Practical Security Assignment

Computer Systems Security

Alexandre Nunes (up202005358)
Fábio Sá (up202007658)
Inês Gaspar (up202007210)

May 2024

Contents

1 Introduction

The aim of this assignment is to develop a secure system where a trusted server provides services to client applications, used by collaborators within an organization. The services are REST endpoints that can be called by the client applications and have an associated security level. Additionally, collaborators can communicate with each other through the client application.

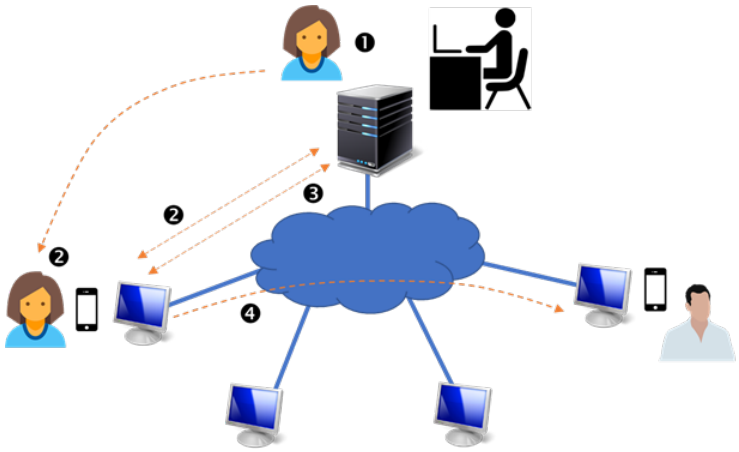

Figure 1: Assignment Scenario

The project requirements are outlined as follows:

A. Collaborators need to perform a **face-to-face pre-registration** (1), where their identity (full name, and other possible and convenient information) is verified and stored in a file or database controlled by the server. At the same time, a username (string with a maximum of 8 characters) and a one-time ID (a random string with 12 characters, comprised of small and capital letters and digits) is generated and communicated to the collaborator. The server also stores the security clearance level (another integer) of the collaborator.

B. The collaborator should then install the client application and perform the **registration operation** with the server (using the username and the one-time ID) (2). In this registration, the client application and server should generate/store the necessary cryptographic means to perform an **automatic authentication using asymmetric cryptography** of the client with the server for each session. That instance of the client after registration always represents the same collaborator, from that point on.

The client application is **responsible for identifying and authenticating the collaborator locally**, and only authenticate with the server after this.

C. **The client can invoke any service available on the server** (3), but authorization is granted only is the security clearance of the collaborator is greater or equal to the security level of the service (this is a simplified version of a mandatory access control). For the purpose of a proof of concept consider three different services (for instance, the calculation of a square root, a cubic root, and a parameterized n-root, where the provided value parameters and the parameter n are from a double float type). Each of these services has a different security level (1, 2, or 3). Moreover, consider at least three users having 3 different security clearances (also 1, 2, or 3).

D. **A collaborator can send a message to another collaborator** (4) using appropriate services provided by the server at security level 1 and enabling such communication. The message should travel directly from the client application of the sender to the client application of the recipient, when he is online. Client applications are listening on a port for those messages. From session to session, the listening IP address and port can change (note that this is what typically happens in mobile applications).

E. **All information** that circulates on the network (even the initial registration) **between clients and the server must have guaranties of confidentiality and integrity**. Also, **each endpoint** in a communication **must be identified and authenticated**, and message sending must have a **protection against replay**. Each message received must have the means to **verify authorship and non-repudiation** too.

F. **The messages must be stored** in the recipient computer or device, **in confidential form**, with proper means to recuperate the clear text, **detect changes, and proof of authorship and non-repudiation**. The recipient collaborator should be able to display the message whenever he wants.

G. **Sensitive data** must be **protected wherever is stored**.

These requirements have been **successfully implemented** and will be elaborated upon in subsequent sections. The technologies employed include **NodeJS** for server-side operations and **Flask** for client-side functionalities. We also assumed that the clients will have the server public key on installation.

2 Notation Key

For symmetric cryptography, we used AES-128/256 in GCM mode. Regarding asymetric cryptography and signatures, we used RSA with PKCS1 padding.

In some algorithms, such as AES, additional parameters like random Initialization Vectors (IV) are used alongside the key. In the subsequent sections, we'll simplify our diagrams by excluding these parameters. It's crucial to note that under the hood, we securely manage all necessary parameters, including those not explicitly depicted. These parameters are also public algorithmic components; thus, our emphasis remains on the management and security of the keys themselves.

