I.

Fig 4.3 illustrates the firmware application architecture. The firmware components are connected to the hardware components through Legato API's. Each inbound or outbound SMS message has to go through the cellular network connectivity component.

4.4 Testing and Validation

When the implementation of the firmware application completed, we installed the application on the MangOH device and tested it with the Android application. In order to setup the testing environment, we chose to conduct the testing in the urban area which is not under high forest cover. We powered the MangOH device through a Linux laptop which enabled us to carry it walking around to test GPS location accuracy as well as monitor the firmware application log at the time.

We started with the first-pairing process in which the user has to go through the signup process to pair the phone with the tracking device and create an account via

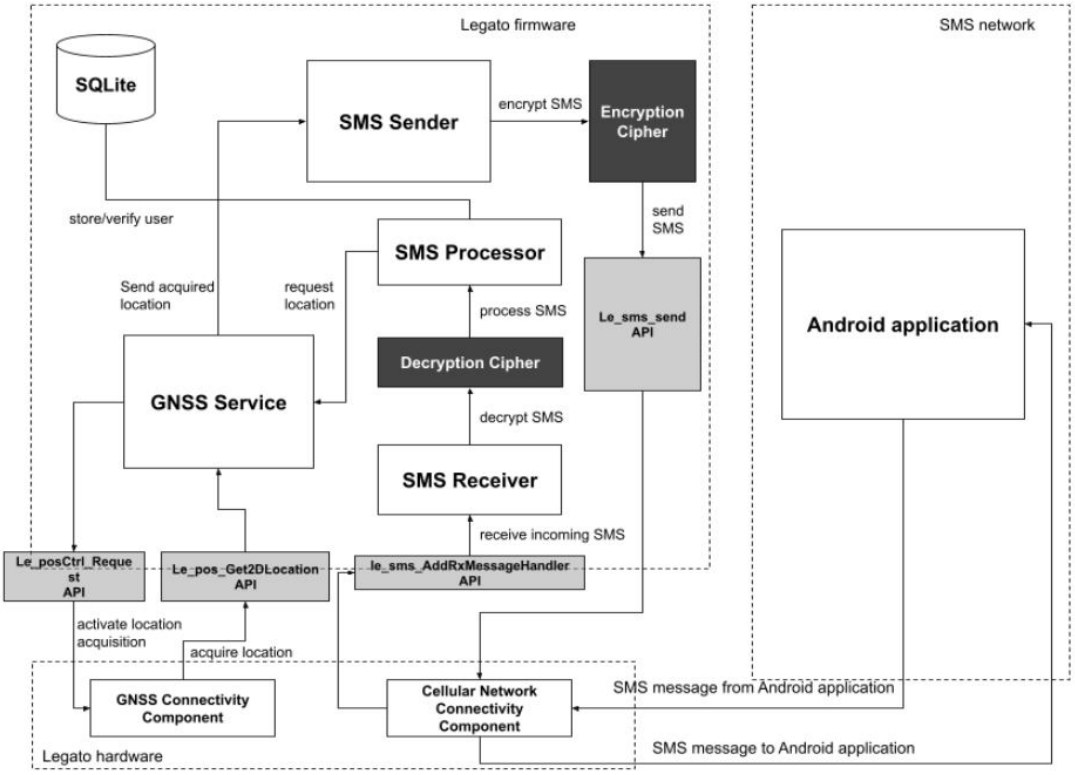

Figure 4.3: Firmware application architecture

the Android application. Meanwhile, we monitored both the SQLite database on the firmware as well the firebase console. Subsequently, we tested the login process once the pairing was set up. As pairing and login process went on, we noticed a significant system delay, due to the GSM network on delivering the SMS from the MangOH tracker to the Android application as well as encryption and decryption on the firmware, which can have an negative impact on the user experience. As we moved to GPS acquisition testing, we started to carry the MangOH roaming around, and log the GPS coordinate acquired by the tracker. Fig 4.4 shows the Android app visualized the GPS location on the Google map. Later on, we compared the logged GPS coordinates of the tracker with the coordinates acquired by Google map. Table 4.1 shows the MangOH GPS coordinate differs from the Google map GPS coordinate in meters.

