Smartphone as a security token - Group A14

Network and Computer Security 2021/2022

Bernardo Quinteiro	Diogo Lopes	Jérémy Breton
93692	93700	101360

1. Problem

The topic we've chosen for this project is "Smartphone as a security token". As a result of that, we've decided that we'll be developing a virtual wallet with two internal servers. The first one will be responsible for the critical system of the wallet, such as the login phase, checking the wallet's balance, depositing money and being able to transfer to other users. The other server will be responsible for managing the communication with the exterior (web application and the smartphone) containing a firewall to prevent attacks.

The main security issue we need to address is the possibility of an unwanted entity gaining access to the critical server of the wallet. We must fully guarantee that attackers can't gain access to this system, by any means. Other than that, the information that is passed between the internal servers and the external entities needs to be confidential and keep its integrity, so these connections have to be secure.

1.1. Solution Requirements

R1: Confidentiality: The data passed between the server and the client, in the web application and in the smartphone, must be encrypted.

R2: Integrity: The data passed between the server and the client, in the web application and in the smartphone, can't be tampered with and should be preserved.

R3: Authentication: To connect to the virtual wallet server the user must perform two-factor authentication, a password and a token generated on the smartphone.

R4: Non-repudiation: The critical data passed between the server and the client, in the web application and in the smartphone, must be signed.

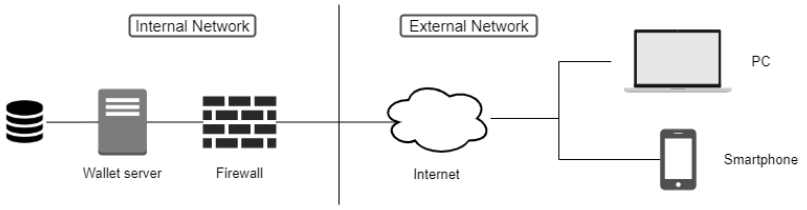
R5: Freshness: The token is randomly and securely generated, and is valid for a short time.

1.2. Trust Assumptions

The user must be logged in the website and can only access the virtual wallet when given the correct login credentials and token. This token is randomly generated every 30 seconds (in order to ensure freshness) by the server and securely stored in the user's smartphone. This way, an attacker will not be able to access the wallet even if he manages to get the user credentials, which ensures authentication.

2. Proposed Solution

2.1. Overview



This image represents a diagram of the pretended solution that consists of three elements. Firstly, we have a server that handles the operations, such as login and checking the virtual wallet balance. We also have a client that wants to do these operations and connects to the server via a web application and, finally, a smartphone that acts as a security token.

In order to be able to login to the web application, the client needs to perform a two-factor authentication, which are the login credentials and a randomly generated token that can only be accessible through the smartphone as a security manner. After logging in, the client has access to his virtual wallet where there are the possibilities to deposit money or send to other users.

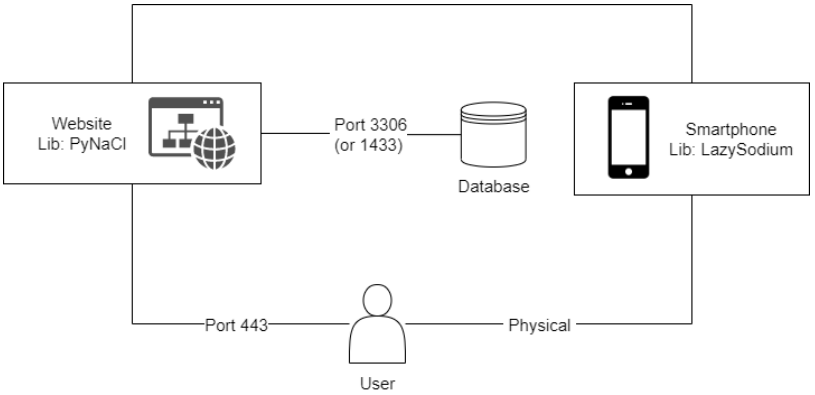
2.2. Deployment

We will have 3 virtual machines connected in an internal network. Firstly, there will be the wallet server machine which is responsible for serving the web application and performing the operations.

