



NYU

POLYTECHNIC SCHOOL
OF ENGINEERING

CS 6903 Modern Cryptography Spring 2014

Project Part: #2

Submitted by

Ishani Parikh

Kedar Parab

Madhura Pradhan

April 20, 2014

Project

Part 1. Symmetric encryption schemes:

Schemes chosen

1. **Scheme based on pseudo-random functions as in Lecture 5 (use AES in CBC mode and IV=0 as a pseudo-random function)**

The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data. It is based on Rijndael cipher. AES is based on a design principle known as a substitution-permutation network. AES can use 16, 24, or 32 byte keys (128, 192, and 256 bits respectively) and block size of 128 bits. In this scheme we have taken key size as 128 bits and initialization vector as 0 with CBC as block cipher mode.

2. **Scheme based on a block cipher (use AES) and a block cipher mode of operation (use CBC)**

Here, the key size is 128 bits. Key and iv both are generated using prng (pseudo random number generator) with CBC again as a block cipher mode.

3. **Scheme based on a block cipher (use Triple DES) and a block cipher mode of operation (use CBC)**

Triple DES-Triple Data Encryption Algorithm is a symmetric key block cipher, which applies the Data Encryption Standard cipher algorithm three times to each data block.

It can make use of 2 different keys or 3 different keys. Here, in this encryption scheme, we have used 3 different keys and Cipher Block Chaining (CBC) as a block cipher mode of operation. The three key variant has a block size of 8 bytes (64-bits) and uses a key with 24 bytes (192 bits). The DES implementation in Crypto++ ignores the parity bits (the least significant bits of each byte) in the key.

Here,

$$\text{ciphertext} = E_{K_3}(D_{K_2}(E_{K_1}(\text{plaintext})))$$
$$\text{plaintext} = D_{K_1}(E_{K_2}(D_{K_3}(\text{ciphertext})))$$

where, K1 K2 K3 are three different keys.

- **Tasks performed by each student**

Kedar Parab : aes in cbc mode and iv = 0 as pseudo random function

Ishani Parikh : aes in cbc mode

Madhura Pradhan : 3des in cbc mode

All the tasks were studied together such as using crypto++ libraries, key generation using prng, encryption and decryption for running respective schemes. Performance of schemes and compiling the report was also done together.

- **Softwares and libraries used**

We have used Crypto++ libraries and Visual Studio 2013 to implement the above schemes.

- **How to run program**

Steps to compile and integrate crypto++ library into visual studio environment:

- Download crypto++ library from the official site (Crypto++ 5.6.2) and extract the files.
- Open “cryptest” solution in the visual studio.
- Select Build << Batch Build and build cryptlib in debug and release libraries for win32.

Steps to run the sample program:

- Go to visual studio and create a new project in Win32 Console Application.
- Write the program.
- Before building the program, we need to set some paths.
- Go to the project properties << Linker << Input << Additional Dependencies and edit the path of your libraries. (Debug or Release).
- Now, again we need to go to Properties << C/C++ << Code Generation << Runtime Library and select option according to the mode.
- Build the program and then run it to give the output.

In our project, under the folder “Project Part 1” << “CryptoPP”, there are 3 folders for three schemes.

3DES_CBC

AES_CBC

AES_CBC_IV=0

All of them are provided with plaintext.txt

Steps:

- First, run KeyGen to produce key in key.txt
- Then run Encrypt to produce ciphertext in ciphertext.txt
- And then run Decrypt to reproduce plaintext in decryptedplaintext.txt

Explanation of how to run all programs to be able to generate keys, encrypted messages and decrypt ciphertexts

1. keygen.cpp

- Generate key using `prng.GenerateBlock`
- Convert it into base 16 format
- Output the key into key.txt file

2. cryptoPP.cpp

- Generate iv using `prng.GenerateBlock`
- Convert it into base 16 format and store it in variable say iv
- Read the file “key.txt” and convert the key from base16 to base64
- Perform encryption scheme
- Use `StreamTransformationFilter` to remove padding as required
- Ciphertext encoded in base16 (hexadecimal) format and stored in variable cipher
- Pre-append iv to the ciphertext and write it in readable hexadecimal ciphertext in the text file cipherntext.txt” (concat iv + ciphertext)
- Send key.txt and ciphertext.txt to receiver

3. decryptoPP.cpp

- Reading the file “key.txt”
- Reading the file “ciphertext.txt”
- Separate iv from ciphertext
- Convert key, iv and ciphertext from base16 to base64
- Perform decryption scheme
- Use `StreamTransformationFilter` to remove padding as required
- Write plaintext in the text file “decryptedplaintext.txt”

Performance analysis

This section will show the results obtained from running the program using different data loads. The results show the impact of changing data load on each algorithm.

