Lab 1: Basic Cryptography - AES, RSA and Kyber
Computer Hardware and System Security(EECE5699)

INSTRUCTOR: Yunsi Fei, TEACHING ASSISTANT: Davis Ranney
Shaun Dsouza - 002811385

QUESTION 1:

First compare the performance of RSA, AES-128, and Kyber512 (on the same size of plaintext - 16 bytes). Choose the appropriate key size for RSA to achieve the same security level as AES- 128. How much slower is RSA/Kyber than AES? With this implementation cost, discuss what scenarios RSA, AES, and Kyber are mainly used for, respectively. Why would someone want to use Kyber rather than RSA?

ANS:

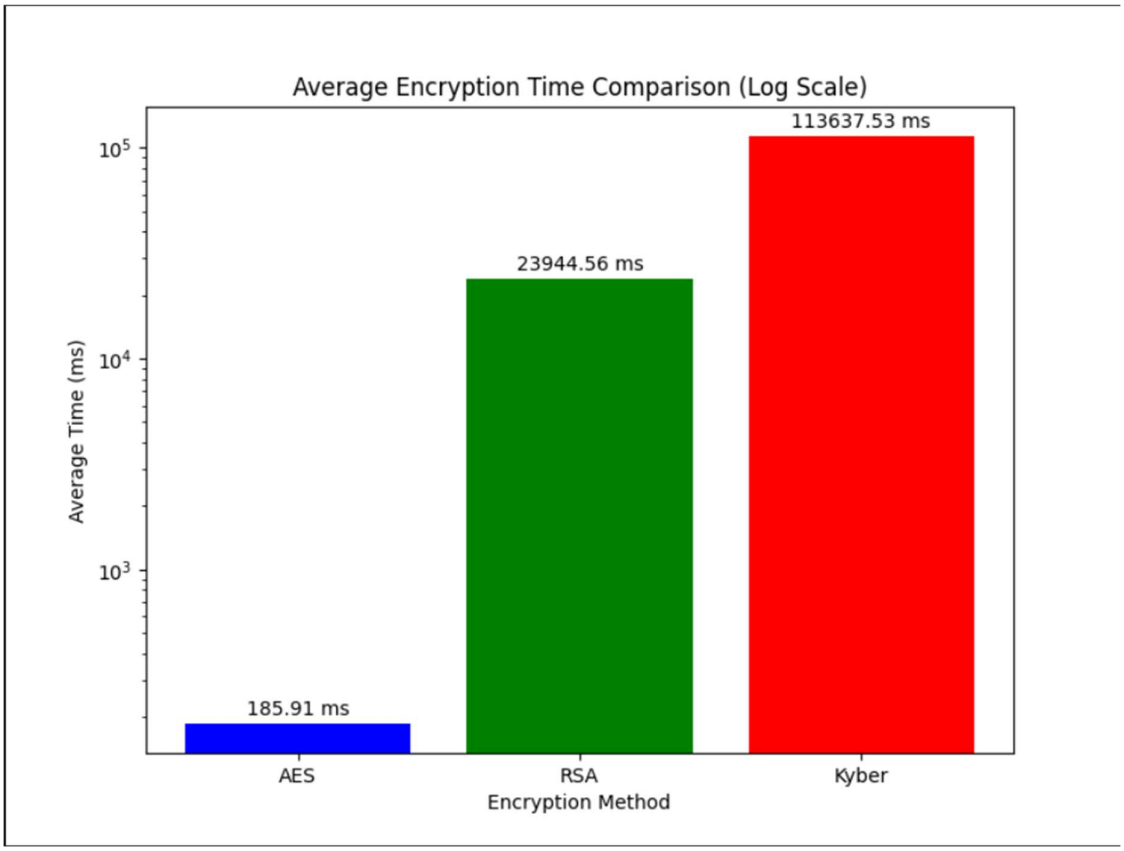

Fig.1: Graph Plot

As seen above **AES – 128 is the fastest among all the three** this is expected because AES is designed for symmetric key encryption.

RSA is significantly slower because RSA is asymmetric encryption algorithm as I have used large key size RSA's performance is much slower than AES when small plaintext like 16 bytes is used.

