Security and Privacy for Modern and Emerging Mobile Systems

Alexander Hjelm

***Abstract*—The fundamentals of secure systems can be presented as the three cornerstones: authentication (trustworthiness of senders), integrity (inability to alter messages) and confidentiality (inability to read messages for anyone except the intended recipient). In this project, an android application has been programmed by the author with the purpose of sending geographical data over a bluetooth connection in a secure peer-to-peer manner. In the application, the three goals were met primarily through the use of certificate validation, encryption and digital signatures. The final application features a user-interface with a Google Maps interface, and query parameters that the user can set when requesting data about a certain position.**

The final application would very much be considered secure in an environment where only two android devices are communicating. However, more steps would have to be taken if the application is to be deployed commercially. The primary questions are scalability, speed and further security complications.

I. Introduction

LOCATION-BASED services (LBS), for instance Google Places, offer smartphone users information about their surroundings by delivering a set of nearby points of interest (POI). These may be for instance restaurants, hotels or other services. In the particular case of Google Places, there is a privacy concern in that anyone who intercepts a mobile user's data traffic can learn about their whereabouts, the places they frequent or which places they are interested in. There is even the concern of privacy towards the company providing the service, since software companies often maintain databases of all users and their past search queries, and trade this information to other companies.

One solution to this security problem is to let android devices send POIs to each other using a peer-to-peer (P2P) solution, which may be encrypted and include an identity-check to not include any eavesdroppers in the communication. It also limits the exposure to the service provider's server, since data is only fetched once from the server, and all subsequent information exchanges are made between the devices.

A. Internet Security Goals

Consider a computer network make up out of nodes and links. The primary issues that network security aims to solve are, according to [1]:

- Authentication: A node on the network has to only accept data from legitimate sources.
- Integrity: An unauthorized node shall not be able to alter messages. Once the receiver node receives a message, they must verify that the message is the same as it was originally sent by the sender node.
- Confidentiality: Any data that is exchanged over any link has to only be readable by the intended recipient. As for instance wi-fi and bluetooth signals are broadcast though radio waves, any receiver can acquire the message as it is sent. Thus any intermediate nodes must not be able to interpret the message.

B. Assignment

The goal of this project was to program an android application where a peer-to-peer communication is used to share data with other android devices. The data specifically is a set of points of interest from the Google Places API, and thus the application must be able to both request a list of POIs from Google Places, as well as send them over a peer-to-peer link. Finally, the POIs must be communicated between the android devices in a secure and privacy-preserving manner. This is the main focus of the assignment.

C. Program Specifications

The details of the project implementation were left to the author, and they were given the opportunity to freely decide upon which technologies and libraries (within the android scope) to use when programming the app. The following guidelines (program specifications) were set up, in addition to the assignment:

- The program will run an HTTP-server and successfully download the correct POIs from Google, using the Google Places API. The user specifies restrictions on the desired POIs through the user interface.
- The client app will send its POIs to nearby devices using Bluetooth Sockets connectivity, upon request.
- The data sent over the bluetooth connection will be encrypted using the RSA-algorithm.
- The devices will verify each others identities using certificates signed by a trusted third-party, prior to sending any data.
- The client app will store its own POIs internally for display and further communication.
- The program shall work according to specifications in an environment where two android devices are connected via bluetooth.

II. Theory

A. Communication, Bluetooth Sockets API

Bluetooth communication is open to developers via the Bluetooth Sockets API, which is detailed in the official documentation at [2]. Communication is primarily handled though

BluetoothSocket objects in the android API. Connecting the devices works by first establishing a server-client relationship. The server creates a *BluetoothServerSocket* which listens for incoming connection attempts, and returns a BluetoothSocket object to manage the connection once the client has connected. The client itself connects by creating a BluetoothSocket object with the remote device as a *BluetoothDevice* object. As such, both devices access the communications link through a BluetoothSocket object, and once connection has been established, there is no longer any distinction between server and client.