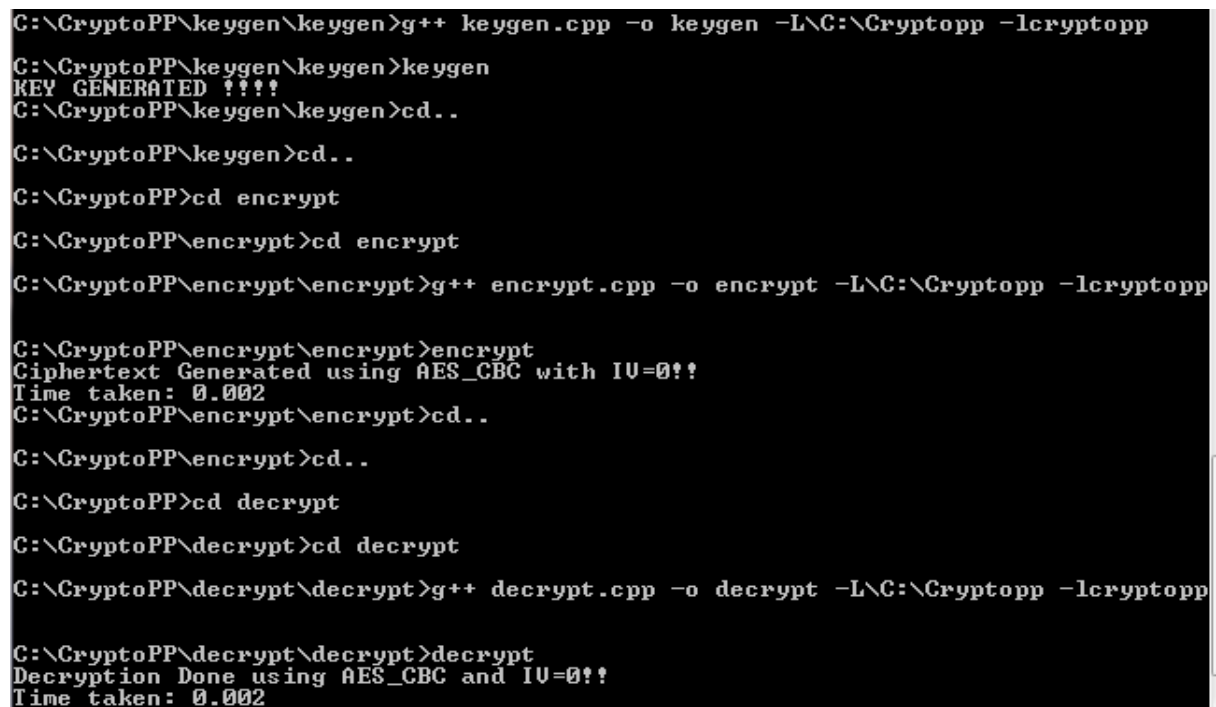
Time was observed by using the following code:

```
#include <time.h>
int main(void) {
    clock_t tStart = clock();
    /* Encryption scheme */
    cout<< "Time taken: "<< (double)(clock() - tStart)/CLOCKS_PER_SEC;
    return 0;
}
```

Here, in AES, we are taking key of size 128 bits with 10 rounds and in Triple DES, we are taking key of size 192 bits with 10 rounds. We can see that aes 128 bit is faster than triple des. Also, aes cbc (iv=0) is faster than aes cbc.

