Cryptography 2nd Project

Le Duy Nguyen **xnguye27**

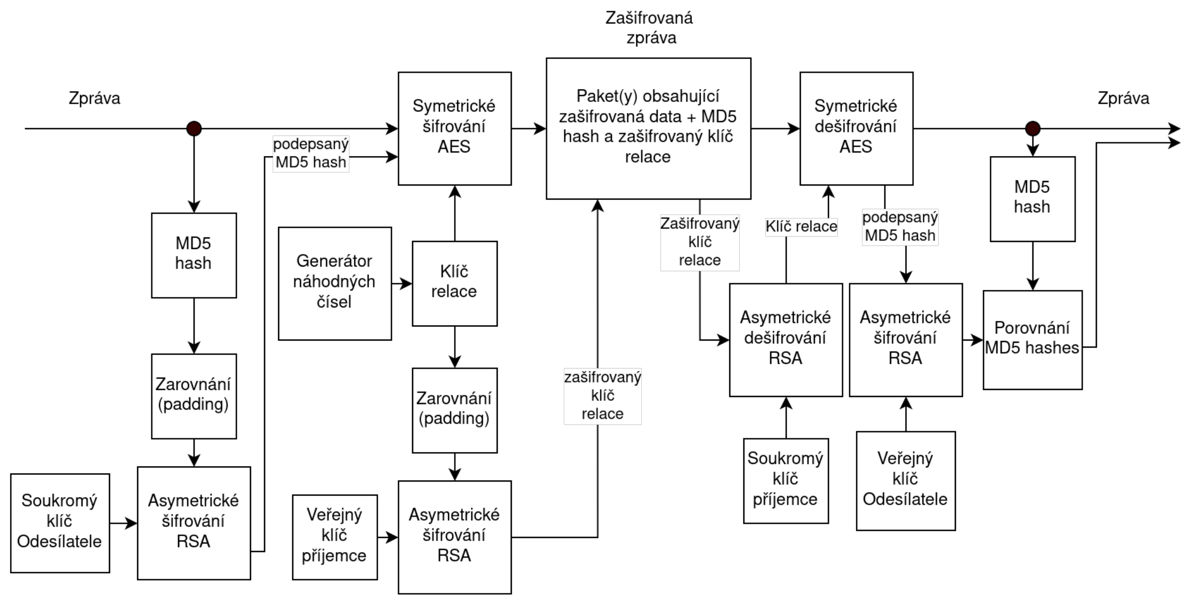
1. Overview

This project implements hybrid encryption using **RSA2048** and **AES128**. The encryption process involves encrypting the **MD5** hash of the message with RSA using the sender's private key, signing it into the message, and encrypting the signed message with AES using a randomly generated key and IV. The padded key is then encrypted with RSA using the receiver's public key, and the encrypted key and IV are appended to the data.

The decryption process involves decrypting the AES key with RSA using the receiver's private key, unpadding the key, and decrypting the signed message with that key and IV. The integrity of the message can be checked using the MD5 hash, which is decrypted with the sender's public key and compared with the message MD5.

The communication process involves the sender encrypting the data using hybrid encryption and adding a nonce to it. The server responds with a 128-bit md5 hash, which is used to check the integrity of the message. If the integrity matches, the receiver responds with the md5 hash of nonce combined with the raw message.

In summary, this project demonstrates how to use hybrid encryption to secure communication between a sender and receiver using RSA2048 and AES128.



2. Communication

2.1. Strategy

```
  Sender                                      Receiver
  ------                                      --------
    |                                             |
    | 1. Encrypt with hybrid encryption           |
    |                                             |
    |                                             |
    | 2. Send encrypted message with              |
    |    a nonce and payload length               |
    |   ------------------------------------>     |
    |                                             |
    |                       3.Decrypt the message |
    |                                             |
    |                          4. Integrity check |
    |                               with md5 sign |
    |                                             |
    |                          4.1 If md5 matches |
    |           4.1.1. Send back hash of combined |
    |                  nonce with the raw message |
    |      <-----------------------------------   |
    |                                             |
    | 4.1.2. Success                              |
    |                                             |
    |                   4.2 If md5 does not match |
    |            4.2.1. Send back 16 random bytes |
    |                  nonce with the raw message |
    |      <-----------------------------------   |
    |                                             |
    | 4.2.2 Failed                                |
    |       Repeat 2.                             |
```

2.2. Package Layout

```
|---------------------------------------------------------------|
|     4 bytes    | 16 bytes |     256 bytes    |       ...      |
|---------------------------------------------------------------|
| Payload length |   nonce  |  AES key + IV    | Message + Sign |
|     (MSB)      |          | (RSA encrypted)  | (RSA encrypted) |
|---------------------------------------------------------------|

|-------------------------------------|
|           Message + Sign            |
|-------------------------------------|
|     256 bytes   |        ...        |
|-------------------------------------|
|     MD5 hash    |  Message content  |
| (RSA encrypted) |                   |
|-------------------------------------|
```

3. Hybrid Encryption Implementation

Hybrid encryption is a combination of two encryption methods: symmetric and asymmetric encryption. In this project, we implemented hybrid encryption using RSA2048 and AES128. The encryption process involves several steps, including padding, hashing, and signing.

