Part 1

In this part, we wrote a code for the four tasks mentioned below. The code is attached with this report. If we run the code, there will be options for all four of these.

```
What do you want to do? Type the associated number to do so.
1. AES encryption/decryption
2. RSA encryption/decryption
3. RSA Signature
4. SHA-256 hashing
```

The details for these options are stated below.

AES encryption/decryption

After selecting this option, there will be four more options like below.

```
What is your preferred mode? Type the associated number to select.
1. ECB 128 bits
2. ECB 256 bits
3. CFB 128 bits
4. CFB 256 bits
```

For all these four options, there will be again two more options like below. Also, if there is no saved key for the respective mode in the file **"data.txt"**, a key will be generated and saved in the file.

```
What do you want to do? Type the associated number to do so.
1. Encryption
2. Decryption
```

Selecting the option 1 that is Encryption, will start an input process where user can type a text to be encrypted. The ciphertext will be saved in the file **"data.txt"**.

```
Type the text to be encrypted: "........."
The encrypted text is: "........."
Encrypted in 0.0007694999999330321 seconds
```

Selecting the option 2 that is Decryption, will start the decryption proccess. Here, if there is a ciphertext saved in the file **"data.txt"**, it will be decrypted. Otherwise an error will be shown.

```
Found an encrypted text in the file. The text is: "........."
The decrypted text is: "........."
Decrypted in 0.000182699999641045 seconds
```

RSA encryption/decryption

After selecting this option, there will be two more options like below. Also, if there is no saved key in the file **"data.txt"**, a key will be generated and saved in the file.

```
What do you want to do? Type the associated number to do so.
1. Encryption
2. Decryption
```

Selecting the option 1 that is Encryption, will start an input process where user can type a text to be encrypted. The ciphertext will be saved in the file **"data.txt"**.

```
Type the text to be encrypted: "........."
The encrypted text is: "........."
Encrypted in 0.0032079000002340763 seconds
```

Selecting the option 2 that is Decryption, will start the decryption proccess. Here, if there is a ciphertext saved in the file **"data.txt"**, it will be decrypted. Otherwise an error will be shown.

```
Found an encrypted text in the file. The text is: "........."
The decrypted text is: "........."
Decrypted in 0.007560499999271997 seconds
```

RSA Signature

After selecting this option, there will be two more options like below. Also, if there is no saved key in the file **"data.txt"**, a key will be generated and saved in the file.

```
What do you want to do? Type the associated number to do so.
1. Generate signature
2. Verify
```

Selecting the option 1 that is generating a signature, will start an input process where user can type the name of the file to generate the signature. The signature will be saved in the file **"data.txt"**.

```
Enter the file name for which you want to generate the signature:
↪  "........."
The signature of the file is:  "........."
The signature generated in 0.008767600000282982 seconds
```

Selecting the option 2 that is verification, will start the verification proccess. Here, if there is a signature saved in the file **"data.txt"**, it will start an input process where user can type the name of the file to verify. Otherwise an error will be shown.

```
Enter the file name you want to verify: "........."
The signature is valid for this file.
The signature verified in 0.0018742000002021086 seconds
```

SHA-256 hashing

After selecting this option, an input process will start where user can type the name of the file to generate the SHA-256 hash. The hash will be saved in the file **"data.txt"**.

```
Enter the file name for which you want to create the hash:
↪  "........."
The hash for the file is: "........."
The hash created in 0.0005947999998170417 seconds
```

Part 2

In this part, we will compare the execution times for different cryptographic algorithm and generate graphs. The code is attached with this report. As, execution times may vary slightly at different times, we have executed those algorithms a number of times and calculated the average time from them. The result is shown below.