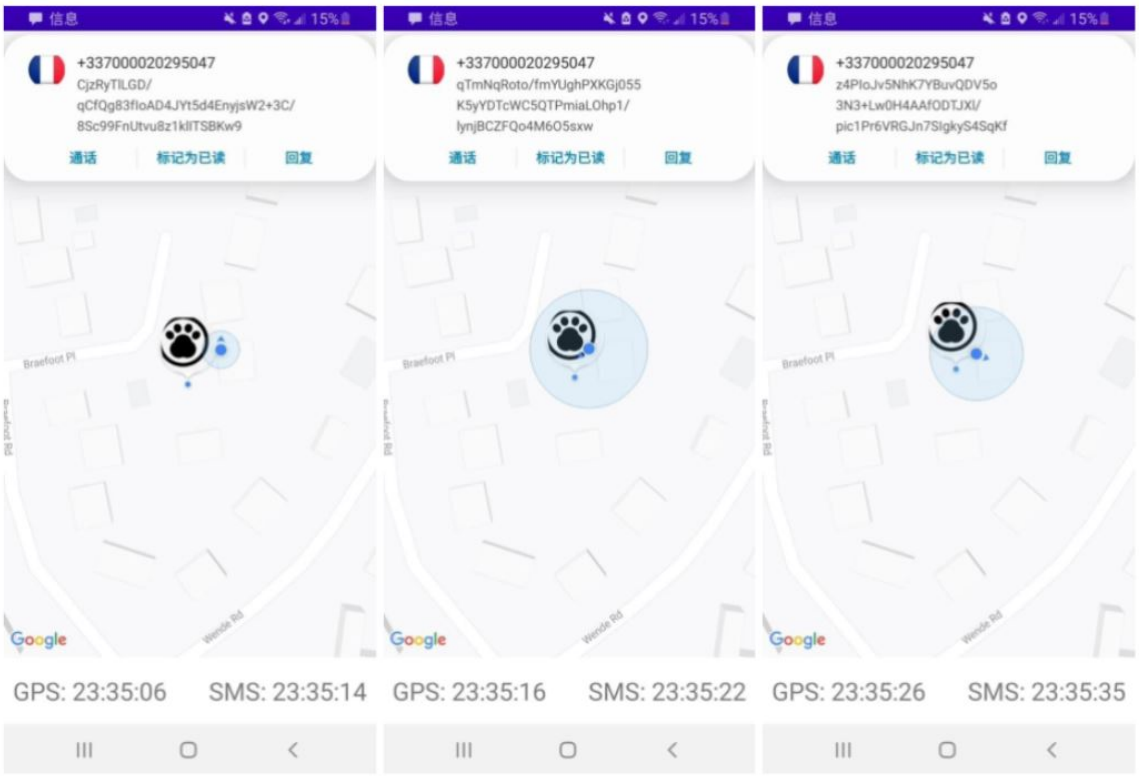

Figure 4.4: Visualized received GPS on Google map

Google map GPS	MangOH GPS	Deviation (meters)
48.467545, -123.345662	48.46754, -123.34567	1.564
48.467602, -123.345616	48.46762, -123.34562	1.953
48.467681, -123.345602	48.46768, -123.34561	1.481
48.467747, -123.345597	48.46775, -123.34560	0.636

Table 4.1: GPS accuracy compared to Google map

Chapter 5

Conclusion and Future Work

Our pet tracking system is an IoT application primarily based on the SMS communication. Not only does it acquire and transmit GPS location in real time, but also it provides cellular networks level tracking range. In addition, taking the security concern into account, the system is protected by the additional security layer, which was designed and implemented by the development team, such as encryption of the SMS messages, authentication mechanism covering both ends. In the future, researchers can potentially add a temperature sensor to the tracker by which the user is able to know the amount of heat energy or even coldness that is generated by the pet. As a note to future researchers, regarding the hardware part, the tracking device needs a case and a battery. Considering pets, especially dogs, would swim in the water while wearing the tracker, it is important to design a water resistant case to ensure the tracker can still function in spite of becoming wet.