Note that the BluetoothServerSocket will block the current thread until connected, meaning that any further code execution will be delayed until the BluetoothServerSocket receives a connection call from a client. Thus, listening for connections on the server should be done separately from the main thread to not interrupt the user experience.

Once a connection has been established, both devices may write bytes to the output stream of their respective socket. The bluetooth protocol ensures that all data sent this way arrives intact at the destination device, and that the packages arrive in order. [2]

Finally, as the bluetooth API does not directly handle the data sent over the Bluetooth link, the developer needs to define their own communications protocol in terms of what messages can be sent and how these are handled at the receiver. The purpose of the bluetooth sockets API is only to ensure the transmission and intactness of the data, and to do nothing further.

B. Public Key Cryptography

The confidentiality goal for secure systems requires that all messages sent must only be readable by the intended recipient and no one else. To address this problem, encryption is commonly used. An encryption function will take clear, human-readable text as input and produce encrypted text (also known as ciphertext), which is not human-readable. The ciphertext will remain non-readable until the associated decryption function is applied to it, meaning that it is safe to transfer the ciphertext to the recipient.

For this to work, the encryption and decryption functions needs to be only applicable by the sender and the recipient, and no one else. This means that they have to share some piece of information that no one else knows about. In cryptography, this information is known as the *key*. Each communication channel has one or more associated keys that the cryptography functions use to operate. [3]

1) Symmetric / Asymmetric Cryptography: The two basic types of cryptographic algorithms are:

- Symmetric algorithms (also known as secret key algorithms). These algorithms use the same key for both encryption and decryption. Secure communication requires one secret key for each pair of users. See figure 1 for a visual representation of the communication flow.
- Asymmetric algorithms (also known as public key algorithms). These algorithms use two different keys for encryption and decryption respectively. Secure communication requires two keys per user on the network. See figure 2 for a visual representation of the communication flow.

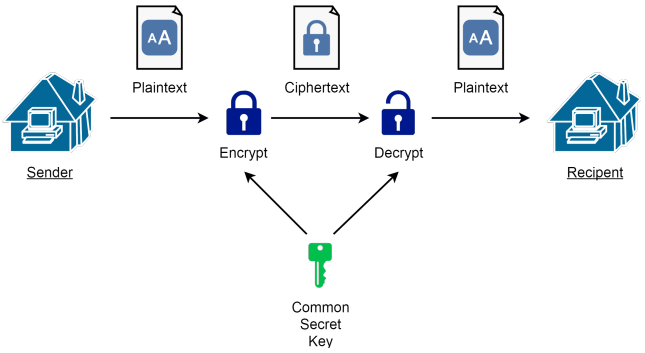

Fig. 1. Scheme of a secret key cryptosystem

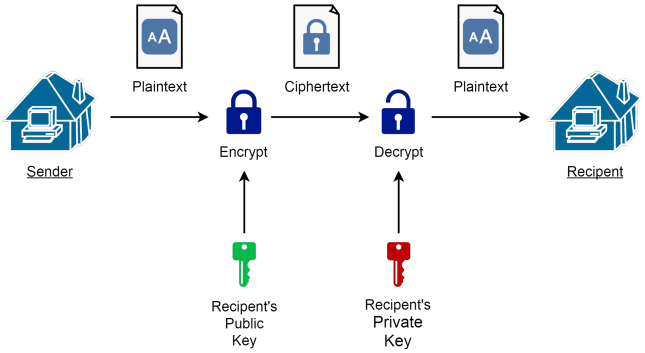

Fig. 2. Scheme of a public key cryptosystem

Symmetric and asymmetric algorithms both pose the same major challenges when it comes to distribution and management of keys. One needs to safely convey keys to those who want to establish a communication channel, and do so without providing vital keys to anyone else. One also needs to have the keys available as soon as needed. [3]

2) The Key Distribution Problem: Symmetric and asymmetric ciphers both suffer from the key distribution problem. If any of the decryption keys were compromised by a third-party entity on the network, and that entity was not already included in the encrypted channel, they may also decrypt any ciphered text along that channel. They may now listen to the conversations and, depending on the cipher used, send forged messages of their own. This violates the confidentiality and integrity goals in secure systems.