3.1. Encryption

1. Hash message using MD5
2. Pad the hash to 2048 bits using custom padding
3. Encrypt the hash with RSA using sender's private key
4. Use the encrypted hash to sign the message
5. Generate the AES key and IV randomly
6. Encrypt the signed message using AES with the generated key and IV
7. Combine AES key and IV together then pad it to 2048 bits
8. Encypt them with RSA using receiver's public key
9. Combine the encrypted key and encrypted signed message together

3.2. Decryption

1. Get AES key and IV by decrypting the AES encrypted key with receiver's private key
2. Decrypt the signed message using that AES key and IV
3. The message is now there
4. Get the MD5 sign by decrypting it using sender's public key
5. Compare the MD5 of the message with the raw message hash to check for integrity

3.3. Keys length

As in the specification of the project

- AES cipher: 128 bits
- RSA cipher: 2048 bits

3.4. Padding

Since we only do the padding on 2 pieces of data with fixed length, the padding is as simply as just fill in random bytes until the desired length is reached. The scheme is as follow:

- Put the data at the end of the padding
- Set the first padding byte to 0
  - This will ensure that the padding data is in RSA modulus
- Fill the rest with random bytes

3.5. Random source

The `/dev/urandom` is used for this project to generate random bytes as it considered cryptographically secure and easy to use.

4. Security Analysis

The security of the hybrid encryption scheme implemented in this project is crucial to ensuring the confidentiality and integrity of the transmitted messages. In this section, we will analyze the potential security threats to the system and evaluate the measures taken to mitigate them.

4.1. RSA Key Security

The RSA encryption used in this project relies on the security of the RSA keys used. The key size of RSA2048 is considered to be secure against current cryptographic attacks and is estimated to require an impractical amount of time and resources to break. However, it is important to periodically re-evaluate the security of the RSA keys used and consider increasing the key size or switching to stronger encryption algorithms as needed to maintain the security of the system.

4.2. AES Key Security

The security of the AES key used in this project depends on the randomness and secrecy of the key and IV generated. The `/dev/urandom` generator used in this project is considered to be cryptographically secure and provides a sufficient source of randomness for the keys and IV. By encrypting the Key and IV with RSA ensures that the key and IV are kept secret and are not leaked to an attacker.

4.3. Integrity

The integrity of the transmitted messages is ensured by signing the message with the sender's private key and verifying the signature with the sender's public key. In addition, the MD5 hash of the message is encrypted with the sender's private key and decrypted with the sender's public key to verify the integrity of the message. This ensures that the message has not been tampered with during transmission and provides a way to detect any potential attacks.

4.4. Confidentiality

The confidentiality of the transmitted messages is ensured by encrypting the message with AES and encrypting the AES key with RSA. The use of AES provides strong encryption and ensures that the message can only be decrypted by the receiver who possesses the correct key and IV. The use of RSA encryption for the AES key ensures that only the receiver can obtain the AES key needed to decrypt the message.

4.5. Relay Attacks

The use of a unique `nonce` for each message transmission *may* prevents some kind of replay attacks, but not all. However, this can be mitigated by using a secure channel for message transmission.

4.6. Conclusion

In conclusion, the hybrid encryption scheme implemented in this project provides strong security for the transmitted messages. The use of RSA and AES encryption ensures confidentiality and integrity of the messages. However, it is important to periodically re-evaluate the security of the RSA keys used and consider increasing the key size or switching to stronger encryption algorithms as needed to maintain the security of the system. In addition, the vulnerability to relay attacks should be addressed to further strengthen the security of the system.

5. Libraries used

- OpenSSL MD5 hash, RSA key generation and encryption/decryption
- tiny-AES-c Simple and portable library for AES cipher in C written by kokke

6. Resources

This project cannot be done without **any** of these

- Information from the assignment by Ing. Daniel Snášel
- Presentations for cryptography course by Dr. Ing. Petr Hanáček
- https://en.wikipedia.org/wiki/Padding_(cryptography)
- https://stackoverflow.com/a/39412887 by vsjones
- https://github.com/kokke/tiny-AES-c by kokke