Appendix A

Android code for Receiving SMS

```
1  import android.content.BroadcastReceiver;
2  import android.content.Context;
3  import android.content.Intent;
4  import android.os.Bundle;
5  import android.telephony.SmsMessage;
6
7  public class ReceiveSms extends BroadcastReceiver {
8      public String msgs;
9      public String source_tel;
10     private static final String SMS_RECEIVED = "android.
           provider.Telephony.SMS_RECEIVED";
11     @Override
12     public void onReceive(Context context, Intent intent) {
13         if(intent.getAction().equals(SMS_RECEIVED)) {
14             Bundle bundle = intent.getExtras();
15             if(bundle != null) {
16                 try {
17                     Object[] pdus = (Object[] ) bundle.get("
                           pdus");
18                     final SmsMessage[] message = new
                           SmsMessage[pdus.length];
19
20                     for (int i = 0; i < pdus.length; i++) {
21                         String format = bundle.getString("
                               format");
```

```
22                          message[i] = SmsMessage.createFromPdu
                               ((byte[]) pdus[i]);
23                          source_tel = message[i].
                               getOriginatingAddress();
24                          msgs = message[i].getMessageBody();
25                      }
26                  } catch(Exception e) {
27                      e.printStackTrace();
28                  }
29              }
30          }
31      }
32  }
```

Appendix B

Android code for sending SMS

```
import android.telephony.SmsManager;

public static boolean sendMessage(String tel, String msg) {
    try{
        SmsManager smsManager = SmsManager.getDefault();
        smsManager.sendTextMessage(tel, null, msg, null,null);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

Appendix C

Android code for extracting GPS coordinates

```
public final static GPSCoordinate getCoordinationFromSms(
   String smsTexts) {
    String lat = "";
    String lng = "";
    String temp = smsTexts.substring(5);
    //"loc: 123.45678, -45.456789"
    else {
        String[] latlng = temp.split("\\s*,\\s*");
        lat = latlng[0];
        lng = latlng[1];
    }
    return new GPSCoordinate(lat, lng);
}
```

Appendix D

Android code for AES encryption/decryption

```
1  import java.security.NoSuchAlgorithmException;
2  import java.util.Base64;
3  import java.math.BigInteger;
4  import javax.crypto.Cipher;
5  import javax.crypto.spec.*;
6
7  public String aes_encrypt(String content) {
8      try {
9          IvParameterSpec iv = new IvParameterSpec(initVector.
               getBytes("UTF-8"));
10         Cipher aesECB = Cipher.getInstance("AES/CBC/
               PKCS5Padding");
11         SecretKeySpec key = new SecretKeySpec((this.passwd).
               getBytes(), "AES");
12         aesECB.init(Cipher.ENCRYPT_MODE, key, iv);
13         return Base64.getEncoder().encodeToString(aesECB.
               doFinal(content.getBytes("UTF-8")));
14     } catch (Exception e) {
15         e.printStackTrace();
16     }
17     return null;
18 }
19
```

```
20 public String aes_decrypt(String content) {
21     try {
22         IvParameterSpec iv = new IvParameterSpec(initVector.
               getBytes("UTF-8"));
23         Cipher cipher = Cipher.getInstance("AES/CBC/
               PKCS5Padding");
24         SecretKeySpec key = new SecretKeySpec((this.passwd).
               getBytes(), "AES");
25         cipher.init(Cipher.DECRYPT_MODE, key, iv);
26         return new String(cipher.doFinal(Base64.getDecoder().
               decode(content)));
27     } catch (Exception e) {
28         e.printStackTrace();
29     }
30     return null;
31 }
```

Appendix E

Legato code for sending SMS

