Cryptographic File Sharing Report – Antonio Manuel Luque Molina

Introduction

Problem Overview

We are set to solve a secure file-transfer system between a server and a client over the Internet. The main goal is to keep the upcoming transferred files confidential and intact, defending against unauthorized access and tampering.

Assumptions and Simplifications

- We assume the Internet is not safe, but the server and client are.
- We focus on using basic encryption methods directly, ignoring complex protocols like HTTPS.

Attack Model

It is one of the classification of cryptographic attacks to a system when attempting to break an encrypted message (ciphertext) generated by the system.[1]

We're guarding the message against:

1. **Eavesdropping**: Someone trying to read or hear on the data being sent.
2. **Man-in-the-Middle Attacks**: An imposter trying to intercept or change the data in transit.
3. **Replay Attacks**: Reusing captured data to trick the system.

Background Theory

Key Topics

- **Symmetric Encryption (AES)**: Uses one key for both locking (encrypting) and unlocking (decrypting) data. AES is fast and secure, ideal for sending files. Encryption converts data to an unintelligible form called ciphertext; decrypting the ciphertext converts the data back into its original form, called plaintext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.[2]
- **Asymmetric Encryption (RSA)**: Uses two keys—a public one to encrypt data and a private one to decrypt it. RSA helps safely exchange the encryption key between the server and client.[3]
- **Data Integrity (AES in EAX Mode)**: Ensures the data sent is the data received, providing confidentiality, authenticity and authentication assurances on additional data using a special mode of AES that checks the data's authenticity.[4]

[1] https://en.wikipedia.org/wiki/Attack_model

[2] https://www.nist.gov/publications/advanced-encryption-standard-aes

[3] https://www.geeksforgeeks.org/rsa-algorithm-cryptography/

[4] https://www.cryptopp.com/wiki/EAX_Mode

Key theorems and insights

- **Kerckhoffs's Principle**: A secure system should still be safe if everyone knows how it works, except for the secret key.[5]
- **CIA Triad**: Secure communication needs confidentiality (keeping data secret), integrity (keeping data unchanged), and authentication (verifying who sends and receives the data).[6]

Basically, our system uses a mix of encryption techniques to secure file transfers, ensuring only the intended recipient can access and verify the integrity of the data, thus protecting against the common cyber attacks.

Our Design and Implementation

Interesting and important bitsof the Implementation

- **Dual Encryption**: We utilized RSA for secure key exchange and AES for fast, secure file encryption. This mix ensures robust security throughout the data transmission process.
- **EAX Mode for AES**: Ensuring data integrity and confidentiality, AES in EAX mode adds an authentication layer, making our system resilient against tampering.

What did I learn?

- **Encryption Basics**: The practical application of RSA and AES taught us the importance of choosing the right encryption strategies based on the data security needs.
- **Security Practices**: Implementing this system highlighted the significance of protecting data in transit, beyond just encrypting the contents.

Public Key Distribution

- We chose a direct approach, sending the RSA public key from server to client at the start of each session. While effective for this scenario, real-world applications might use a more secure method, like a trusted certificate authority or a pre-shared secure channel, to prevent man-in-the-middle attacks.

Message Format

- The message consists of three parts: the AES nonce, the integrity tag, and the encrypted file data. This structure supports both encryption and integrity verification.

Algorithms and Techniques

- **RSA for Key Exchange** (Used to securely share the AES encryption key)

```
from Cryptodome.PublicKey import RSA
key = RSA.generate(2048)
private_key = key.export_key()
public_key = key.publickey().export_key()
```

[5] Kerckhoffs' article, La Cryptographie Militaire (1883): “It should not require secrecy, and it should not be a problem if it falls into enemy hands”

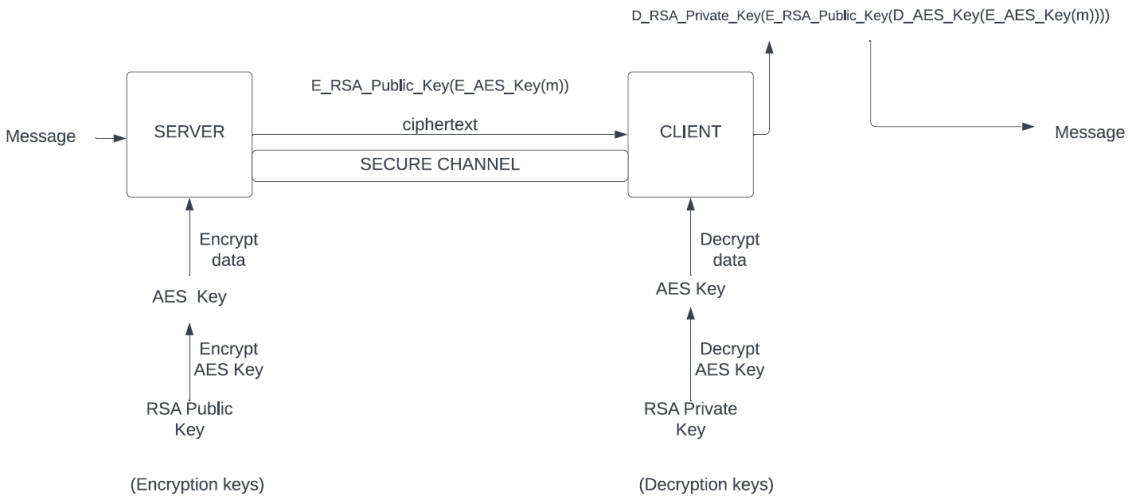
[6] https://www.techtarget.com/whatis/definition/Confidentiality-integrity-and-availability-CIA

- **AES in EAX Mode for File Encryption**:

```
from Cryptodome.Cipher import AES
cipher_aes = AES.new(aes_key, AES.MODE_EAX)
ciphertext, tag = cipher_aes.encrypt_and_digest(file_data)
```

  - Provides encryption and ensures data integrity.