1. AES in CBC mode with IV=0.

- With input file of length 20 bytes.



```
C:\CryptoPP\keygen\keygen>g++ keygen.cpp -o keygen -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\keygen\keygen>keygen
KEY GENERATED !!!!!
C:\CryptoPP\keygen\keygen>cd ..
C:\CryptoPP\keygen>cd ..
C:\CryptoPP>cd encrypt
C:\CryptoPP\encrypt>cd encrypt
C:\CryptoPP\encrypt\encrypt>g++ encrypt.cpp -o encrypt -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\encrypt\encrypt>encrypt
Ciphertext Generated using AES_CBC with IV=0!!
Time taken: 0.002
C:\CryptoPP\encrypt\encrypt>cd ..
C:\CryptoPP\encrypt>cd ..
C:\CryptoPP>cd decrypt
C:\CryptoPP\decrypt>cd decrypt
C:\CryptoPP\decrypt\decrypt>g++ decrypt.cpp -o decrypt -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\decrypt\decrypt>decrypt
Decryption Done using AES_CBC and IV=0!!
Time taken: 0.002
```

- With input file of length 5 kb

```

C:\CryptoPP\keygen>cd keygen
C:\CryptoPP\keygen\keygen>g++ keygen.cpp -o keygen -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\keygen\keygen>keygen
KEY GENERATED !!!!!
C:\CryptoPP\keygen\keygen>cd..
C:\CryptoPP\keygen>cd..
C:\CryptoPP>cd encrypt
C:\CryptoPP\encrypt>cd encrypt
C:\CryptoPP\encrypt\encrypt>g++ encrypt.cpp -o encrypt -L\C:\Cryptopp -lcryptopp

C:\CryptoPP\encrypt\encrypt>encrypt
Ciphertext Generated using AES_CBC with IV=0!!
Time taken: 0.006
C:\CryptoPP\encrypt\encrypt>cd..
C:\CryptoPP\encrypt>cd..
C:\CryptoPP>cd decrypt
C:\CryptoPP\decrypt>cd decrypt
C:\CryptoPP\decrypt\decrypt>g++ decrypt.cpp -o decrypt -L\C:\Cryptopp -lcryptopp

C:\CryptoPP\decrypt\decrypt>decrypt
Decryption Done using AES_CBC and IV=0!!
Time taken: 0.005

```

- With input file of length 428 kb.

```

C:\CryptoPP\keygen\keygen>g++ keygen.cpp -o keygen -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\keygen\keygen>keygen
KEY GENERATED !!!!!
C:\CryptoPP\keygen\keygen>cd..
C:\CryptoPP\keygen>cd..
C:\CryptoPP>cd encrypt
C:\CryptoPP\encrypt>cd encrypt
C:\CryptoPP\encrypt\encrypt>g++ encrypt.cpp -o encrypt -L\C:\Cryptopp -lcryptopp

C:\CryptoPP\encrypt\encrypt>encrypt
Ciphertext Generated using AES_CBC with IV=0!!
Time taken: 0.177
C:\CryptoPP\encrypt\encrypt>cd..
C:\CryptoPP\encrypt>cd..
C:\CryptoPP>cd decrypt
C:\CryptoPP\decrypt>cd decrypt
C:\CryptoPP\decrypt\decrypt>g++ decrypt.cpp -o decrypt -L\C:\Cryptopp -lcryptopp

C:\CryptoPP\decrypt\decrypt>decrypt
Decryption Done using AES_CBC and IV=0!!
Time taken: 0.287

```

2. AES in CBC mode

- With input file of length 20 bytes.

```
C:\CryptoPP\keygen\keygen>g++ keygen.cpp -o keygen -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\keygen\keygen>keygen
KEY GENERATED !!!!
C:\CryptoPP\keygen\keygen>cd..
C:\CryptoPP\keygen>cd..
C:\CryptoPP>cd encrypt
C:\CryptoPP\encrypt>cd encrypt
C:\CryptoPP\encrypt\encrypt>g++ encrypt.cpp -o encrypt -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\encrypt\encrypt>encrypt
AES_CBC Ciphertext Generated!!
Time taken: 0.011
C:\CryptoPP\encrypt\encrypt>cd..
C:\CryptoPP\encrypt>cd..
C:\CryptoPP>cd decrypt
C:\CryptoPP\decrypt>cd decrypt
C:\CryptoPP\decrypt\decrypt>g++ decrypt.cpp -o decrypt -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\decrypt\decrypt>decrypt
AES_CBC Decryption Done!!
Time taken: 0.004_
```