```
#include "legato.h"
#include "interfaces.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

le_result_t sms_SendMessage( const char* destinationPtr, const
    char* textPtr)
{
  le_result_t res;
  le_sms_MsgRef_t myMsg;

  myMsg = le_sms_Create();
  if(!myMsg){
    LE_ERROR("SMS message creation has failed");
    return LE_FAULT;
  }

  res = le_sms_SetDestination(myMsg, destinationPtr);
  if(res != LE_OK){
    LE_ERROR("le_sms_SetDestination has failed (res.%d)!", res
       );
    return LE_FAULT;
  }
```

```
24
25    res = le_sms_SetText(myMsg, textPtr);
26    if(res != LE_OK){
27      LE_ERROR("le_sms_SetText has failed (res.%d)!", res);
28      return LE_FAULT;
29    }
30
31    res = le_sms_Send(myMsg);
32    if(res != LE_OK){
33      LE_ERROR("le_sms_Send has failed (res.%d)!", res);
34      return LE_FAULT;
35    }else{
36      LE_INFO("\"%s\" has been successfully sent to %s.",
          textPtr, destinationPtr);
37    }
38    le_sms_Delete(myMsg);
39    return LE_OK;
40  }
```

Appendix F

Legato code for receiving SMS

```
#include "legato.h"
#include "interfaces.h"

static le_sms_RxMessageHandlerRef_t RxHdlrRef;
static char tel[LE_MDMDEFS_PHONE_NUM_MAX_BYTES];
static void RxMessageHandler(le_sms_MsgRef_t msgRef,void*
   contextPtr)
{
  PrintMessageContext_t * msgContextPtr = (
     PrintMessageContext_t *)(contextPtr);
    le_result_t res;

    if ( (msgContextPtr->msgToPrint != -1) && (msgContextPtr->
       msgToPrint != msgContextPtr->nbSms) ){
        /* Skipping message */
        msgContextPtr->nbSms++;
        return;
    }
  res = le_sms_GetSenderTel(msgRef, tel, sizeof(tel));
    char text[LE_SMS_TEXT_MAX_BYTES] = {0};
    if (res == LE_OK){
        res = le_sms_GetText(msgRef, text, sizeof(text));
    LE_INFO("Message content: \"%s\"", text);
    }

```

```
23    if (msgContextPtr->shouldDeleteMessages){
24          res = le_sms_DeleteFromStorage(msgRef);
25          LE_ASSERT((LE_OK == res) || (LE_NO_MEMORY == res));
26
27          le_sms_Delete(msgRef);
28      }
29
30      msgContextPtr->nbSms++;
31
32
33  }
34
35  /**
36   * This function installs an handler for message reception.
37   */
38  void sms_Receiver(void)
39  {
40    static PrintMessageContext_t context = {
41          .nbSms = 0,
42          .shouldDeleteMessages = false,
43          .msgToPrint = -1,
44      };
45    RxHdlrRef = le_sms_AddRxMessageHandler(RxMessageHandler, &
        context);
46
47  }
```

Appendix G

Legato code for implementing Message Processor

```
void messageProcessor(char* tel,char* text)
{
  int len;
  if (strstr(text, "SIGNUP")) {
    //if tel exists in database, require LOGIN
    if(findNumber(tel) == LE_OK) {
      //40001: tel is registered already, please sign in
      strcpy((char *)content, "CODE: 40001");
      printf("Sending... %s", content);
      sms_SendMessage(tel, (char*)content);
    }
    else {
      char *fingerprint = strremove(text, "SIGNUP+");
      if(LE_OK == insertValue(tel, fingerprint)){
        //20002: signup succeed
        strcpy((char *)content, "CODE: 20002");
        printf("Sending... %s", content);
        sms_SendMessage(tel, (char*)content);
      } else {
        //40005: database error
        strcpy((char *)content, "CODE: 40005");
        printf("Sending... %s", content);
        int result = aes_encrypt(content,strlen((const char*)
```