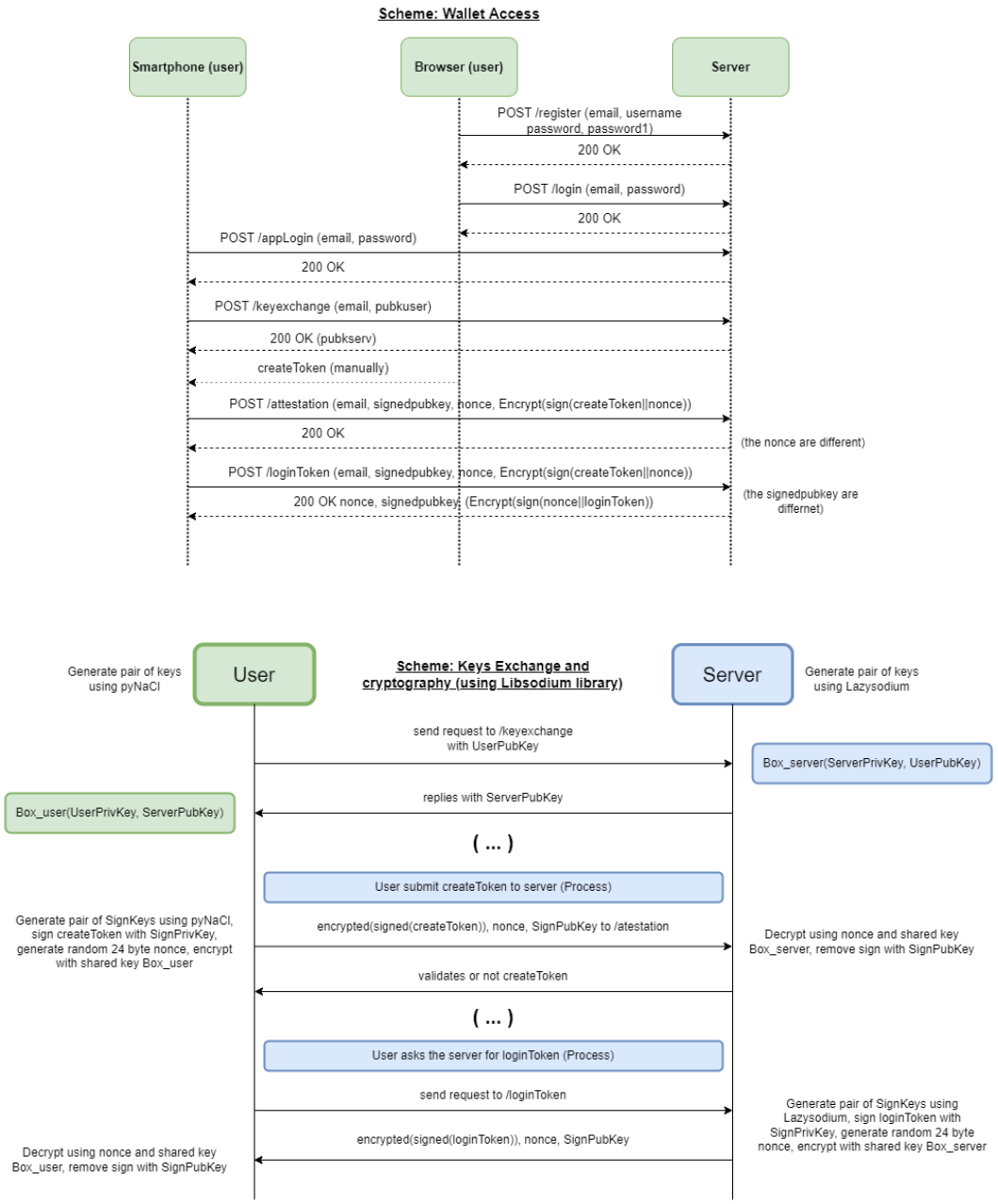
There will also be a machine acting as the user computer, where the user will be able to access the wallet web application.

Finally, there will be a virtual smartphone emulator (genymotion) that will act as the user's smartphone, where the user can login and access the temporary loginToken in order to authenticate in the web application.

2.3. Secure channel(s) to configure



2.4. Secure protocol(s) to develop



Who will communicate?

The wallet server will be able to communicate with a MySQL database in order to have access to the users characteristics. The information in this database, such as passwords will be hashed using SHA1.

There will also be encrypted HTTPS communications between the server and the user browser, on which the user will be able to securely login and access the virtual wallet functionalities. These secure communications will be ensured with OpenSSL by using SSL protocol.

