|  |  |
| --- | --- |
| **Name** | **Student Number** |
| Mahomed Aadil Ally | ALLMAH002 |
| Tapera Chikumbu | CHKTAP011 |

**CSC4026Z: NIS - Assignment**

**Cryptosystem Design**

The cryptosystem is designed to ensure secure communication between two parties, Alice and Bob, by leveraging various cryptographic techniques. The primary components of the system are:

* **Asymmetric Key Encryption**: RSA is used for asymmetric key encryption and decryption, employing the "RSA/ECB/PKCS1Padding" algorithm. Each party (Alice and Bob) generates their own RSA key pair (public and private keys).
* **Symmetric Key Encryption**: AES is used for symmetric key encryption and decryption, utilizing the "AES/CBC/PKCS5Padding" algorithm. The shared secret key for AES is derived from the Diffie-Hellman key exchange.
* **Key Exchange**: Diffie-Hellman key exchange is implemented to securely generate and exchange a shared secret key between Alice and Bob.
* **Hashing**: SHA-256 is used for securely hashing the compressed message, ensuring message integrity.
* **Compression**: The zlib library is used for compressing the message (caption and image data) before encryption, improving efficiency.
* **Certificate-based Authentication**: X.509 certificates are used for authenticating Alice and Bob. Each party's certificate is signed by a trusted Certification Authority (CA). The CA's public key and certificate are pre-shared between Alice and Bob.

**Communication Connectivity Model**

The communication model follows a client-server architecture, where Alice acts as the client, and Bob acts as the server. The steps involved in the communication process are:

1. Bob starts the server and listens for incoming connections.
2. Alice connects to Bob's server.
3. Alice and Bob exchange their certificates and verify each other's certificates using the trusted CA's public key.
4. Alice and Bob perform the Diffie-Hellman key exchange to generate a shared secret key.
5. Alice compresses the message (caption and image data), generates a SHA-256 hash of the compressed message, encrypts the compressed message using the shared AES key, and encrypts the hash using Bob's RSA public key.
6. Alice sends the encrypted message and the encrypted hash to Bob.
7. Bob receives the encrypted message and the encrypted hash, decrypts the message using the shared AES key, and decrypts the hash using Alice's RSA public key.
8. Bob verifies the integrity of the decrypted message by comparing the decrypted hash with the computed hash of the decrypted message.
9. If the integrity check passes, Bob decompresses the message, saves the image, and displays the caption.
10. Alice and Bob close the connection.

**Key Management**

Key management is a crucial aspect of the cryptosystem, and the following measures are implemented:

* **RSA Key Generation**: Alice and Bob each generate their own RSA key pairs (public and private keys) using the cryptography library.
* **Diffie-Hellman Key Exchange**: Alice and Bob perform the Diffie-Hellman key exchange to securely generate and exchange a shared secret key, which is then used to derive the AES key for symmetric encryption and decryption.
* **Certificate Management**: Alice and Bob's public keys are packaged into X.509 certificates signed by a trusted Certification Authority (CA). The CA's public key and certificate are pre-shared between Alice and Bob.
* **Key Distribution**: Alice and Bob exchange their certificates and public keys during the initial communication phase, allowing them to authenticate each other and establish a secure communication channel.

**Choice of Cryptographic Algorithms**

The following cryptographic algorithms were chosen for this implementation:

* **RSA**: RSA is a widely-used and secure algorithm for asymmetric key encryption and decryption. It is employed for encrypting the message hash and exchanging public keys.
* **AES**: AES is a highly secure and efficient algorithm for symmetric key encryption and decryption. It is used for encrypting the compressed message using the shared secret key derived from the Diffie-Hellman key exchange.
* **SHA-256**: SHA-256 is a secure hashing algorithm used for generating a message digest, ensuring message integrity.
* **Diffie-Hellman**: The Diffie-Hellman key exchange algorithm is used to securely generate and exchange a shared secret key between Alice and Bob, which is then used to derive the AES key for symmetric encryption and decryption.

These algorithms were chosen based on their widespread acceptance, security strength, and compatibility with the cryptography library used in the implementation.

**Testing Procedures and Assumptions**

The testing procedures for the implementation involve the following steps:

1. Run the Bob.py script to start the server.
2. Run the Alice.py script to initiate the client connection and secure communication.
3. Verify the console output on both Alice's and Bob's sides, ensuring that the encrypted messages, hashes, and digests are displayed correctly.
4. Check that the received image file (received\_image.png) is saved correctly and matches the original image.
5. Verify that the displayed caption matches the original caption.

The following assumptions were made during the development and testing of the implementation:

1. The CA's public key and certificate are pre-shared and available as ca\_public\_key.pem and ca\_cert.pem files, respectively.
2. Alice and Bob have access to the CA's private key for signing their certificates. If this is not the case, self-signed certificates or certificates obtained from a trusted CA should be used instead.
3. The image file (image.png) is present in the same directory as the scripts.
4. The cryptography library is installed and available for use.
5. The code is executed in a secure environment, and the private keys and certificates are kept confidential.

**Execution Instructions**

To execute the submitted program, follow these steps:

1. Ensure that the cryptography library is installed. If not, install it using pip install cryptography.
2. Place the ca\_public\_key.pem, ca\_cert.pem, and image.png files in the same directory as the Alice.py and Bob.py scripts.
3. Run the Bob.py script first to start the server: python Bob.py
4. In a separate terminal or console window, run the Alice.py script to initiate the client connection and secure communication: python Alice.py
5. Observe the console output on both Alice's and Bob's sides to verify the encryption, decryption, and message integrity processes.
6. Check for the received\_image.png file in the same directory, which should contain the received image.
7. The caption of the received image will be displayed in the console output.