

	<p>Afonso Gomes 92410</p>		<p>Miguel Henriques 102148</p>		<p>António Martins 102280</p>
---	-------------------------------	---	------------------------------------	---	-----------------------------------

Contact Tracing

Solution Report

1. Problem

Contact Tracing systems can be very useful in controlling the spread of a virus during a pandemic, however, due to the nature of the data these systems deal with, they can also constitute a great privacy problem for users. There are two main pieces of sensitive information of users that a contact tracing system deals with and stores in some way: a user's infection status and a user's contacts (which may include the location and time of those contacts). With our solution we intend on reducing as much as possible the ways of attacking the system and unravelling sensitive information of users.

1.1. Solution Requirements

The following list includes all the security requirements for our solution, and additionally it also contains a short description of the solution's mechanism that addresses it (in parenthesis).

- R1.** Network traffic must be reliable, secure, and encrypted. (TLS)
- R2.** A user must not be able to falsely claim that they have been infected. (Infection Claim Codes)
- R3.** When a user claims to be infected, no sensitive data should be sent to the central server. (Overall design of the system)
- R4.** A network observer must not be able to learn that a person is infected by the simple existence of a message. (Dummy messages sent at random intervals)
- R5.** Local sensitive data (mainly location information) must be stored encrypted. (Encryption with public key and obfuscation of private key through a password)
- R6.** A user must not be able to use a message received by another user to impersonate that user when claiming to be infected to the central server. (Sending MIDs to nearby devices instead of SKs)
- R7.** The central server should be resistant to simple DoS attacks. (Firewall configuration)

1.2. Trust assumptions

The health authority will be fully trusted to only provide one-time codes for users to claim to be infected when appropriate.

The users will be untrusted, as users can easily be attackers, and so can't be trusted to always act correctly and morally.

The central server will be partially trusted and thus will include as little personal information from users as possible, as a security breach of some kind could pose a great threat to the user's privacy otherwise.

The system administrators will be fully trusted not to try to correlate data received to the specific users that sent it. They will also be fully trusted not to insert fake data in the system.

The external systems and libraries used, like gRPC, PostgreSQL and SQLite JDBC will be partially trusted, as they'll be trusted to have as few vulnerabilities as possible (as it is almost impossible to have none), and to quickly release patches for discovered vulnerabilities.

2. Proposed solution

2.1. Overview

The system will include the mobile app used by the users and a central server.

When the app is executed for the first time, it will generate a random Secret Key (SK), associated with the user. The SK will be used to deterministically generate different Message IDs (MID), throughout time. The app will then run in the background of the device, periodically sending Bluetooth Low Energy (BLE) messages, which other users' apps will use to store contact between each user. When a message is received, the app stores the acquired MID, and the current time and location.

When a user becomes infected with the virus, the health authorities must give them a one-time Infection Claim Code (ICC) generated by the server, that they must enter in the app, to prevent fake reports. After entering the ICC, the app sends it to the central server along with the oldest SK that the app has stored (which will usually be the SK from 14 days ago). The server will then store this SK as infected.

Finally, users will then query the server periodically for the infected list, retrieve the SKs, and compare their stored MIDs with those generated using the infected SKs.

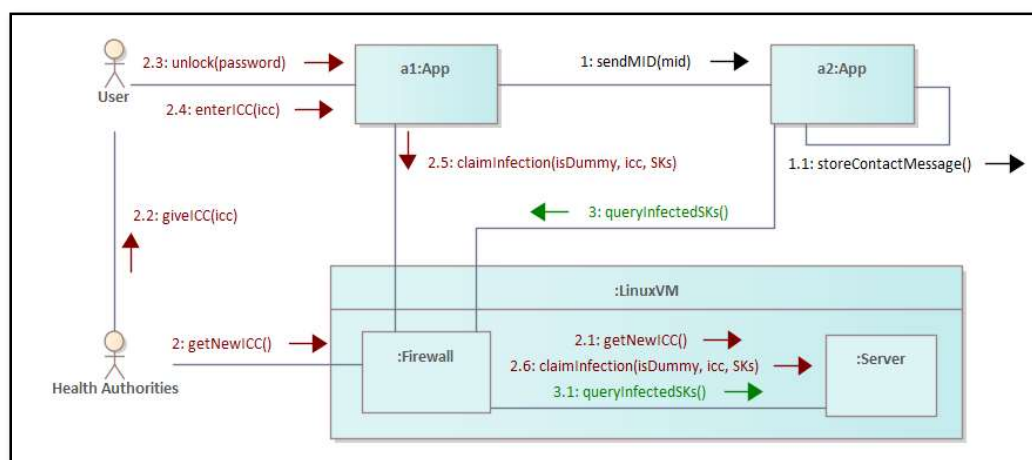


Diagram 1 - Communication diagram of the system's interactions.

2.2. Deployment

The whole system will include the Linux machine where the Server runs on, and the Android mobile devices of the users where the App will run on.

Diagram 1 describes all the interactions between these machines.

2.3. Secure channel(s) to configure

The app installed in the users' devices will communicate with the server through TLS.

The library/tool we'll use for TLS is gRPC, which has "SSL/TLS integration".

For this communication, no keys will be distributed apart from the ones distributed by the TLS implementation.

2.4. Secure protocol(s) to develop

Every user's app will generate a public-private key pair during the initial setup and ask the user to create a password. The app will also generate and store two "salts", one for the verification of the password and one for the obfuscation of the private key.