```
            content), ciphertext, &len, (unsigned char*)
               shared_key);
24          int encode_str_size = EVP_EncodeBlock(base64,
               ciphertext, len);
25          printf("encryption: %d, base64_encode: %d", result,
               encode_str_size);
26          sms_SendMessage(tel, (char*)base64);
27        }
28      }
29
30    }
31    if(strstr(text, "LOGIN")) {
32      if(findNumber(tel) == LE_OK) {
33        //20000: authenticated
34        strcpy((char *)content, "CODE: 20000");
35        sms_SendMessage(tel, (char*)content);
36      }else {
37        //NOT found send error code to prompt SECCODE on Android
38        //40004: tel not in database/registered
39        strcpy((char *)content, "CODE: 40004");
40        sms_SendMessage(tel, (char*)content);
41      }
42    }
43    //user requested location
44    if(strstr(text, "START")) {
45
46      if(findNumber(tel) == LE_OK) {
47        //check whether user wants any
48        int mins = atoi(text);
49        GPSService(mins);
50      }
51    }
52    //user wants to stop receiving location
53    if(strstr(text, "STOP") && findNumber(tel) == LE_OK)
         terminate_GPSService();
54
55  }
```

Appendix H

Legato code for GPS service

```
static le_timer_Ref_t gpsTimerRef;
le_result_t GetCurrentLocation( int32_t *latitudePtr, int32_t
   *longitudePtr,int32_t *horizontalAccuracyPtr,uint32_t
   timeoutInSeconds){
  le_result_t result;
  const time_t startTime = time(NULL);
  LE_INFO("Checking GPS position");
  while(true){
    result = le_pos_Get2DLocation(latitudePtr, longitudePtr,
       horizontalAccuracyPtr);
    if(result == LE_OK){
      break;
    }else if((timeoutInSeconds !=0)&&(difftime(time(NULL),
       startTime)>(double)timeoutInSeconds)){
      result = LE_TIMEOUT;
      break;
    }else{
      sleep(1);
    }
  }
  return result;
}
static void GpsTimer(le_timer_Ref_t gpsTimerRef){
    int32_t latitude;
    int32_t longitude;
```

```
22      int32_t horizontalAccuracy;
23    //get loction from GetCurrentLocation
24    //if cant get location in 60s, send Code: 40006
25      const le_result_t result = GetCurrentLocation(
26          &latitude, &longitude, &horizontalAccuracy, GPSTIMEOUT
                *60);
27      if (result == LE_OK)
28      {
29      //DEBUG: Get current time when location is available
30      time_t now = time(NULL);
31      unsigned char absoluteTimeStamp[256];
32      sprintf((char*)absoluteTimeStamp, "%ld", now);
33
34      //send valid location to Android end
35      LE_INFO("Loc:%f,%f", (float)latitude/1000000.0, (float)
           longitude/1000000.0);
36          sprintf((char*)content, "loc: %f, %f;", (float)
               latitude/1000000.0, (float)longitude/1000000.0);
37      //"loc: 123.2131, -34.232423;10:23:32"
38      ustrcat(content, absoluteTimeStamp);
39          sms_SendMessage(tel, (char*)content);
40
41      }
42      else
43      {
44          latitude = DefaultLatitude;
45          longitude = DefaultLongitude;
46          LE_INFO(
47              "Couldn't get GPS location.  Publishing default
                   location: %d, %d",
48              latitude,
49              longitude);
50      //send default GPS
51      //40006: GPS failed
52      strcpy((char *)content, "CODE: 40006");
53      sms_SendMessage(tel, (char*)content);
54      le_timer_Stop(gpsTimerRef);
```

```
55
56      }
57  }
58  void GPSService(int mins){
59      le_clk_Time_t  timerInterval = {.sec =
            GPS_SAMPLE_INTERVAL_IN_SECONDS, .usec = 0};
60    gpsTimerRef = le_timer_Create("GPS Timer");
61    le_timer_SetInterval(gpsTimerRef, timerInterval);
62    le_timer_SetRepeat(gpsTimerRef, 6*mins);
63    le_timer_SetHandler(gpsTimerRef, GpsTimer);
64    // Explicitly call the timer handler so that the app
        publishes a GPS location immediately
65    // instead of waiting for the first timer expiry to occur.
66    GpsTimer(gpsTimerRef);
67    le_timer_Start(gpsTimerRef);
68  }
69
70  void terminate_GPSService(le_timer_Ref_t gpsTimerRef){
71      le_timer_Stop(gpsTimerRef);
72  }
```

Appendix I

Legato code for AES encryption/decryption