For public key cryptosystems this is less of an issue if there is no way to easily derive the private key from the public key. Because of this, a node on a network may freely distribute the public encryption key to anyone who wishes to send data to it over an encrypted channel. The public key is then used to produce an encrypted message, which the private decryption key can decipher into the original clear text. As long as each receiver on the network have their own private keys, and these are never compromised, the encrypted channel is considered secure. [4]

3) Stream Cipher / Block Cipher: An equally important distinction between type of encryption algorithms is that between stream ciphers and block ciphers.

- Stream ciphers: ciphers that take an input stream and converts symbol by symbol into ciphertext. Decryption is carried out in the same way. An example is the ROT-n cipher where each character is shifted alphabetically by n steps.
- Block ciphers: ciphers that encrypt a group of text symbols at a time. The group is known as a block, and is always of a fixed size (e.g. 1024 bits). Most modern symmetric encryption algorithms are block ciphers.

Both ciphers have their respective advantages. Stream encryption is faster and has low error propagation in that a single error only affects one symbol. Block encryption is slower and has higher error propagation, since an error in one symbol may corrupt the entire block. High error propagation may also be a valuable quality however, since it provides immunity against manual changes in the ciphertext, which will result in corrupted clear text which will not be handled by the receiver. [5]

4) The RSA Algorithm: RSA is a public key block cipher which is commonly used in secure systems today. The RSA documentation provides four properties which the encryption and decryption procedures must follow for public key cryptography:

- Encrypting a message and then decrypting it must yield the message.
- Decrypting a message and then encrypting it must yield the message.
- Publicly revealing the encryption key must not reveal an easy way to compute the decryption key.
- New public and private keys must be easy and fast to generate.

Note that the first and second properties imply that the decryption method is truly the inverse of the encryption method, as long as the proper keys are provided.

In public-key cryptography, the encryption algorithm uses a public/private key pair. In RSA specifically, these are two large (100+ digit) prime numbers, which are always generated by the same function. The message to be encrypted is represented as a large integer through some mapping function.

The encryption function involves the message and the public key integers, and outputs a new integer which is the encrypted message. The decryption function uses the private key integer, and operates on the encrypted message integer, and produces the original message integer, which may once again be represented as clear text. Decryption will only work if the private key is available. This means that if only the receiver has access to it, it will be nearly impossible for anyone else to read the original message. [4]

The size of the RSA algorithm determines how large the keys that will be used are, which in turn determines how large the message blocks can be. Typical keys sizes are 1024 to 4096 bits (meaning we can choose prime numbers among the first 2^{1024} to 2^{4096} positive integers, and the block size in either case is 1024 bits (128 bytes) or 4096 bits (512 bytes)). [6]

Since RSA is a block cipher, larger messages have to be broken up into blocks of a fixed size, for instance 128 bytes for RSA 1024-encryption. One would then send message packets in chunks of 128 bytes to the receiver, directly after successfully encrypting. In this manner, even a block cipher can be made to encrypt messages of an arbitrary length. This means, however, that the trailing block will likewise be 128 bytes in size, even if the actual text is shorter. [7]

The security of RSA depends on the fact that it is difficult to factor large prime numbers, even with the fastest known factoring algorithms. Even using a brute-force attack to guess the keys would be far less efficient, as it would take an average of 2^{1023} decryption attempts to correctly guess a user's private key for RSA 1024, which is far too slow for any modern computer. As such, RSA is currently considered secure. [4]

C. Digital signatures

The integrity goal in secure systems is addressed through what is known as *digital signatures*, which are sent along with the data in each transmission. A digital signature is simply the result of the sender applying their RSA decryption function to the original message once. Since the sender's encryption function is the true inverse of their decryption function, the receiver (who has the sender's public key) can encrypt the signature to receive the original message, and verify that it is the same as the actual message. Since the encryption procedure requires that the correct encryption key is used, successful verification of the signature will mean that the message was truly sent by the sender, who is presumed to be the only one who has access to their own decryption key.