3 Face-to-face pre-registration

3.1 Protocol

We created endpoints both on the client and server sides to simulate this process, which could otherwise be done in a face-to-face mode. The user only provides their name, selects a security level, and the server responds with a generated username and onetimeid, which are necessary for the next phase.

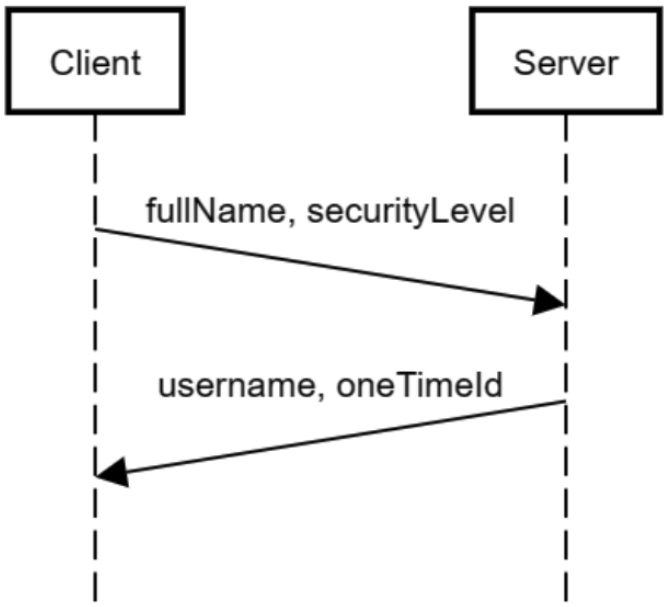

Figure 2: Face-to-face pre-registration

3.2 Proof of Security

At this stage, we don't have security concerns in the application since it's supposed to be a face-to-face process.

4 Registration

4.1 Protocol

For the registration protocol, we have the following:

1. Client sends his username and the message "REGISTER" encrypted with the oneTimeID received in the face-to-face registration.
   - The server will search the username in the database and fetch the corresponding oneTimeID. If he is able to decrypt the message and get the "REGISTER" message he accepts the request
2. Server generates a pair or asymmetric keys for the client, stores the publicKey in the database and returns the privateKey and the session information (ID, key and nonce).
   - The asymmetric keys will be used later to be able to authenticate the user
   - In this process we return a session since we can also perform the login in the same procedure
   - The nonce will be used in subsequent requests to prevent replay attacks
3. Client acknowledges that he received the keys
   - The sessionID will be used in all subsequent session requests to identify the user, we could use the username but to have some confidentially on who is making the requests we will use a random generated identifier
   - The need of this message is described below

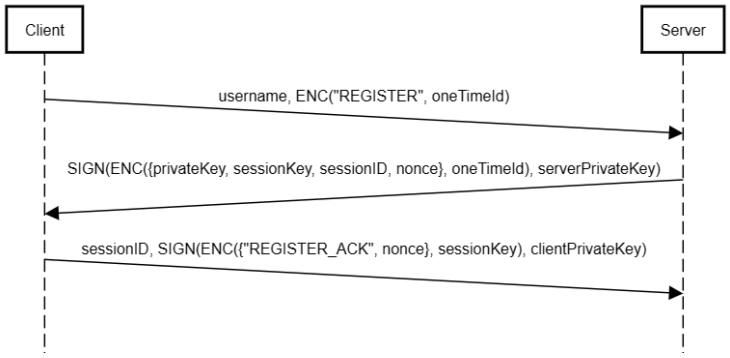

Figure 3: Register Protocol

4.2 Why do we need the "REGISTER_ACK" message?

Consider the following scenario where the client initiates communication, but the server's response is intercepted by an attacker, halting the registration process.

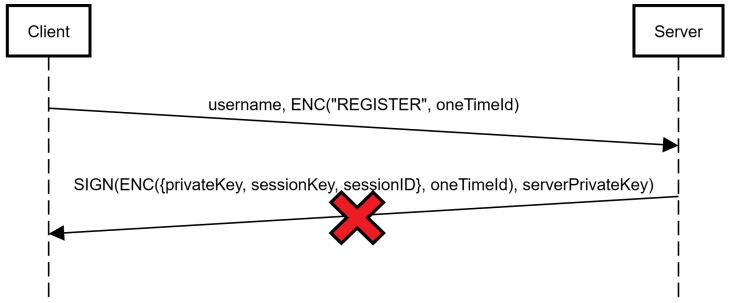

Figure 4: Need for an ACK in register

Despite the attacker inability to do anything, as consequence the client didn't received his private key. This will result in a state where the user can't do any operation, neither register again, and needs another one-time-id since the registration process is intended to be an one-time operation.