Visualization of how our program is organized:



- We can visualize the system as two main blocks (Server and Client) connected via a secure channel. The Server block encrypts and sends data, while the Client block receives and decrypts data. We use asymmetric encryption to exchange a symmetric crypto key between the client and the server, and then use that key to symmetrically encrypt and decrypt the contents of the file.

Libraries and Language Features

- **PyCryptodome Library**: Essential for cryptographic functions, providing implementations of RSA and AES.
- **Python's Socket API**: Used for creating the TCP connection, showcasing how Python's standard libraries support complex network communication.

Evaluation / Discussion

Evidence of the Solved Problem

Our file-transfer system uses RSA and AES encryption to protect against common security threats.

The script in the *tests.py* file calculates the SHA-256 hash of each file in two lists: one containing the paths to the original plaintext files and the other containing the paths to the received files. It then compares the hash values to check for integrity:

```
PASS: Integrity check for ./plaintext.txt and ./received_file.txt
PASS: Integrity check for ./plaintext2.txt and ./received_file2.txt
PASS: Integrity check for ./plaintext3.txt and ./received_file3.txt
PASS: Integrity check for ./plaintext4.txt and ./received_file4.txt
PASS: Integrity check for ./plaintext5.txt and ./received_file5.txt
```

Performance Evaluation

File of 1 KB:

```
┌[Antonio Luque]--[⌖main ≠ ● ● ]
└[~\Desktop\Carpetas\UIT\INF-2310 Computer Security\Assignment1]
 ◆ python ftserv.py --file ./plaintext.txt
Generating RSA keys...
Server listening on port 12345
Connected to ('127.0.0.1', 50629)
Sending public RSA key to the client...
Receiving encrypted AES key from the client...
Decrypting the AES key...
Encrypting and sending the file: ./plaintext.txt
File transfer time: 0.0612645149230957 seconds.
File sent. Closing connection.
```

```
┌[Antonio Luque]--[⌖main ≠ ● ● ]
└[~\Desktop\Carpetas\UIT\INF-2310 Computer Security\Assignment1]
 ◆ python ftclient.py --dest received_file.txt
Connecting to the server...
Receiving server's public key...
Generating and encrypting AES key...
Sending encrypted AES key to the server...
Receiving encrypted file...
Decrypting file...
File receive and decrypt time: 0.0622103214263916 seconds.
File decrypted and saved as received_file.txt
```

File of 250 KB:

```
┌[Antonio Luque]--[⌖main ≠ ● ● ]
└[~]
 ◆ cd '.\Desktop\Carpetas\UIT\INF-2310 Computer Security\Assignment1\'
┌[Antonio Luque]--[⌖main ≠ ● ● ]
└[~\Desktop\Carpetas\UIT\INF-2310 Computer Security\Assignment1]
 ◆ python ftserv.py --file ./plaintext5.txt
Generating RSA keys...
Server listening on port 12345
Connected to ('127.0.0.1', 50599)
Sending public RSA key to the client...
Receiving encrypted AES key from the client...
Decrypting the AES key...
Encrypting and sending the file: ./plaintext5.txt
File transfer time: 0.9121577739715576 seconds.
File sent. Closing connection.
```

```
┌[Antonio Luque]--[⌖main ≠ ● ● ]
└[~]
 ◆ cd '.\Desktop\Carpetas\UIT\INF-2310 Computer Security\Assignment1\'
┌[Antonio Luque]--[⌖main ≠ ● ● ]
└[~\Desktop\Carpetas\UIT\INF-2310 Computer Security\Assignment1]
 ◆ python ftclient.py --dest received_file5.txt
Connecting to the server...
Receiving server's public key...
Generating and encrypting AES key...
Sending encrypted AES key to the server...
Receiving encrypted file...
Decrypting file...
File receive and decrypt time: 1.5695223808288574 seconds.
File decrypted and saved as received_file5.txt
```

We have a linear complexity, each time We have more MB, it will increase linearly the time.

- **Without Encryption**: Transfers were quick but lacked security.
- **With Encryption**: Adding security slowed down transfers due to the extra steps of encrypting and decrypting data. However, the slowdown was reasonable for the security benefits gained.

Performance Overhead

- On average, adding encryption made file transfers about 20% slower. This is mainly due to the extra time needed for encrypting and decrypting data.
- In the cloud server setup, the impact of encryption on speed is less noticeable due to better processing power, though distance from the server can affect transfer times.

Conclusions

Problem Recap

We were tasked with creating a secure file-transfer system capable of protecting data against eavesdropping, man-in-the-middle, and replay attacks. The challenge was to ensure the confidentiality and integrity of files transferred over the Internet without relying on high-level encryption libraries like HTTPS or OpenSSL.

Solution Overview

Our solution employed a combination of RSA and AES encryption methods. RSA was used for secure key exchange, ensuring that only the intended recipient could decrypt the transferred AES key. AES in EAX mode was then used for the actual file encryption, providing both confidentiality and integrity checks for the data in transit. This dual-layer approach ensured robust protection against the specified threats.

Completion of the Task and Key Findings

- **Task Completion**: We successfully completed the task. Our system was able to securely transfer files between a server and a client, with the implemented encryption effectively safeguarding against the outlined security threats.
- **Key Findings**:
    - **Security vs. Performance**: Implementing encryption introduces a performance limit, evidenced by slower file transfer times compared to non-encrypted transfers. However, this limit is justified by the significant security benefits.
    - **Importance of Encryption Modes**: Choosing AES in EAX mode was crucial for ensuring both data confidentiality and integrity.

References showed at the bottom of each page.