- With input file of length 5 kb.

```
C:\CryptoPP\keygen\keygen>g++ keygen.cpp -o keygen -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\keygen\keygen>keygen
KEY GENERATED !!!!
C:\CryptoPP\keygen\keygen>cd..
C:\CryptoPP\keygen>cd..
C:\CryptoPP>cd encrypt
C:\CryptoPP\encrypt>cd encrypt
C:\CryptoPP\encrypt\encrypt>g++ encrypt.cpp -o encrypt -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\encrypt\encrypt>encrypt
AES_CBC Ciphertext Generated!!
Time taken: 0.01
C:\CryptoPP\encrypt\encrypt>cd..
C:\CryptoPP\encrypt>cd..
C:\CryptoPP>cd decrypt
C:\CryptoPP\decrypt>cd decrypt
C:\CryptoPP\decrypt\decrypt>g++ decrypt.cpp -o decrypt -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\decrypt\decrypt>decrypt
AES_CBC Decryption Done!!
Time taken: 0.004
```

- With input file of length 428 kb.

```

C:\CryptoPP\keygen>cd keygen
C:\CryptoPP\keygen\keygen>g++ keygen.cpp -o keygen -L\C:\Cryptopp -lcryptopp
C:\CryptoPP\keygen\keygen>keygen
KEY GENERATED ???
C:\CryptoPP\keygen\keygen>cd..
C:\CryptoPP\keygen>cd..
C:\CryptoPP>cd encrypt
C:\CryptoPP\encrypt>cd encrypt
C:\CryptoPP\encrypt\encrypt>g++ encrypt.cpp -o encrypt -L\C:\Cryptopp -lcryptopp

C:\CryptoPP\encrypt\encrypt>encrypt
AES_CBC Ciphertext Generated!!
Time taken: 0.186
C:\CryptoPP\encrypt\encrypt>cd..
C:\CryptoPP\encrypt>cd..
C:\CryptoPP>cd decrypt
C:\CryptoPP\decrypt>cd decrypt
C:\CryptoPP\decrypt\decrypt>g++ decrypt.cpp -o decrypt -L\C:\Cryptopp -lcryptopp

C:\CryptoPP\decrypt\decrypt>decrypt
AES_CBC Decryption Done!!
Time taken: 0.212_

```

3. Triple DES in CBC mode.

- With input file of length 20 bytes.

```

C:\Windows\system32\cmd.exe - decrypt
C:\CryptoPP\keygen\keygen>keygen
KEY GENERATED ???
C:\CryptoPP\keygen\keygen>cd..
C:\CryptoPP\keygen>cd..
C:\CryptoPP>cd encrypt
C:\CryptoPP\encrypt>cd encrypt
C:\CryptoPP\encrypt\encrypt>g++ encrypt.cpp -o encrypt -L\C:\Cryptopp -lcryptopp

C:\CryptoPP\encrypt\encrypt>encrypt
3DES Ciphertext Generated!!
Time taken: 0.013
C:\CryptoPP\encrypt\encrypt>cd..
C:\CryptoPP\encrypt>cd..
C:\CryptoPP>cd decrypt
C:\CryptoPP\decrypt>cd decrypt
C:\CryptoPP\decrypt\decrypt>g++ decrypt.cpp -o decrypt -L\C:\Cryptopp -lcryptopp

C:\CryptoPP\decrypt\decrypt>decrypt
3DES Decryption Done!!
Time taken: 0.002

```

- With input file of length 5 kb.


```

C:\CryptoPP\keygen\keygen>keygen
KEY GENERATED !!!!
C:\CryptoPP\keygen\keygen>cd..
C:\CryptoPP\keygen>cd..
C:\CryptoPP>cd encrypt
C:\CryptoPP\encrypt>cd encrypt
C:\CryptoPP\encrypt\encrypt>g++ encrypt.cpp -o encrypt -L\C:\Cryptopp -lcryptopp

C:\CryptoPP\encrypt\encrypt>encrypt
3DES Ciphertext Generated!!
Time taken: 0.011
C:\CryptoPP\encrypt\encrypt>cd..
C:\CryptoPP\encrypt>cd..
C:\CryptoPP>cd decrypt
C:\CryptoPP\decrypt>cd decrypt
C:\CryptoPP\decrypt\decrypt>g++ decrypt.cpp -o decrypt -L\C:\Cryptopp -lcryptopp

C:\CryptoPP\decrypt\decrypt>decrypt
3DES Decryption Done!!
Time taken: 0.006_