The communications between the wallet server and the user smartphone will be secure HTTP requests, since every message will be signed and encrypted using the Libsodium library.

Which security properties will be protected?

Every message sent between the smartphone application and the wallet server will be signed and encrypted using unique key pairs. This way, the sender will be known, which guarantees authentication, non-repudiation and authenticity, and that the message was not altered in transit, which ensures integrity.

What keys will exist and how will they be distributed?

Firstly, the smartphone application and the wallet server will each generate unique key pairs and the public keys will be exchanged. This way, each side can create a Box with the respective private keys and the exchanged public keys. This box will be the same on both sides and acts as a shared key which can be used with a nonce to encrypt messages and decrypt given ciphertexts. This process is similar to what happens in Diffie-Helman but using the Libsodium library.

Afterwards, every time a message is going to be sent, it will be signed with a SignPrivKey and then encrypted with the before generated Box using a randomly generated 24 byte nonce. Then, the encrypted(signed(message)), nonce and SignPubKey will be sent and if the receiver can decrypt and remove the sign means the communication was secure and ensures the properties mentioned before. This exchange and cryptography is better explained in the pictures shown in the beginning of this section.

3. Used technologies

The wallet server was created using Python and the web application using Python Flask that can render HTML and CSS pages. Furthermore, there were also used some Flask libraries, like SQLALchemy in order to communicate easier with the database and PyNaCl that was used for the cryptography related part of the python server.

We used XAMPP, which is a software distribution which provides the MySQL server that we used for the database.

The smartphone application was developed in Android Studio, using Android and Java. We also used libraries to facilitate the connection of the smartphone application to the wallet server and LazySodium for the cryptography related part of the smartphone application.

4. Results

4.1. Requirement concretization

R1: Confidentiality: Satisfied, the wallet server and smartphone application generate each one a unique key pair, exchange public keys and obtain a shared key (Box, same methodology as Diffie-Helman) that is used to encrypt and decrypt the messages with a randomly generated 24 byte nonce. This way we can ensure confidentiality, because every connection between the wallet server and the smartphone application is encrypted.
R2: Integrity: Satisfied, the key exchange and further generated shared key guarantee that the message will securely arrive, hasn't been opened or tampered with. This way we can ensure integrity.
R3: Authentication: Satisfied, after the user login in the web application, the user will be prompted to introduce an authentication token that can only be obtained in the smartphone linked to the respective user. This two-factor authentication can ensure this property.
R4: Non-repudiation: Satisfied, every message sent between the wallet server and the smartphone application is signed. The sender generates a SignKeyPair, signs the message with the SignPrivKey and sends the SignPubKey to the receiver. If the receiver is able to remove the sign from the message we will have ensured non-repudiation.
R5: Freshness: Satisfied, the token needed to confirm the user authentication in the web application is randomly generated every 30 seconds for each user, this way we can guarantee freshness.

4.1. Main implementation choices

The connection between the smartphone application and the wallet server is secure HTTP requests since all the data is encrypted and signed as explained throughout the report. Then we had the goal of implementing HTTPS connections between the wallet server and the web application with OpenSSL, which we managed to do and everything worked fine in the web application but it would affect all the routes (including the ones the smartphone app accesses, which would receive HTTP requests). This way we were not being able to manage HTTP and HTTPS requests at the same time. We found a python flask library (flask-talisman) that can "transform" the HTTP requests to HTTPS or let us choose on which web application routes to force HTTPS but we didn't manage to fully implement it in the available time. Apart from this, we also implemented Cross-Site Request Forgery with a flask library (flask_wtf.csrf) but we couldn't make it work with the smartphone application.

In relation to the deployment on a live system, we managed to deploy the server and user virtual machines and connect them in an internal network, so the user could access the web application on his virtual environment with the python server being run on the other virtual machine. We decided to use Genymotion for the virtualization of the smartphone but we weren't able to connect it to the internal network.

We also did the iptable commands for a firewall to try prevent DDoS attacks but we weren't able to test it.

All this is explained and documented in the README file, even though we didn't manage to successfully implement it all in the available time.

5. References

- Git overview
- Git topics
- Android Studio
- XAMPP
- MySQL
- Python
- Flask
- PyNaCl
- Java
- Lazy Sodium
- Genymotion
- CSRF Protection
- SQLAlchemy
- Talisman
- Advanced HTTP URL
- Libsodium