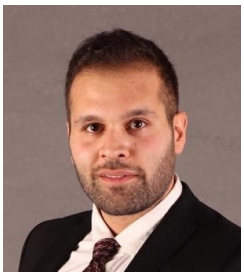
# Smartphone as a Security Token

## Seamless Multi Factor Authentication

Network and Computer Security
Alameda
Group 28


Afonso Gonçalves
89399


Emil Njor
98073



Farzad Terhanian
98425


January 13, 2022

# 1 Problem

When multiple people require access to the same gate, e.g. to get into the office building of a company, it is often desirable to not distribute physical keys to everyone. The standard solution for this problem today is to use a shared key/PIN which is entered at a keypad near the gate to get access. Another solution is to use smart cards. These systems have some major flaws: It is very easy to find the PIN and gain access to the building or to clone/steam cards.

A more secure alternative is to use biometric data to control access. These alternatives are however more expensive than regular keypads. Furthermore, biometric data cannot be fully relied on since it can get compromised and it is expensive to replace due to its scarcity. Fingerprint scanners have the additional drawback that everyone is touching the same surface, which can spread diseases.

Multi-factor authentication improves security by adding extra authentication layers. As such we do not have to rely on just one method of authentication. However, increasing the authentication steps often makes life harder for the users. This results in users slacking on the security, which presents new issues. Therefore, it is important to consider the amount of interaction required when implementing new security systems.

The core of the problem is to only allow selected people to pass through a gate. We can do this by using physical keys, PINs, biometric data, etc, but they all have security flaws when used alone.

## 1.1 Requirements

In order to implement this secure system, the following requirements should be considered:

**Security Requirements:**

- The user should be authenticated using more than one factor of authentication;
- The protocols used to authenticate a user must ensure integrity, authenticity and freshness.

**Non-Security Requirements:**

- The system should not require more interaction from the user than e.g. using a physical key.

- The system should be cheaper to implement than having biometric sensors installed at gates.
- The system should not have a shared surface where diseases can be spread.

### 1.2 Trust assumptions

The developed solution should fully trust the gate, and the devices used by the gate to read requests. It also assumes that the implementation is correct based on the design, and that eventual private keys are kept private. Furthermore, initial configuration must be right, and biometric authentication devices are assumed to be reliable. We also trust the users do not try to unlock the gate while not in close proximity with the gate.

Whatever authentication factors we use in the solution will be partially trusted, which combined should create a reliable service. Eventual communication channels will not be trusted, and it is assumed that an attacker can do whatever it wants with messages sent through the channel.

## 2 Proposed solution

### 2.1 Overview

The core of our solution is to use asymmetric keys to establish authentication. The private key would be stored in the user's smartphone and unlocked with the phone's fingerprint scanner. This allows us to both save costs on fingerprint scanners, and to have no shared surfaces or keys. The communication will rely on Bluetooth technology, ensuring that the user is near the gate. The only interaction required is when unlocking the private key with the user's fingerprint.

The resulting security depends on three authentication factors [AZE09]:

- *"Something you have"*: The solution requires the users to carry their smartphones to gain access through the gate.
- *"Something you are"*: The user's fingerprint is unique and very difficult to replicate.
- *"Somewhere you are"*: The Bluetooth's low range ensures that the user is near the gate while authenticating.

The gate is opened after the client authenticates himself with his private key, as shown in Figure 1. Since the gate only opens when all these factors are met, this solution provides Multi-Factor Authentication.

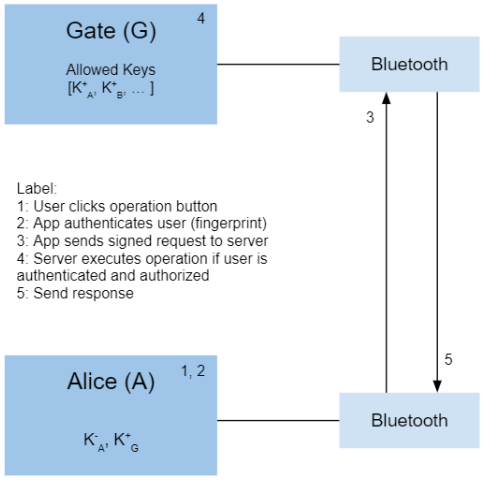



Figure 1: Interaction overview

## 2.2 Deployment

The solution will consist of a mobile application that connects to the gate over Bluetooth. The gate is controlled by a *Raspberry Pi*, which is listening for incoming requests.