In most practical applications, the signature is not the message itself but some hash that is shorter in length, to save computation time. [4]

D. Certificate Verification

[8] explains that, for the purpose of authentication, a user's public key should be sent inside what is known as a *certificate*. A certificate is simply a document which contains the public key, along with information on the identity of a user, as well as sometimes their location or department within a company.

Certificates are always issued by a *certificate authority* (CA), which is simply some higher instance on the same network, which is deemed trustworthy. All certificates that are issued by the CA are signed using the CA's own private key. If the CA's public key is embedded into the end application, every user of the application may verify another user's certificate to determine if they may be trusted, and begin sending encrypted data to them on the network. More specifically: successful verification of a certificate means the user can trust that the certificate was truly issued by the CA, and that the CA trusts whoever owns the associated public key.

In such a system, the CA has knowledge of all users that have previously subscribed to the application. The CA may also revoke certificates by maintaining a list of untrustworthy sources. Once an end-user wants to verify the credibility of another user, they may check their identity against the certificate revocation list (CRL). If the serial number of the other user's certificate has been previously revoked, that user is deemed untrustworthy and is excluded from any further communication.

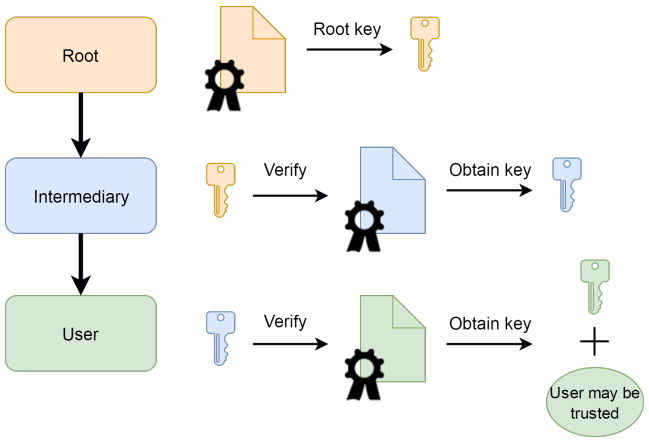

Fig. 3. Visualization of the chain of trust, and how keys may be obtained from certificates.

In larger networks, it may be necessary to decentralize the process of certificate signing, to enable client companies or third-party applications to use the trust of a company's CA. In such situations the top-level CA owns a root certificate, which is used to sign certificates to other CAs on the network. The so-called *chain of trust* is a list of certificates or serial numbers that have to be verfied in a certain order to access a user's certificate and key, (starting with the root key) and thus one may verify that the owner of the bottom certificate is to be trusted. Many software companies, such as Microsoft, Apple and Mozilla, have their own root certificate programs, which users and other developers may subscribe to. See figure 3 for a representation of the chain of trust. [8]

III. Results

A. Generation of Credentials

The public- and private key pairs where generated in 1024 bit RSA using OpenSSL for ubuntu. Certificates and CA credentials were also generated using OpenSSL. Certificates were generated in the x509 format, and signed by an intermediary CA below the root CA.

B. Program Overview

Figure 4 shows a screen capture of the user interface of the final application. Referring to the same figure, the app has the following feature set:

1) Pressing the "Start BT Sockets Server"-button at the top left of the screen enables broadcasting of data from the device. The device now acts as a bluetooth server, and any other client running the app may connect to it.
2) The user may navigate the map by swiping to move and pinching to zoom in and out. Pressing and holding anywhere on the map will create a request for POIs on that global position, and send that request to the server device.
3) At the very bottom of the user interface are query parameters
   - Radius - How far (in meters) from the selected position the user is interested in POIs.