While the public key will be stored in plain text, the private key will be securely stored through obfuscation.

The private key will be obfuscated and de-obfuscated through the following operations:

$$\text{ObfPrivKey} = (\text{PrivKey} \oplus \text{Hash}(\text{Pwd} + \text{Salt}))$$

$$\text{PrivKey} = (\text{ObfPrivKey} \oplus \text{Hash}(\text{Pwd} + \text{Salt}))$$

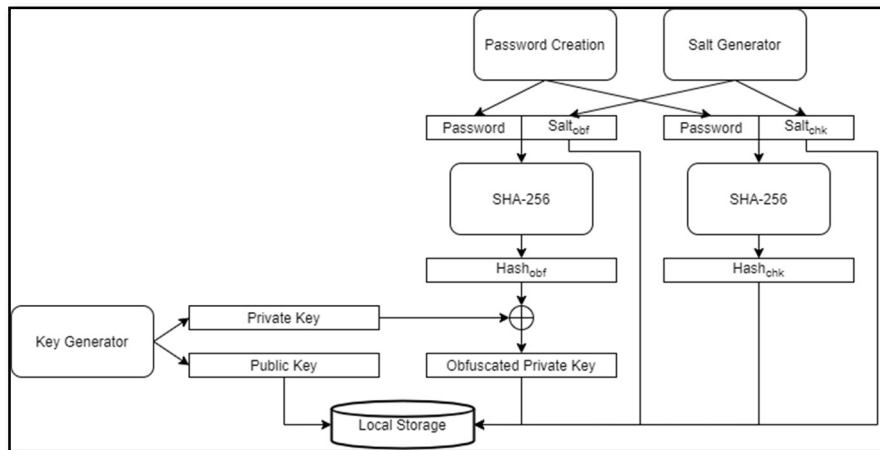


Diagram 2 - Diagram of the initial setup of the protocol.

Upon receiving a contact message, the public key will be used to immediately encrypt and store the information regarding the GPS location of this contact.

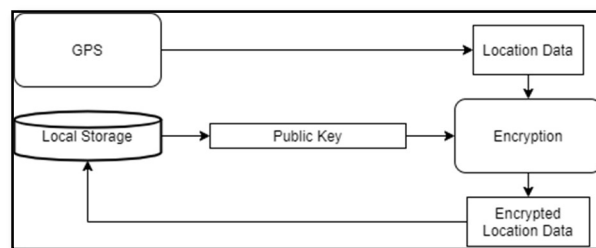


Diagram 3 - Diagram of the process of encrypting location data.

When the app notifies the user of a risk of infection, it also prompts the user for his password, in order to de-obfuscate the private key and decrypt the information regarding the locations where there was contact with an infected person.

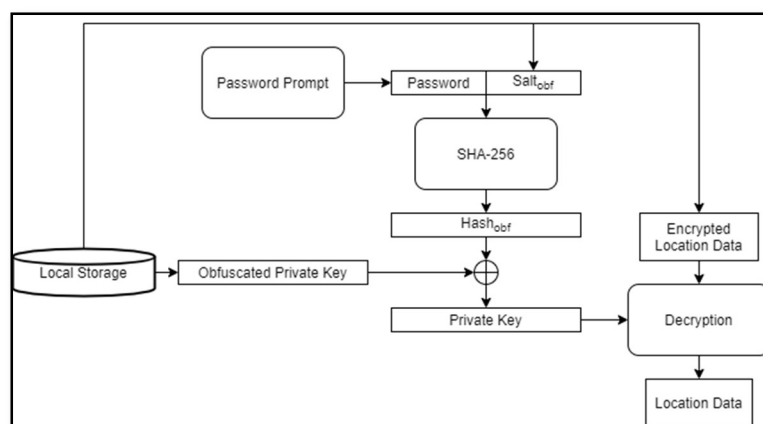


Diagram 4 - Diagram of the process of decrypting location data.

3. Used technologies

Programming Language:	Java 11
Communication between App and the Server:	gRPC with SSL/TLS.
Communication between Apps:	Bluetooth Low Energy (BLE)
Mobile app local storage:	SQLite JDBC.
Server-side storage:	PostgreSQL 14
Building:	Gradle
Firewall Utility:	Iptables
Tools for development:	Android Studio, IntelliJ IDEA

4. Results

All these previously mentioned security requirements were satisfied:

- R1. Through the use of TLS in the communication between the Apps and the Hub.
- R2. Through the requirement of a single-use code, issued by health authorities, in order to claim an infection.
- R3. Satisfied by the overall design of the system.
- R4. Prevented by periodically sending fake dummy infection claims.
- R5. Through the use of asymmetric cryptography, where the private key is obfuscated by a password chosen by the user.
- R6. Satisfied by the fact that what is sent to other user's apps is a message that is generated through a user's secret key, and the fact that what is sent to the Hub when claiming an infection is the user's secret key.

The only security requirement that wasn't fully satisfied was the requirement R7, which was still partially satisfied, through the use of a simple firewall configuration.

5. References

- <https://ncase.me/contact-tracing/>
- <https://github.com/DP-3T/documents/blob/master/Security%20analysis/Privacy%20and%20Security%20Attacks%20on%20Digital%20Proximity%20Tracing%20Systems.pdf>
- <https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf>
- <https://grpc.io/docs/guides/auth/>