To mitigate this risk, the server needs to know the legitimate user received the private key. To satisfy this requirement the server must implement a mechanism to accept registration requests again and resend the keys upon request. Once the ACK is received, confirming that the legitimate user has obtained the keys, the server stops accepting any subsequent registration requests for that user.

This procedure ensures that legitimate users are not unfairly obstructed from registering due to communication disruptions or malicious interference.

4.3 Proof of Security

4.3.1 Confidentiality

The attacker doesn't know the one-time-id so he can't decrypt the messages, including the session key. The only information he knows is the username publicly sent in the first message.

4.3.2 Integrity

Any attempt by the attacker to modify messages will lead to failure in signature verification by the client/server. He doesn't know the corresponding private keys so he also can't forge signatures.

In the first message, that is a special case, the attacker also can't modify the message, otherwise the server will fail to fetch the one-time-id or fail to get the "REGISTER" string.

4.3.3 Authentication/Authorship/Non-repudiation

These attributes are guaranteed by the signature, as only the client/server possesses the knowledge of the one-time ID and possesses the ability to sign messages using the corresponding private keys.

4.3.4 Replay Attacks

It would be a problem if the server generated different keys each time he receives the first message of the protocol, however, we handle that and it only generates them upon first client request. In the subsequent requests he answers with the previous generated keys. When the server receives the ACK he stops accepting register requests, so there isn't any potencial threat.

5 Login

5.1 Protocol

For the login protocol, we have the following:

1. Client sends the username and a message "LOGIN" along with the nonce encrypted with the server public key, signed with his private key.
   - The nonce was obtained in previous interactions. Both the client and the server store this value to synchronize and prevent replay attacks.
2. Server returns the session details encrypted with the client public key, signed with his private key.

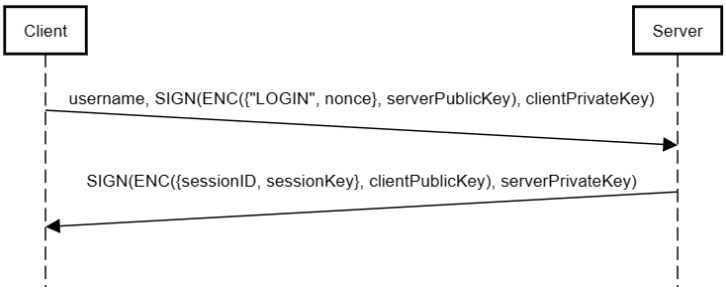

Figure 5: Login Protocol

5.2 Proof of Security

5.2.1 Confidentiality

The messages exchanged are encrypted with the other end public key so only the legitimate actor is able to decrypt. The attacker doesn't have the private key of any of the actors so he can't decrypt anything. He only knows the username of the user trying to login.

5.2.2 Integrity

Any attempt by the attacker to modify messages will lead to failure in signature verification by the client/server. He doesn't know the corresponding private keys so he also can't forge signatures.

5.2.3 Authentication/Authorship/Non-repudiation

These attributes are guaranteed by the signature, as only the client/server possesses the ability to sign messages using the corresponding private keys.

5.2.4 Replay Attacks

The nonce ensures that the attacker can't replay the message to start new sessions by the user. Every login message by the legitimate user will have different values.

6 Server Services

6.1 Protocol

For the services protocol, we have the following:

1. Client sends the sessionID and a message with the operation to be performed (square root, cubic root or nth root), the value of the operation and the nonce, everything encrypted with the session key and signed with his private key.

2. Server returns the result encrypted with the session key and signed with his private key.

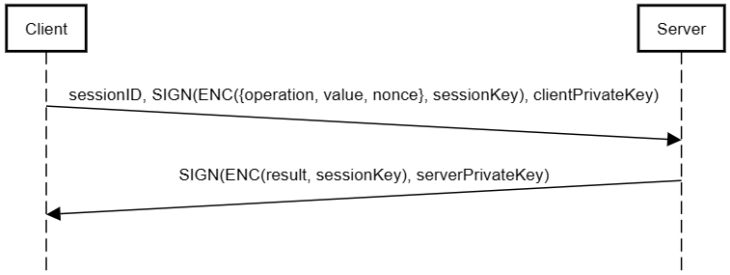

Figure 6: Services Protocol

6.2 Proof of Security

6.2.1 Confidentiality

The messages exchanged are encrypted with the session key so only the legitimates actors are able to decrypt. The attacker doesn't have the key so he can't decrypt anything.

6.2.2 Integrity

Any attempt by the attacker to modify messages will lead to failure in signature verification by the client/server. He doesn't know the corresponding private keys so he also can't forge signatures.