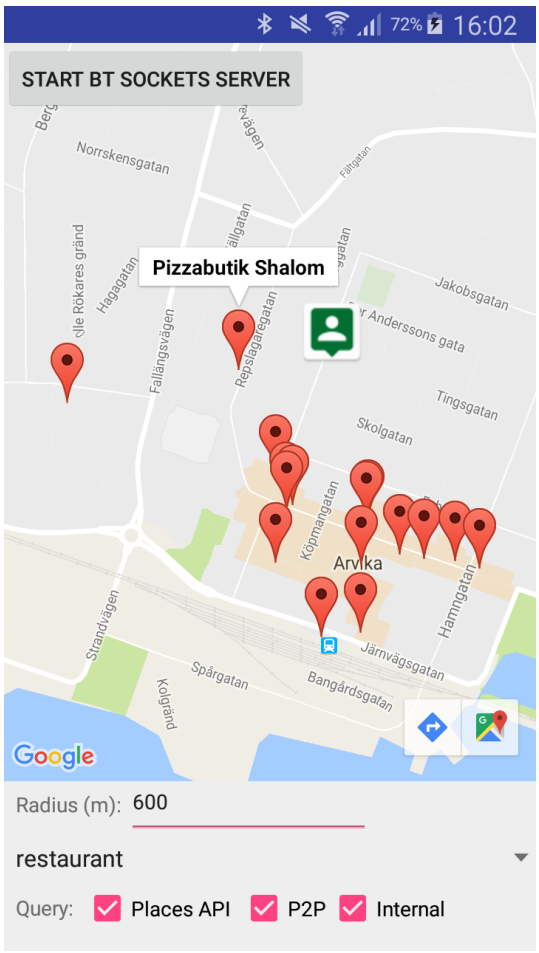

Fig. 4. A screen capture showing the final application as it runs on a Samsung Galaxy Alpha.

   - Query type - A drop-down menu where the user may select what type of POIs they are interested in. The possible types are specified in the Google Places API, and may be for instance restaurant, lodging or spa.
   - Source check boxes: Places API, P2P, Internal - Specifies from which sources POIs are to be obtained once the user presses the map.

The devices will have to have bluetooth enabled. Otherwise a prompt will appear, allowing the user to enable bluetooth from inside the app. From here, the devices need to be paired before any communication can occur. The user may pair the devices from the bluetooth interface on their android device.

C. Communication Flow

The communication sequence is illustrated as a sequence diagram in figure 5. The contents of the messages that are sent is represented in figure 6.

IV. Discussion

A. How Does the Implementation Solve The Problem?

1) Authentication: The authentication requirement is effectively handled by the certificate verification system. The CA

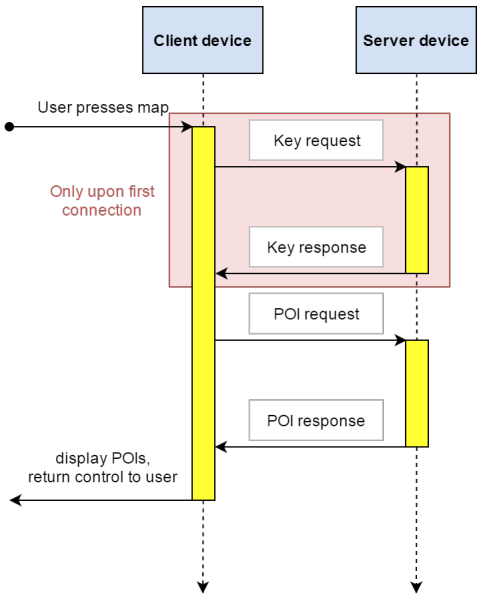

Fig. 5. A sequence diagram that details the client-server relationship, and in what order the messages are sent once the user requests information by pressing the map.

