Instituto Superior Técnico - UL

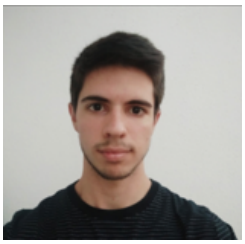
SIRS

Smartphone as a security token

Report - Group A24

Vasco Faria - 89559

Gonçalo Mateus - 93713



Guilherme Saraiva - 93717

---

28 of January of 2022

Contents

1 Problem

Nowadays, almost every system implements a Log in feature, but this may not be enough to ensure the authenticity of the user. Therefore, in this scenario, we have a web-based encrypted email system that allows users to communicate safely between each others.

The main problem that needs to be solved is the authentication, since attackers may try to overpass the login's step by performing several attacks like *bruteforce*, steal user passwords from other less secure systems, *phishing*, *keyloggers*, etc. Besides that, we have to ensure that every operation is performed by the authenticated user and every client-server communication is encrypted.

1.1 Solution Requirements

- **R1 - Confidentiality:** Data must be encrypted over the server-client communication.
- **R2 - Integrity:** Data must be preserved and not be tampered over the server-client communication.
  - **R2.1 - Non-repudiation:** Data must be signed over the server-client communication.
- **R3 - Freshness:** Messages are always sent with a fresh token.
- **R4 - Availability:** An authenticated user is always allowed to perform any operation.
- **R5 - Usability:** The application should be practical besides all the security features.

1.2 Trust assumptions

In this scenario, we assume that our server can partially trust the client. After pairing with the server, we know that the user's device is the only one allowed to connect to the server. So, whenever the user wants to perform an operation, the server should grant him the permission to execute it, while the session hasn't expired.

2 Proposed solution

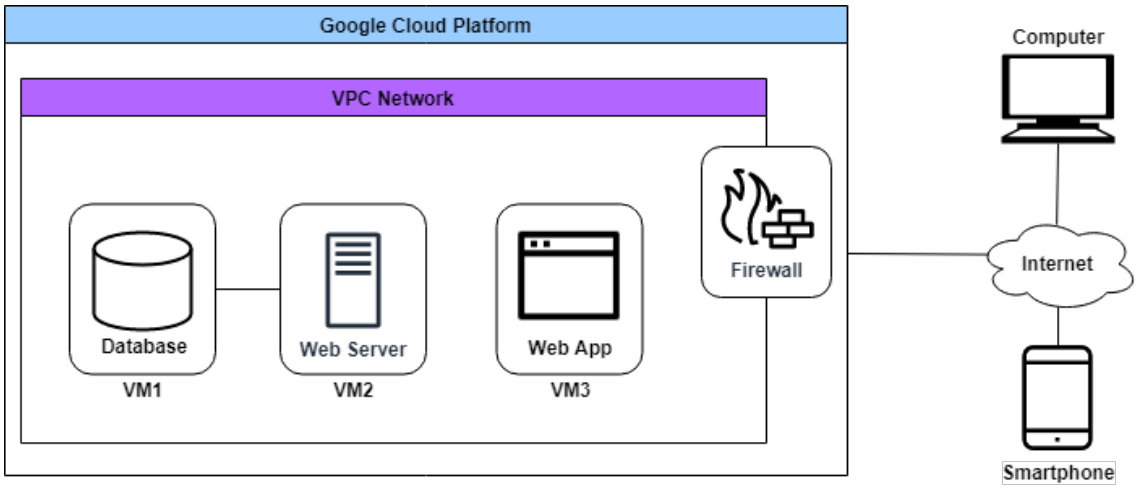


Figure 1: Network communication between devices.

2.1 Overview

Our solution consists of an internal network composed of three virtual machines. The first one has a database to store the necessary information. The second has a web server (running on port 443 that accepts HTTPS traffic). The third one is running the web application also on port 443. These virtual machines are connected in a VPC network configured with firewall rules. The user requests, both from the website and mobile application, come from the external network.
To register a new account, the user must sign up on the website page with the desired email, a strong password that must be at least nine chars long and contain both lower and capital letters and numbers and also a confirmation password to avoid typing mistakes. Then, the server replies with a code presented to the user as a **QR code** to be scanned by the mobile application, registering the user's device.
To login into the website, the user must sign in with the registered credentials and with the **2FA token** [1] that is available on the mobile application. After the login, the user gets a session token, that is used to perform all the following API requests.

2.2 Deployment

We used **Google Cloud Platform** [2] to deploy the whole infrastructure with the intention of making the process closer to reality. By creating three different virtual machines, we accomplish to deploy the database, the web server and the web app server, each one in a different machine. These three machines are inserted in an internal network.
The necessary products of the GCP are the following:

- **Virtual Private Cloud Network** for the private internal network (cryptomail network) and the necessary firewall rules.
- **Compute Engine** to create the virtual machines.
- **Container Registry** with the docker container of both servers and database
- **DNS Managed Zone** to publish our domain names to the global DNS.