```
int aes_encrypt(unsigned char* in, int inl, unsigned char *out
    , int* len, unsigned char * key){
  int result = 0;
  unsigned char iv[16] = "encryptionIntVec";
  EVP_CIPHER_CTX ctx;
  EVP_CIPHER_CTX_init(&ctx);
  result = EVP_EncryptInit_ex(&ctx, EVP_aes_256_cbc(), NULL,
     key, iv);
  *len = 0;
  int outl = 0;
  result = EVP_EncryptUpdate(&ctx, out+*len, &outl, in+*len,
     inl);
    *len+=outl;
    int test = inl>>4;
    if(inl != test<<4){
      result = EVP_EncryptFinal_ex(&ctx,out+*len,&outl);
      *len+=outl;
  }
  result = EVP_EncryptFinal_ex(&ctx,out+*len,&outl);
  EVP_CIPHER_CTX_cleanup(&ctx);
  return result;
}

```

```
21 int aes_decrypt(unsigned char* in, int inl, unsigned char *out
       , unsigned char *key){
22   int result = 0;
23   unsigned char iv[16] = "encryptionIntVec";
24   EVP_CIPHER_CTX ctx;
25   EVP_CIPHER_CTX_init(&ctx);
26   result = EVP_DecryptInit_ex(&ctx, EVP_aes_256_cbc(), NULL,
         key, iv);
27   int len = 0;
28   int outl = 0;
29   result = EVP_DecryptUpdate(&ctx, out+len, &outl, in+len, inl
         );
30     len += outl;
31     result = EVP_DecryptFinal_ex(&ctx, out+len, &outl);
32     len+=outl;
33   out[len]=0;
34   EVP_CIPHER_CTX_cleanup(&ctx);
35   return result;
36 }
```

Bibliography

[1] "2020 Canadian PET population FIGURES Released: News," 2020 Canadian Pet Population Figures Released — News — Canadian Animal Health Institute (CAHI). [Online]. Available: https://www.cahi-icsa.ca/news/2020-canadian-pet-population-figures-released#:~:text=Overall%20pet%20ownership%20in%202020,the%20population%20of%20cats%20stabilized. [Accessed: 16-Sep-2021].

[2] "Missing pet statistics in the United States," Peeva. [Online]. Available: https://peeva.co/missing-pet-epidemic-facts-and-figures#:~:text=The%20American%20Humane%20Association%20estimates,U.S.%20are%20reunited%20with%20their. [Accessed: 16-Sep-2021].

[3] S. Chen, H. Xu, D. Liu, B. Hu and H. Wang, "A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective," in IEEE Internet of Things Journal, vol. 1, no. 4, pp. 349-359, Aug. 2014, doi: 10.1109/JIOT.2014.2337336.

[4] Karen Rose, Scott Eldridge, Lyman Chapin, "THE INTERNET OF THINGS: AN OVERVIEW", pp. 5, Oct 2015.the Internet Society Available: https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf. [Access: 29-Oct-2021]

[5] "(PDF) PetTracker - Pet tracking system USING MOTES." [Online]. Available: https://www.researchgate.net/publication/237368760_PetTracker_-_Pet_Tracking_System_Using_Motes. [Accessed: 16-Sep-2021].

[6] S. Tangsripairoj, P. Kittirattanaviwat, K. Koophiran and L. Raksaithong, "Bokk Meow: A Mobile Application for Finding and Tracking Pets," 2018 15th Interna-

tional Joint Conference on Computer Science and Software Engineering (JCSSE), 2018, pp. 1-6, doi: 10.1109/JCSSE.2018.8457351.

[7] W. by Helen, W. by, Helen, N. P. enter your name here, and P. enter your name here, "LynQ review 2021 [GPS SMART tracker for Hiking, running]," Trail and Kale, 19-May-2021. [Online]. Available: https://www.trailandkale.com/gear/lynq-review-smart-compass/#:~:text=Lynq%20a%20range%20of%20up,Daylight%2Dvisible%20screen. [Accessed: 16-Sep-2021].