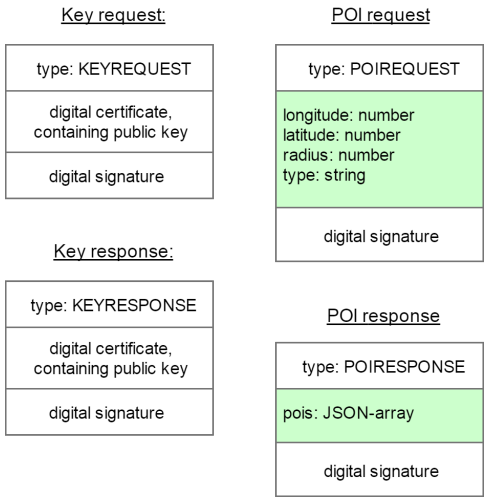

Fig. 6. A detail schema of the messages. The fields marked in green are encrypted using RSA 1024.

has knowledge of all nodes that has subscribed to the network, and carries on this knowledge to the nodes through issuing signed certificates. These certificates are verified by the client running the app, who decides that another user is trustworthy upon successful verification.

As of now however, the certificates have been created and signed manually using a console, and then manually transferred to the device independently of installing the application. This will be too inefficient in an environment where the application is sold on the market and new users will appear on the network on the fly. However little change would have to be needed to the app in order to tie in an automated certificate signing program (using certificate signing requests to the CA) which allows for greater scalability.

Also, on the decentralized level, the digital signature is used as a mean of verifying the identity of the sender, since a valid signature requires that the message originated from the owner of the private key with which it was signed.

2) Integrity: The integrity requirement is firstly fulfilled by the strong message policy that dictates the format of the messages (JSON structure), what message types are allowed and what key-value pairs are expected to be found in each type. The internal handler will only operate on these specified values, and if anything were to be changed so that the handler cannot perform its task correctly, the app will exit out of the message handler and drop the incoming message before doing anything further.

For the same purpose, the messages are encrypted. This means that any direct changes to the ciphertext will result in clear text that is meaningless to the message handler in that it does not match the JSON data format.

Even if some modification were to be made to the ciphertext that results in meaningful clear text, the digital signature still has to match the original message, which is only possible if no change were made to the message between sending and receiving.

The only integrity concern is that the RSA private key may be compromised or cracked, which may very well be a concern since as of this writing moment, the key pairs are stored in a human-readable format in the devices memory. It would be better for future applications if the keys were stored in some serialized or encrypted format, or through the use of session-based keys, so that the user does not have direct access.

3) Confidentiality: The confidentiality requirement is largely solved by the usage of a public-key cipher. This means that nodes on the network will only be able to receive from other nodes that have obtained their own public key.

The RSA cryptosystem further ensures that no one except the intended receiver (who has the correct private key) may read the contents of the messages sent. In the event that a message is obtained by a node that is excluded from communication, they will only be able to read the cipher text, which is meaningless without knowledge of the correct decryption key.

Furthermore, the public-key infrastructure allows for public keys to be distributed exclusively to other clients running the app, as long as they have the CA root public key stored internally, and may verify the other user.

Again, the most topical system weakness is that the cipher keys may be compromised.

B. Design Topics to Improve Upon

1) Security: In an environment where the application is freely available for download, it may not be practical to manually generate one certificate per device using a console. The application could instead send a certificate signing request (CSR) to the CA, using some hash that is generated from a user id and some time stamp. The CA may use the hash to determine if the user is legitimately using the application or not, since no one else would have knowledge of what hash is used internally.

The use of session keys and session certificates with short expiration times may further improve security against malicious nodes, since they have a limited time to exploit a key once they obtain them. Though security by this design should not be taken for granted.

Finally there is the concern of the speed of RSA encryption, since RSA 1024 is very slow for encryption large chunks of data. A better procedure for large sets of POIs would have been to use a faster symmetric cipher such as AES to encrypt the POI data, and in turn encrypt the AES key using RSA 1024. Both the AES-encrypted data and the RSA-encrypted secret key would have then been transferred over bluetooth upon a POI request.