2.3 Secure channels configured

In this scheme, we have 3 main entities: the **web server**, the **web application** and the **smartphone application**. The web server handles the authorization protocol with the smartphone as well as serving the web application and performing its operations. Therefore, to secure this communications, we rely on **HTTPS** [3] that encrypts all communications using the **TLS** [4] protocol. With this protocol we aim primarily to provide privacy and data integrity between communications. In addition, we used OpenSSL [5] to generate three certificates, one for each server, signed by the other that corresponds to our CA and should be installed on both the computer and the smartphone.

2.4 Secure protocols developed

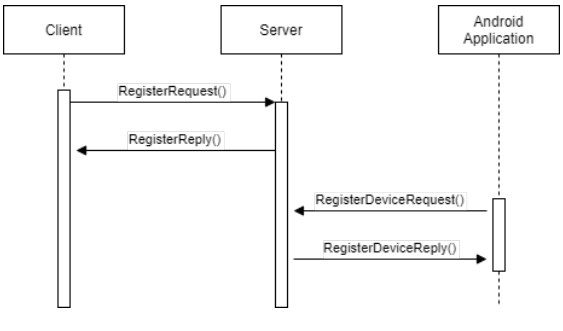


Figure 2: Register protocol

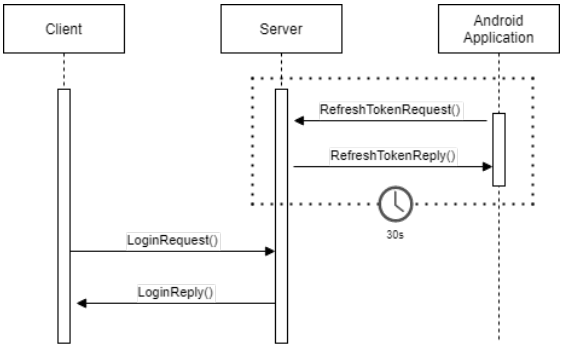


Figure 3: Login protocol

2.4.1 Register protocol

The custom protocol starts with the register of the Client in the Web App, in which he sends a `RegisterRequest()` with his mail and password twice. The server will only get the digest of the password string in order to guarantee the user's privacy. Afterwards, the Server responds to the request with a QR code that the user has to read with our application in his smartphone. Then, the smartphone sends the `deviceId` that identifies the smartphone, the QR code read and the client's email, to which the server will reply with an Ack to confirm that everything went smoothly.

```
RegisterRequest {
    string mail
    string password *
    string confirmationPassword *
}
```

```
RegisterReply {
    string QRcode
}
```

```
RegisterDeviceRequest {
    string deviceId *
    string QRcode
    string mail
}
```

```
RegisterDeviceReply {
    bool Ack
}
```

2.4.2 Login protocol

After the register has been made successfully, we are now able to start the login protocol. In this phase, the client sends his credentials to the server to which the server will respond with an Ack. After this step, the smartphone application will ask the server for an authentication token that will allow the user to complete the login in the Email System.

```
LoginRequest {
    string mail
    string password *
    string tfaToken
}
```

```
LoginReply {
    bool Ack
    string sessionToken
}
```

```
RefreshTokenRequest {
    string mail
    string deviceId *
}

*hashed
```

```
RefreshTokenReply {
    string TFAtoken
    string expirationTS
}
```

3 Used Technologies

- **Vue.js:** JavaScript framework for building the Web Application interface.[6]
- **Kotlin for Android:** Programming language for building the Mobile Application.[7]
- **Spring Boot:** Java framework for building the Web Application backend.[8]
  - **Spring Data JPA:** ORM Database support to store tables with tokens, user's and mails' information.[9]
- **amen.pt**: Service to register the cryptomail.pt domain.
- **ssllabs.com**: Deep analyzer of the configuration of our SSL servers.
- **hostedscan.com**: Platform to scan servers and websites for security risks.
- **securityheaders.com**: Tool to assess the security of our HTTPS response headers.

4 Results

The presented solution fully satisfies all the proposed requirements except for the availability that is partially satisfied which is explained during this section. Since all the traffic is done through https with TLS version 1.3, the TLS protocol ensures the most part of the desired requirements like confidentiality, integrity, data origin authentication, and freshness.