6.2.3 Authentication/Authorship/Non-repudiation

These attributes are guaranteed by the signature, as only the client/server possesses the ability to sign messages using the corresponding private keys.

6.2.4 Replay Attacks

The nonce ensures that the attacker can't replay requests.

7 Messages between collaborators

7.1 Protocol

With regard to the protocol implemented for exchanging messages, we have drawn up the following diagram. To make it easier to explain, we named the two clients Alice and Bob.

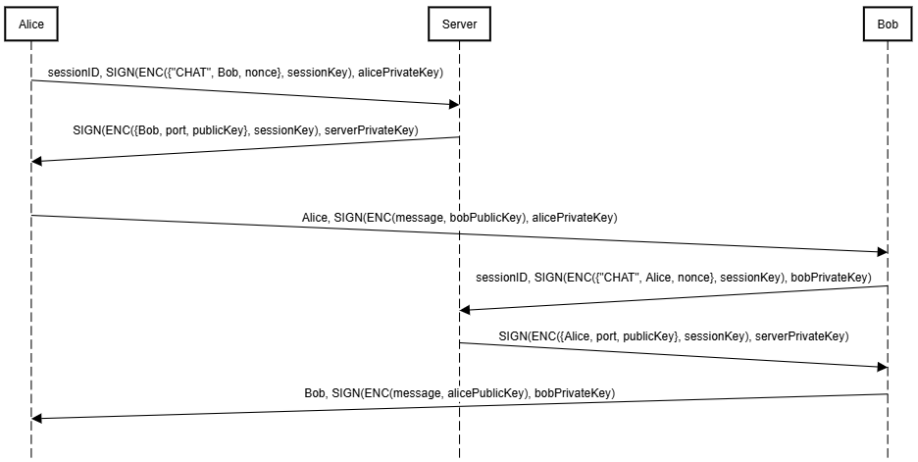

Figure 7: Message Protocol

Suppose Alice wants to initiate contact with Bob. Alice needs to know the port Bob is on. So Alice sends an encrypted and signed message with the operation she wants to, 'CHAT', and the username of the person she wants to contact. The server replies to Alice with the port Bob is on and his encrypted and signed public key. As soon as she receives the server's reply, Alice decrypts the message and from then on has access to Bob's port and stores his public key locally. From then on, Alice can send messages directly to Bob.

Alice then sends Bob a message with the content she wants and her username. This message is encrypted with Bob's public key and signed with Alice's private key.

As soon as Bob receives a message from Alice for the first time, he sends a request to the server to find out the port where Alice is and her public key. The server responds to Bob with this information.

From the moment Bob receives this data, he and Alice can send messages directly to each other without the intermediary of the server. The messages they send are encrypted with each other's public key (guaranteeing confidentiality) and signed with their private key (guaranteeing non-repudiation). All messages sent are stored locally in a file so that a history of the messages can be obtained.

7.2 Proof of Security

7.2.1 Confidentiality

Confidentiality is guaranteed in the same way as before: the fact that messages are encrypted with the recipient's public key means that only the recipient can decrypt the message, i.e. even if the message is intercepted by an attacker, he cannot access the content.

7.2.2 Integrity

Integrity is once again guaranteed in the same way as in other protocols explained. The fact that the messages are signed with the sender's private key ensures that if the message is altered it is not possible for the attacker to forge the signature.

7.2.3 Authentication/Authorship/Non-repudiation

The fact that messages are signed guarantees the authorship of the sender and also guarantees non-repudiation. The fact that we save the history of messages in a file also serves as a guarantee of non-repudiation.

7.2.4 Replay Attacks

The nonces, used in the other protocols mentioned, guarantee that there are no replay attacks.

8 Tests

All protocols and use cases were tested thorough unit tests. We carefully considered every possible scenario and potential attack for each protocol, creating tests accordingly.

We use the following npm packages: **mocha**, **supertest** and **chai**. They allowed us to perform requests to the server as a client, this way we could ensure that both the system and its protocols works precisely as intended. By aligning with the security proofs outlined in this report, we further boost the confidence in the system's security.

In total, there are 20 tests. These demonstrate the application's capabilities in its normal operation and against replay attacks, as well as the throwing of exceptions if integrity parameters, authentication, and others are compromised.

9 Conclusion

This project allowed us to deepen the knowledge of cybersecurity, as well as to become more aware of the complexity and importance of thinking about and designing a secure system from the very first stage, namely from the moment

the functionalities and accesses are defined. Another very fundamental point was to realize how challenging it can be to model and build a system in which we have to have all the guarantees of confidentiality, integrity and authenticity (and non-repudiation) and prevention against certain accounts.