[8] M. Braeden, W. by M. Braeden, M. Braeden, S. B. says: Z. says: O. says: "mega888 says: and M. B. says: '10 best Pet trackers and DOG Gps Collars 2021: Techobark,' Technobark, 25-Aug-2021. [Online]. Available: https://technobark.com/best-dog-gps-trackers/. [Accessed: 16-Sep-2021].

[9] "Gps signal plan," GPS Signal Plan - Navipedia. [Online]. Available: https://gssc.esa.int/navipedia/index.php/GPS_Signal_Plan. [Accessed: 17-Sep-2021].

[10] "Canadian wireless data prices among most expensive of 28 COUNTRIES: STUDY," MobileSyrup, 03-Sep-2020. [Online]. Available: https://mobilesyrup.com/2020/09/03/canada-wireless-data-prices-expensive-study/. [Accessed: 16-Sep-2021].

[11] T. Moore, T. Kosloff, J. Keller, G. Manes and S. Shenoi, "Signaling system 7 (SS7) network security," The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002., 2002, pp. III-III, doi: 10.1109/MWSCAS.2002.1187082.

[12] K. Ullah, I. Rashid, H. Afzal, M. M. W. Iqbal, Y. A. Bangash and H. Abbas, "SS7 Vulnerabilities—A Survey and Implementation of Machine Learning vs Rule Based Filtering for Detection of SS7 Network Attacks," in IEEE Communications Surveys Tutorials, vol. 22, no. 2, pp. 1337-1371, Secondquarter 2020, doi: 10.1109/COMST.2020.2971757.

[13] J. Ball.(2014, Jan) "NSA collects millions of text messages daily in 'untargeted' global sweep". the Guardian [online] Available: https://www.theguardian.com/world/2014/jan/16/nsa-collects-millions-text-messages-daily-untargeted-global-sweep [Accessed 4 July 2021].

[14] P. Kruchten. “4+1 architectural view model,” in IEEE Software 12 (6), 1995, pp. 42-50.

[15] M. Hassinen, ”SafeSMS - end-to-end encryption for SMS,” Proceedings of the 8th International Conference on Telecommunications, 2005. ConTEL 2005., 2005, pp. 359-365, doi: 10.1109/CONTEL.2005.185905.

[16] Mahfouz, Ahmed Sayed, Awny Abdel-latef, Bahgat Mahmoud, Tarek. (2010). “Hybrid Compression Encryption Technique for Securing SMS.” International Journal of Computer Science and Security.

[17] Neetesh Saxena and Narendra S. Chaudhari. 2012. A secure approach for SMS in GSM network. In Proceedings of the CUBE International Information Technology Conference (CUBE ’12). Association for Computing Machinery, New York, NY, USA, 59–64. DOI:https://doi.org/10.1145/2381716.2381729

[18] En.wikipedia.org. 2021. Block cipher mode of operation - Wikipedia. [online] Available at: `https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation` [Accessed 4 July 2021].

[19] F. J. D’souza and D. Panchal, ”Advanced encryption standard (AES) security enhancement using hybrid approach,” 2017 International Conference on Computing, Communication and Automation (ICCCA), 2017, pp.647-652, doi: 10.1109/CCAA.2017.8229881.

[20] the National Institute of Standards and Technology (NIST) (2001, Nov 26) Announcing the ADVANCED ENCRYPTION STANDARD (AES) Federal Information Processing Standards Publication 197 pp.19-20.

[21] “Skipjack (cipher),” Wikipedia, 31-Dec-2020. [Online]. Available: `https://en.wikipedia.org/wiki/Skipjack_(cipher)`. [Accessed: 19-Sep-2021].

[22] Wstein.org. 2021. [online] Available at: `https://wstein.org/edu/2010/414/projects/chu.pdf` [Accessed 19 September 2021].

[23] En.wikipedia.org. 2021. RSA (cryptosystem) - Wikipedia. [online] Available at: `https://en.wikipedia.org/wiki/RSA_(cryptosystem)` [Accessed 19 September 2021].