KYBER being a post quantum cryptography algorithm is taking the longest to encrypt 16 bytes of plaintext it is due to lattice based cryptography

Comparing all three (on the same size of plaintext - 16 bytes) :
AES is the winner in terms of performance since AES is suited for small amounts of data. RSA is taking longer than AES but is performing better than KYBER Due to asymmetric encryption. **KYBER512 is the slowest** but offers security benefits as compared to both.

```
shaundsz@SHAUN:~/lab-1-Shaundsz/submission$ python3 encryption_times.py
AES mean time: 185.91 ms
RSA mean time: 23944.56 ms
Kyber mean time: 113637.53 ms
```

Fig 2: Mean times

Appropriate key size for RSA to achieve the same security level as AES-128:
For RSA to provide equivalent security, a key size of **3072 bits** is required. RSA 3072 is even slower than RSA 2048 Wherein RSA is insufficient to handle task that require faster encryption.

On Calculating

RSA is 128.81 times slower than AES.
And KYBER is 611.16 times slower than AES

With this implementation cost, discuss what scenarios RSA, AES, and Kyber are mainly used for, respectively:
AES is used where speed and efficiency is required like secure communication for real time scenarios, video and audio streaming and **WIFI encryption**. Applications that require frequent encryption and decryption operations. It is also used for large scale data encryption where performance is very crucial. **In applications like TLS/SSL, VPN.**

RSA is used in **Key Exchange, Digital Signatures, Certificate based authentication** it is used for **email encryptions and secure key exchange** RSA provides a high level of security, especially for **authentication and secure key exchange**. However, it is computationally expensive, particularly for encryption and decryption. Thus **RSA is used primarily for securing small amounts of data rather than bulk data encryption.**

KYBER is designed to exchange keys in quantum computing world, **Digital signatures to ensure integrity and authenticity**, but with quantum resistance KYBER provides quantum resistance. Organizations can protect data for long periods in highly sensitive industries Kyber over RSA will be used Because of its **POST QUANTUM SECURITY** and also because it can **Future-proof** cryptographic systems against quantum threats.

QUESTION 2 :

Compare these two encrypted images and comment on the security. What is the downside of CBC mode in terms of performance? Suggest one operation mode you think is the best and give your reason.

Preparing the Image:

To facilitate encryption, the image is split into its header (unmodified) and body (which will be encrypted).

Extracting the Header of the image : **head -n 3 penguin.PPM > header.txt**

The head of the image is extracted and stored in a header.txt file

Extracting the body of the image : **tail -n +4 penguin.PPM > body.bin**

Encryption Modes:

ECB Mode: In this mode, each block of plaintext is encrypted independently. The command used was: **openssl enc -aes-128-ecb -nosalt -pass pass:"A" -in body.bin -out enc_body.bin**

CBC Mode: In CBC mode, each plaintext block is XORed with the ciphertext of the previous block before encryption, making it more secure.

The command used for CBC was: **openssl enc -aes-128-cbc -nosalt -pass pass:"A" -in body.bin -out enc_body.bin**

After encryption, the encrypted body was concatenated with the original header to reconstruct the encrypted image using

For ECB mode: **cat header.txt enc_body.bin > ecb_penguin.ppm**

For CBC mode: **cat header.txt enc_body.bin > cbc_penguin.ppm**

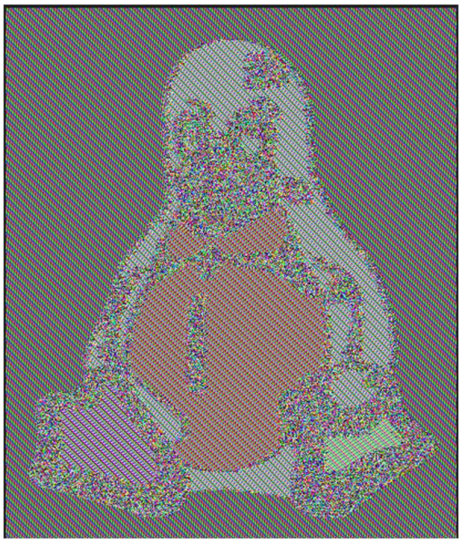

Fig 3: ECB Mode

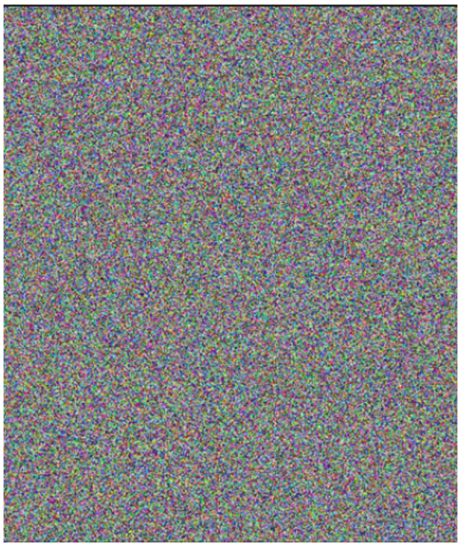

Fig 4: CBC Mode

Comparing both the Images:

On comparing the two encrypted images the **ECB mode is showing a clear pattern of a penguin** it is because the identical **plaintext blocks are encrypted into identical ciphertext blocks**. Here the patterns in the plaintext are preserved thus for repetitive data ECB is unsuitable in case of images at least (repetitive data) because it shows too much information about the underlying plaintext.
on the other hand CBC has a feedback kind of mechanism that ensures **the same plaintext block does not produce the same ciphertext block**. Basically plaintext is XOR with previous plaintext block**. If the two plaintext block is identical the result ciphertext block will differ.**
thus CBC is stronger than EBC

Downside Of CBC:

The downside of CBC lies in the performance as CBC encryption **will require the previous block to XOR with current block which will make it slower in performance** as compared to ECB. This dependence on the previous block introduces a delay if the dataset is large

Operation Mode Which is Best:

Still **CBC is best** because of security It is slower yes but the **ability to mask patterns and secure the data using cryptography** will outweight the performance cost where security is a concern . it will ensure even if the same plaintext is encrypted again and again ciphertext resulted from it will always be different.

MODULE 3: SECURE COMMUNICATION WITH RSA

In this module for Option 1 - RSA and AES is used RSA is used for public key encryption allowing secure exchange of AES key

1. **Generating RSA Key :**
   Generated the Public Key and the Private Key The server requires the RSA key pair for secure key exchange with the client.

2. **Implementing the Client and Server Protocol**
   I closely followed the client-server protocol as outlined in the provided PDF to establish secure communication between the client and server. The protocol ensures that:
   - The client generates an AES key and encrypts it using the server's RSA public key.
   - The encrypted AES key is securely sent to the server.
   - The server decrypts the AES key using its private RSA key.
   - The server then encrypts the secret message using the AES key and sends it back to the client.
   - The client decrypts the message using the shared AES key.
     The communication flow was implemented and rigorously tested on a local server setup to ensure all message exchanges and key exchanges followed the protocol securely and efficiently.

3. **Debugging and Testing the Client Code**
   - The RSA key worked as it should have the client correctly encrypted the AES key using the servers public RSA key also decrypted the AES key and used it to encrypt the message.
   - Encrypted and decrypted the message using AES.

4. **Created Makefile Using OpenSSL**
   - Specifying the OpenSSL library paths for RSA and AES functionalities.

5. **Set the Target in Makefile**
   - Make p3

   Decrypted message: Shor's algorithm

MODULE 3: QUANTUM SECURE COMMUNICATION WITH KYBER

In this module, I implemented a secure communication system utilizing Kyber for quantum-safe public key encryption and AES for secret message encryption. The Kyber encryption scheme was used to establish a shared secret between the client and server, which was then used as the key for AES encryption.

1. **Implementing the Kyber Key Exchange Protocol**
   - Kyber512 was used to encapsulate a shared secret between the client and server.
   - The client generated a Kyber public key and sent it to the server.
   - The server generated a ciphertext using the client's public key and sent it back to the client.
   - The client decapsulated the ciphertext to derive a shared secret, which was then used as the AES encryption key for secure message transmission.

2. **Generating Kyber Keys and Establishing Communication**
   - Generated the kyber key pairs
   - Clients public key sent to the server then the server responded with encrypted cyphertext using clients public key. The client decapsulated the cyphertext

3. **Debugging and Testing the Client Code**
   - Correctly sending and receiving the Kyber public key and ciphertext across the network.
   - Verifying that the shared secret derived by both the client and server matched, ensuring successful key encapsulation and decapsulation.

4. **Encrypting the Secret Message Using AES**
   - Once the Kyber shared secret was successfully derived, I used it as the AES encryption key for securely encrypting and decrypting the secret message

5. **Step 5: Creating the Makefile for Compilation**
   - make p3

Secret Key : Quantum Security