2) Bluetooth/Wireless Technology: As the application works now, the devices have to be paired manually before any of the users start making requests. This is indeed a bottleneck in terms of user friendliness and speed, since it requires the user to exit out of the app and enter the android bluetooth interface to scan for other devices and connect to them. Connection management and device pairing could instead be done internally from within the application, given that the android bluetooth API allows for this. This would be a topic of future research.

Furthermore, bluetooth is relatively high on power consumption. For a future project one may look into using the Bluetooth Low Energy (BLE) protocol for power saving purposes. BLE does not use sockets like the Bluetooth Sockets API, but rather acts as a server-client relationship where the server broadcasts data with set intervals or upon request. The transmitter is turned off at all other times, which limits energy usage. As for all bluetooth technologies, they has a standard range of approximately 100 meters, which limit their usability to cases where users of the application are located in each others vicinity. For greater data availability, one may choose to use TCP over wi-fi or 3G/4G. One may then have a web-based automated tracker which provides clients with a start-up list of peers. This may imply sacrificing decentralization for availability, but it still solves the problem of limiting the exposure to the provider of the location-based service. Even still, the tracker would only manage connections, and have no insight in what data is actually shared on the peer-to-peer network, which promotes privacy.

3) Internal Storage: One of the clearer issues with using the application is that of the time it takes to obtain POIs from the internal storage, which affects the speed of delivery for both internal queries as well as when the other device is queried over bluetooth. One reasonable solution would be to try implementing some existing database technology (for instance MongoDB or SQL technologies like SQLite), rather than storing POIs in an array in JSON format. This would in theory provide quicker access, as technologies such as these optimize searching through the use of hash tables and indexes.

V. Conclusion

This report presented the final version of a student-programmed android application which sends data in a secure manner. The fundamentals of secure systems was presented, with its cornerstones being authentication, integrity and confidentiality. In the application, these goals were met primarily through the use of certificate validation, encryption and digital signatures respectively. Even though the application would be considered secure in an environment where only two android devices are communicating, there are more steps to be taken if the application is to be deployed commercially. The primary concerns for future development are scalability, speed and management of encryption keys.

Acknowledgements

The author would like to thank their supervisors Hongyu Jin and Mohammad Khodaei for having frequent discussions on how to organize the project, as well as program and optimize a secure application. Special thanks also goes to professor Panos Papadimitratos for teaching the fundamentals of secure systems, and fellow KTH student Peter Caprioli for suggesting a design for the peer-to-peer communication system.

References

[1] H. Jin, M. Khodaei, and P. Papadimitratos, "Security and privacy in vehicular social networks," in *Vehicular Social Networks*. Taylor & Francis Group, 2016.

[2] Android API Reference. Bluetoothsocket. [Online]. Available: https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html.

[3] B. Young. Foundations of computer security, lecture 44: Symmetric vs. asymmetric encryption. [Online]. Available: https://www.cs.utexas.edu/users/byoung/cs361/lecture44.pdf.

[4] Virginia Tech. Cryptography: Rsa algorithm. [Online]. Available: http://courses.cs.vt.edu/∼cs5204/fall00/protection/rsa.html.

[5] B. Young. Foundations of computer security, lecture 45: Stream and block encryption. [Online]. Available: https://www.cs.utexas.edu/users/byoung/cs361/lecture45.pdf.

[6] IBM Knowledge Center. (2014). Size considerations for public and private keys. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.icha700/keysizec.htm.

[7] F. Westreicher. (2010, Jun). Encrypting and decrypting large data using java and rsa. [Online]. Available: http://coding.westreicher.org/?p=23m.

[8] M. Bruce. (2010, Aug). Chain certificates. [Online]. Available: https://www.entrust.com/chain-certificates/.