AES encryption/decryption

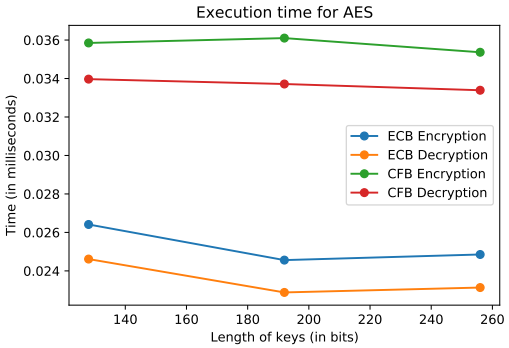

Figure 1: Running the algorithm 1000 times

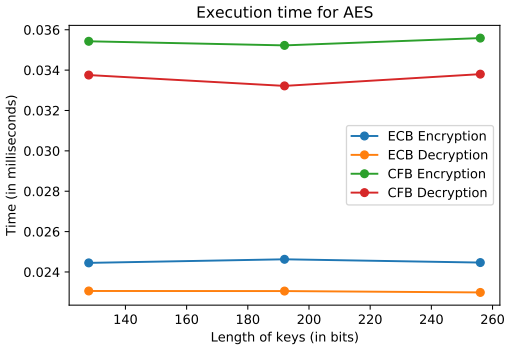

Figure 2: Running the algorithm 10000 times

From those figures, we can see that it is really hard to find a correlation between the length of keys and execution times in the two modes of AES. Sometimes the execution times increase for increasing the length of keys, sometimes they decrease. However, in theory, the execution times should always increase in accordance with the increase in key length because the number of rounds in AES increases if key length increases. But it is not the case in our experiment. It can be seen that the execution times almost even out for different modes when the number of execution becomes higher. So, we can say that in AES, the correlation between the length of keys and execution times is negligible in a practical scenario.

Also, it is evident that for the same length of keys CFB mode takes more time to execute for encryption and decryption than ECB mode. Moreover, it is clear that both modes of AES encryption take slightly more time than decryption.

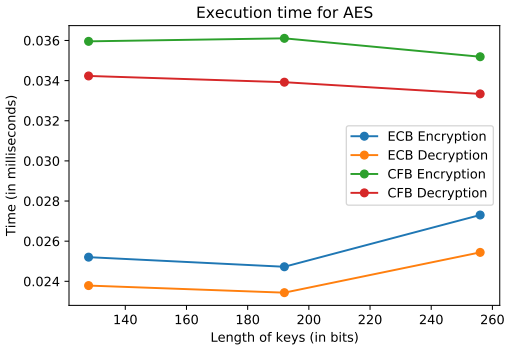

Figure 3: Running the algorithm 100000 times

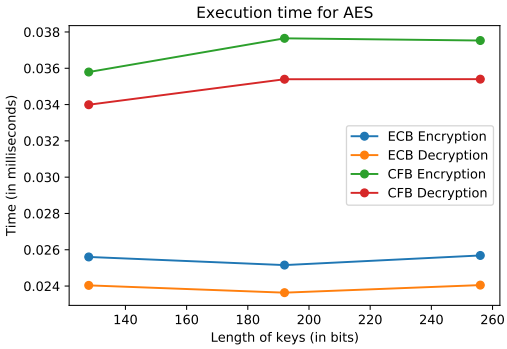

Figure 4: Running the algorithm 1000000 times

RSA encryption/decryption

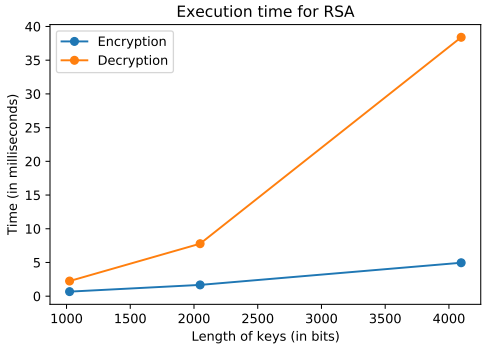

Figure 5: Running the algorithm 10 times

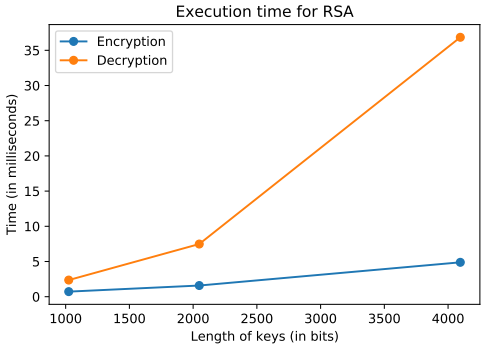

Figure 6: Running the algorithm 100 times

As RSA is a public/private key cryptosystem, it is really time-consuming. In fact, from the figures, we can see that it is about 1000 times slower than AES. As a result, running the algorithm more than 100 times at once is beyond our scope.

From the figures, it is clear that there is a correlation between the length of keys and the execution time in RSA. It can be seen that the execution times grow exponentially in accordance with the increase in key length. This is because, if the key length increases the algorithm needs to generate bigger prime numbers which are time-consuming. So, in RSA our experiment gives results similar to what should be in theory. Also, it is evident that, unlike AES, RSA actually takes more time in decryption than in encryption.

References

[1] https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html

[2] https://pycryptodome.readthedocs.io/en/latest/src/public_key/rsa.html

[3] https://pycryptodome.readthedocs.io/en/latest/src/signature/pkcs1_v1_5.html

[4] https://pycryptodome.readthedocs.io/en/latest/src/hash/sha256.html

[5] https://asecuritysite.com/encryption/aes_modes

[6] https://stackabuse.com/reading-and-writing-json-to-a-file-in-python/