## 2.3 Secure channel(s) to configure

Since our project focuses on developing a secure channel over Bluetooth, its configuration is explained in section 2.4.

## 2.4 Secure protocol(s) to develop

**Authentication** is the key security property of this project: Without it, the gate loses its purpose. Consequently, **Integrity** and **Freshness** are required. Confidentiality will not be focused on since there is no sensitive information in the communications.

The protocol to execute an arbitrary action (*e.g.* open the gate or register a new user) is described as follows and as depicted in Figure 1:

1. The user ($U$) gets close to the gate ($G$) and requests the android application ($A$) to perform an operation ($O$);
2. $A$ authenticates $U$'s fingerprint. If it fails the operation is not executed;

3. $A$ requests $G$ to perform the operation. The request message contains the ID of $U$ ($U_{ID}$), $O$, a nonce ($N_U$) and a timestamp ($T_U$). The entire request is digitally signed using $U$'s private key.
4. $G$ checks that $U$ is a registered user and authenticates him by checking the digital signature; It also confirms the uniqueness of the received $N_U$ and that $T_U$ satisfies the defined threshold (5 seconds in this implementation). If any check fails, the operation aborts;
5. $G$ checks that $U$ has permission to execute the requested operation.
6. $G$ executes the requested operation and sends back a response to the application, indicating the status of the operation to the application.

The main protocol depends on a correct public key distribution. The proposed solution assumes that there are administrator users that are allowed to add public keys to the gate's white list. It is assumed that the public keys are manually and correctly delivered to the administrator.

# 3 Results

The solution was implemented as two separate projects, one for the android platform to act as the client sending authorization requests, and the other developed for a *Rasbperry Pi* to act as the server receiving authorization requests. We will now describe implementation details in these two projects.

## 3.1 Android Client

This project is developed using the Android SDK, which contains its own support for both fingerprint authentication and Bluetooth connectivity by interacting with the underlying operating system. The project requires an android SDK version of at least 26, and requires a phone with access to biometric authentication, which may be a limitation of the solution, as this requires a newer phone.

The digital signature is a *SHA-256* hash of the request encrypted using *RSA* with the private key of the user to ensure the integrity and authenticity of the message.

## 3.2 Raspberry Pi Server

The server starts a Bluetooth server, and listens for incoming requests.

The known public keys are defined dynamically: The server saves a file that contains a serialized Map including the pairs <*user, publicKey*>. This map can be updated if an administrator sends a *register* request, containing the new username and the corresponding Public Key as arguments.

The user permissions are set via a white-list JSON file. This file is read when the server starts and results in an association between each allowed command and the set of users that are allowed to perform it.

### 3.3 Strengths and Weaknesses

In this section we will discuss the strengths and weaknesses of our developed solution.

1. The solution allows for authentication and authorization in $< 4$ seconds.
2. The solution meets the defined requirements.
3. Key distribution is manual, which may impact large scale applications. However access permissions to e.g. a building do not change often, therefore manual distribution is sufficient and allows supervised access control;
4. The current system only supports a single gate: An attacker could perform a replay attack if a replayed message is sent to a different gate that hasn't received the client's Nonce. There may also be the case that an attacker fakes a gate and a victim sends a request to this gate. This would allow the attacker to replay this received message to any other gate. We can overcome this problem by having gate authentication and specifying the target gate in the sent requests. This could be done by having legitimate gates' public keys certified by the responsible company (BlueGate).
5. The server stores every Nonce it received so it can detect a replay attack. It also checks if the received message is fresh (if it is more recent than the established threshold). This may lead to an infinite amount of Nonces being stored, affecting this project scalability. This could be overcame by regularly deleting Nonces that were sent before $T$ seconds ago, being $T$ the defined threshold for timestamp acceptance. In this case a replayed message would never be accepted since it would have a duplicate $N_U$ in the specified time frame or the timestamp would not be fresh enough to be accepted.

## References

[AZE09] F. Aloul, S. Zahidi, and W. El-Hajj. "Two factor authentication using mobile phones". In: *2009 IEEE/ACS International Conference on Computer Systems and Applications*. 2009, pp. 641–644. DOI: 10.1109/AICCSA.2009.5069395.