4.1 Authentication

As authentication is the main goal of this work, we needed to fully satisfy the following requirements in addition to using two-factor authentication:

- **Integrity:** TLS provides data integrity by calculating a message digest.
- **Non-repudiation:** As to perform any operation the user needs to be logged in and to log in it is necessary the user password and access to the user device, we assume the user made the login and every following operations containing the generated session token.
- **Freshness:** In TLS handshake, a nonce (random number, unix timestamp) is used to provide freshness to each message avoiding replay attacks.
- **Data origin Authentication:** For server authentication, during TLS handshake the client encrypts the data used to compute the secret key with the server's public key present in the server certificate, which must be valid. Only if the server can decrypt the data with the correct private key can it generate the secret key. For client authentication, the server decrypts the data sent by the client during the handshake using the public key in the client certificate.
- **Entity Authentication / Identification:** After login into the system, the user is identified by a session token signed by the server containing the user id, email and the timestamp of the expiration time.

4.2 Confidentiality

This requirement it's fully satisfied since all the client-server communication data is encrypted, since all the traffic is done through https.
During the TLS handshake, the TLS client and server agree on an encryption algorithm and a shared secret key that will be used only for that session. All messages sent between the TLS client and server are encrypted with that algorithm and key, ensuring the message's privacy even if intercepted. There is no key distribution problem because TLS uses asymmetric encryption when transporting the shared secret key.

4.3 Perfect Forward Secrecy

TLS employs the Ephemeral Diffie-Hellman key exchange protocol, which generates a one-time key that is only valid for the duration of the current network connection. The key is discarded at the conclusion of the session. Attackers can still capture and store encrypted network traffic, but they'll need the unique key for each session to decode it. A session key won't assist an attacker discover future session keys or decode data sent during a previous session.

4.4 Availability

We choose not to limit login attempts since the attacker could do that in order to compromise the availability of the system trying to login n times with any user email making the correct user unable to login. This decision comes with the problem of brute force dictionary attacks, that we mitigate by using a key derivation function on the client side since it consists of iterating the hash of the user password 1000 times making it expensive for the attacker perform this type of attacks. This key derivation function also consists of salting the user password with the respective email to difficult the use of rainbow tables when performing dictionary attacks since the attacker needs to generate a table for each user email.
We also mitigate DDoS Attacks by defining the firewall rules present in Figure 4 preventing some common attacks of this type like ping and HTTP flooding. However, it is still possible for the attacker to perform HTTPS flooding attacks but those could be prevented by filtering the requests by IP (e.g. Spring Cloud Gateway[10]), and so if some IP starts making unusual traffic like a lot of requests per second the server drops the requests.

Rule	Itf.	Source IP	Source Port	Transp. Proto.	Dest. IP	Dest. Port	State	Action
1	*	*	*	*	*	*	Establ.	Accept
2	lan	webserver-ip	>1023	TCP	db-ip	5432	New	Accept
3	inet	external	>1023	TCP	webserver-ip	443	New	Accept
4	inet	external	>1023	TCP	webapp-ip	443	New	Accept
5	*	*	*	*	*	*	*	Drop

Figure 4: Firewall Rules.

4.5 Usability

Besides all the security provided by the application, we ensure that users can easily interact with the application, recurring to the use of QR codes in the registration process, a simple two-factor authentication token with just four digits, and a session token automatically injected into the following requests. We choose to force the user to write the two-factor token instead of just pressing an authorization button to ensure that the user doesn't allow unknown login attempts unintentionally.

4.6 Additional Attacks

- **Password dictionary attacks:** By forcing the user to provide a strong password, we reduce the risk of a login attempt to succeed.
- **Cross-site scripting (XSS):** Since we configured the request headers such as *X-Content-Type-Options, X-Frame-Options, X-XSS-Protection, Strict-Transport-Security* and *Content-Security-Policy*, we prevent any type of injection with malicious scripts.
- **SQL Injection:** With Spring Boot, we were able to perform Parameterized Queries that forbids malicious queries.
- **Man-In-The-Middle Attack:** By providing Data Origin Authentication, we assured that the clients are exchanging messages with the server and not someone else.

5 References

[1] "Multi-factor authentication," Dec 2021. [Online]. Available: https://en.wikipedia.org/wiki/Multi-factor_authentication

[2] "Google cloud platform." [Online]. Available: https://cloud.google.com/

[3] "Https," Dec 2021. [Online]. Available: https://en.wikipedia.org/wiki/HTTPS

[4] "Transport layer security," Dec 2020. [Online]. Available: https://pt.wikipedia.org/wiki/Transport_Layer_Security

[5] I. OpenSSL Foundation. [Online]. Available: https://www.openssl.org/

[6] [Online]. Available: https://vuejs.org/

[7] "Kotlin and android: Android developers." [Online]. Available: https://developer.android.com/kotlin

[8] "Spring boot." [Online]. Available: https://spring.io/projects/spring-boot

[9] "Spring data jpa." [Online]. Available: https://spring.io/projects/spring-data-jpa

[10] S. Boot. [Online]. Available: https://spring.io/blog/2021/04/05/api-rate-limiting-with-spring-cloud-gateway