Logo

Description automatically generated

**CSE4057**

**Information Systems Security**

**Project**

150119661 Samet Enes Örsdemir

150119663 Resul Akçakaya

150118886 Hamza Kavak

# **Part 1: Basic Cryptography**

# **1-) Generation of public-private key pairs**

By using the Crypto library in Python, we have created an RSA public-private key pair. The length of the keys is 1024 bits.

Ka+ value stored in file name: **public\_key[username].pem**

Ka- value stored in file name: **private\_key[username].pem**

A screen shot of a computer code

Description automatically generated with low confidence

Output Ka+ value and Ka- value

A picture containing text, screenshot, font, document

Description automatically generated

# **2-)Generation of Symmetric Keys**

256-bit symmetric key is generated using the PBKDF2 algorithm with a password and salt value. The derived key length is set to 32 bytes, which corresponds to 256 bits.

A picture containing text, screenshot, software, font

Description automatically generated

Output

A screen shot of a computer

Description automatically generated with low confidence

# **3) Generation and Verification of Digital Signature**

We have text over 100 characters. Thanks to the SHA256 algorithm, we get the hashed string and then encrypt and decrypt the hashed string to show m, H(m) and the digital signature (hence verification) on the screen. You can see that the encrypted and decrypted values are the same. The validation result is correct.

A screen shot of a computer program

Description automatically generated with low confidence

A picture containing text, screenshot, font

Description automatically generated Output

**4-) AES Encryption/Descryption**

A Random Initialization Vector (IV) was generated for encryption in CBC Mode with AES (256-bit key). Then the text message was encrypted with AES (256 bits) using IV and key.

The ciphertext was decrypted using AES (256 bits) using the key and the IV decrypted the message. Confirmed that the decrypted text is the same as the original text.   
A screenshot of a computer program

Description automatically generated with medium confidence

Output

A picture containing text, screenshot, font, document

Description automatically generated

# **5-) Message Authentication Codes**

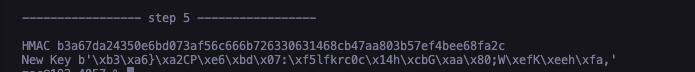
We created message verification code by applying HMAC-SHA256 algorithm using any symmetric key.

We applied the pre-generated Message Authentication Code (HMAC-SHA256) to KS to generate a new 256-bit key.

A screen shot of a computer code

Description automatically generated with low confidence

Output



You can see the codings for part 1 in the main.py file. These codes are also used for part 2 in the client.py file.

# **Part 2**

**Server-side**

The server must be running for users to start messaging. It listens for requests from port 8080 and creates a new threat for each new request.

The server stores and logs its own public - private RSA keys and all steps of process. A screenshot of a computer

Description automatically generated

**Client-side**

**Our project designs and implements a secure P2P messaging system. The system encompasses the following features:**

To initiate messaging, the client.py file needs to be executed. Upon running, a login screen welcomes the user, allowing them to perform the login and registration processes.

A screenshot of a computer

Description automatically generated with medium confidence

A unique port number is generated based on the username and stored on the server. This allows other users to obtain the port number from the server when they intend to send a message.

A picture containing text, screenshot, font

Description automatically generatedFurthermore, each user generates a public-private key pair during the initial setup and registers with the server using their username and public key.

The server, upon receiving the user's public key, signs it using its private key. This process creates a certificate, which is stored by the server and sent to the user. Upon receiving the certificate, the user verifies its authenticity and ensures that the server has received the correct public key.

A screenshot of a computer

Description automatically generated with medium confidence

In the certificate authority (the server), the public key of each user is stored encrypted with the server's own public key. Consequently, decryption of these public keys can only be done by the server itself.

A picture containing text, screenshot

Description automatically generated

All relevant logs, including keys, certificates, and the verification process results, are stored in a file named client\_[username].log. This log file provides a comprehensive record of the application's activities and allows users to review the various components of their messaging setup.

A screenshot of a computer program

Description automatically generated with medium confidence

When User1 wishes to communicate with User2, they send a hello message along with their public key certificate to User2. User2 sends back a random number (nonce) along with their public key certificate. User1 encrypts the nonce with their private key and sends it back to User2. User2 verifies the nonce and sends an acknowledgement to User1.

Then User1 generates a master secret and sends it to User2 in a secure way.

A screenshot of a chat box

Description automatically generated with medium confidence

*User1 chat screen*

A screenshot of a chat box

Description automatically generated with medium confidence

*User2 chat screen*

Both User1 and User2 generate the necessary keys for encryption and Message Authentication Code (MAC), as well as initialization vectors (IVs). These keys and IVs are derived from the master secret. All messages between pairs are encrypted using AES in CBC mode.

Despite the secure transmission of messages with encryption and MACs, a potential replay attack is thwarted. Added timestamp method for item 6 of the assignment. Here we block replay attacks using timestamp. A screen shot of a computer program

Description automatically generated with low confidence

For example, we prevent messages that have passed 1 minutes from being brought back at other times.

Another feature of the system is that it retains the conversation history. When a user logs in again and wishes to send a message to a previously contacted user, they can also view their past conversations. This functionality allows users to access and review their previous interactions in a visually appealing manner.

A screenshot of a computer

Description automatically generated with medium confidence

This project provides a secure platform for P2P messaging, ensuring the confidentiality, integrity and authenticity of messages exchanged between users.

**Security Hole:**

Storing sent messages as plain text can create security vulnerabilities. Therefore, encryption should also be used for local message storage. Encryption protects against unauthorized access by ensuring the confidentiality and integrity of messages.

**How we communicate and coordinate:**

First we created a WhatsApp group and when we started the project, we held meetings on Discord and communicated there.

**Which parts are done by whom:**

**Samet Enes Örsdemir:**

Part 1: Generation and Verification of Digital Signature

Part 1: AES Encryption/Descryption

Part 2: Resistance to Replay Attacks.

Part 2: Handshaking.

**Hamza Kavak:**

Part 1: Message Authentication Codes

Part 2: Key Generation.

Part 2: Message Encryption.

Part 2: Integrity Check.

**Resul Akçakaya:**

Part 1: Generation of public-private key pairs.

Part 1: Generation of Symmetric keys

Part 2: Public Key Certification.

Bug fix and comment lines

**Collaborative work:** Preparing report, User Interface

**How did we merge our codes:**

Once each team member completed their respective tasks, we organized discussions to review and merge our codes. During these collaborative sessions, we utilized screen sharing capabilities on Discord to facilitate code merging. This allowed us to carefully examine and combine our code segments, ensuring seamless integration and compatibility across different sections of the project. Through this iterative process, we achieved a unified codebase that reflected the contributions of each team member.