[24] En.wikipedia.org. 2021. Key size - Wikipedia. [online] Available at: https://en.wikipedia.org/wiki/Key_size [Accessed 19 September 2021].

[25] "Google releases first public beta of Android 11," SiliconANGLE, 11-Jun-2020. [Online]. Available: https://siliconangle.com/2020/06/10/google-releases-first-public-beta-android-11/. [Accessed: 23-Dec-2021].

[26] M. Backes, S. Bugiel, O. Schranz, P. Von Styp-Rekowsky and S. Weisgerber, "ARTist: The Android Runtime Instrumentation and Security Toolkit," 2017 IEEE European Symposium on Security and Privacy (EuroSP), 2017, pp. 481-495, doi: 10.1109/EuroSP.2017.43.

[27] "1.2: Fragment lifecycle and communications," 1.2: Fragment lifecycle and communications · GitBook. [Online]. Available: https://google-developer-training.github.io/android-developer-advanced-course-concepts/unit-1-expand-the-user-experience/lesson-1-fragments/1-2-c-fragment-lifecycle-and-communications/1-2-c-fragment-lifecycle-and-communications.html. [Accessed: 29-Oct-2021].

[28] "SharedPreferences: android developers," Android Developers. [Online]. Available: https://developer.android.com/reference/android/content/SharedPreferences. [Accessed: 19-Oct-2021].

[29] "Encryptedsharedpreferences: android developers," Android Developers. [Online]. Available: https://developer.android.com/reference/androidx/security/crypto/EncryptedSharedPreferences. [Accessed: 19-Oct-2021].

[30] "Broadcastreceiver : Android developers," Android Developers. [Online]. Available: https://developer.android.com/reference/android/content/BroadcastReceiver. [Accessed: 18-Sep-2021].

[31] Google. [Online]. Available: https://developers.google.com/maps/documentation/android-sdk/map. [Accessed: 18-Sep-2021].

[32] "Iec 8859-1," Wikipedia, 06-Apr-2015. [Online]. Available: https://en.wikipedia.org/wiki/ISO/IEC_8859-1. [Accessed: 19-Sep-2021].

[33] "Base64," Wikipedia, 11-Sep-2021. [Online]. Available: https://en.wikipedia.org/wiki/Base64. [Accessed: 19-Sep-2021].

[34] C. Staff, "Why are cell phone plans so expensive in Canada?," Cansumer, 16-Sep-2021. [Online]. Available: https://cansumer.ca/canada-phone-plan-pricing/#:~:text=Canadians%20pay%2020%25%20more%20than,US%20and%20%2427%20in%20Australia. [Accessed: 16-Sep-2021].

[35] "Red Open-source hardware development platform operational ..." [Online]. Available: https://fccid.io/N7NRED/Operational-Description/41110401-mangOH-Red-Developers-Guide-r4-3846689. [Accessed: 29-Oct-2021].

[36] "Yocto project," Wikipedia, 16-Aug-2021. [Online]. Available: https://en.wikipedia.org/wiki/Yocto_Project. [Accessed: 17-Sep-2021].

[37] En.wikipedia.org. 2021. AES instruction set - Wikipedia. [online] Available at: https://en.wikipedia.org/wiki/AES_instruction_set [Accessed 19 September 2021].

[38] S. Bhunia and M. Tehranipoor, "Side-channel attacks," Hardware Security, 09-Nov-2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128124772000137. [Accessed: 19-Sep-2021].

[39] "Application fundamentals nbsp;: nbsp; android developers," Android Developers. [Online]. Available: https://developer.android.com/guide/components/fundamentals. [Accessed: 24-Dec-2021].

[40] "Fragments nbsp;: nbsp; android developers," Android Developers. [Online]. Available: https://developer.android.com/guide/fragments. [Accessed: 24-Dec-2021].