```

- With input file of length 428 kb.

```

C:\CryptoPP\keygen\keygen>keygen
KEY GENERATED !!!!
C:\CryptoPP\keygen\keygen>cd..
C:\CryptoPP\keygen>cd..
C:\CryptoPP>cd encrypt
C:\CryptoPP\encrypt>cd encrypt
C:\CryptoPP\encrypt\encrypt>g++ encrypt.cpp -o encrypt -L\C:\Cryptopp -lcryptopp

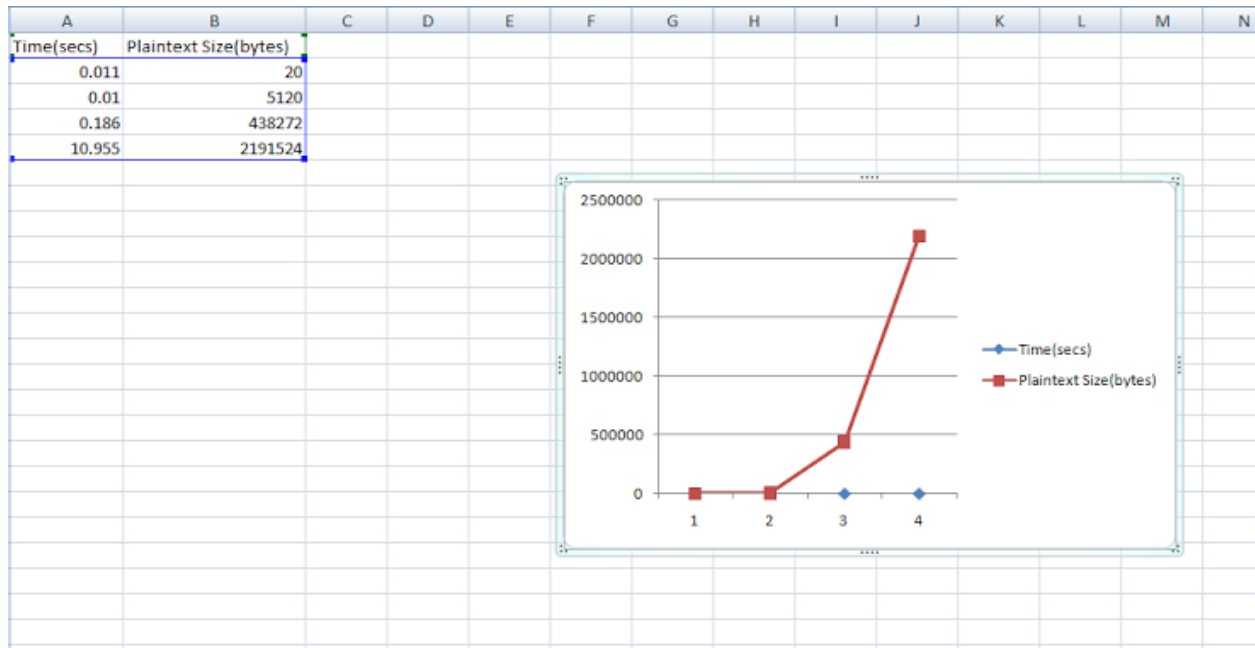
C:\CryptoPP\encrypt\encrypt>encrypt
3DES Ciphertext Generated!!
Time taken: 0.204
C:\CryptoPP\encrypt\encrypt>cd..
C:\CryptoPP\encrypt>cd..
C:\CryptoPP>cd decrypt
C:\CryptoPP\decrypt>cd decrypt
C:\CryptoPP\decrypt\decrypt>g++ decrypt.cpp -o decrypt -L\C:\Cryptopp -lcryptopp

C:\CryptoPP\decrypt\decrypt>decrypt
3DES Decryption Done!!
Time taken: 0.254_

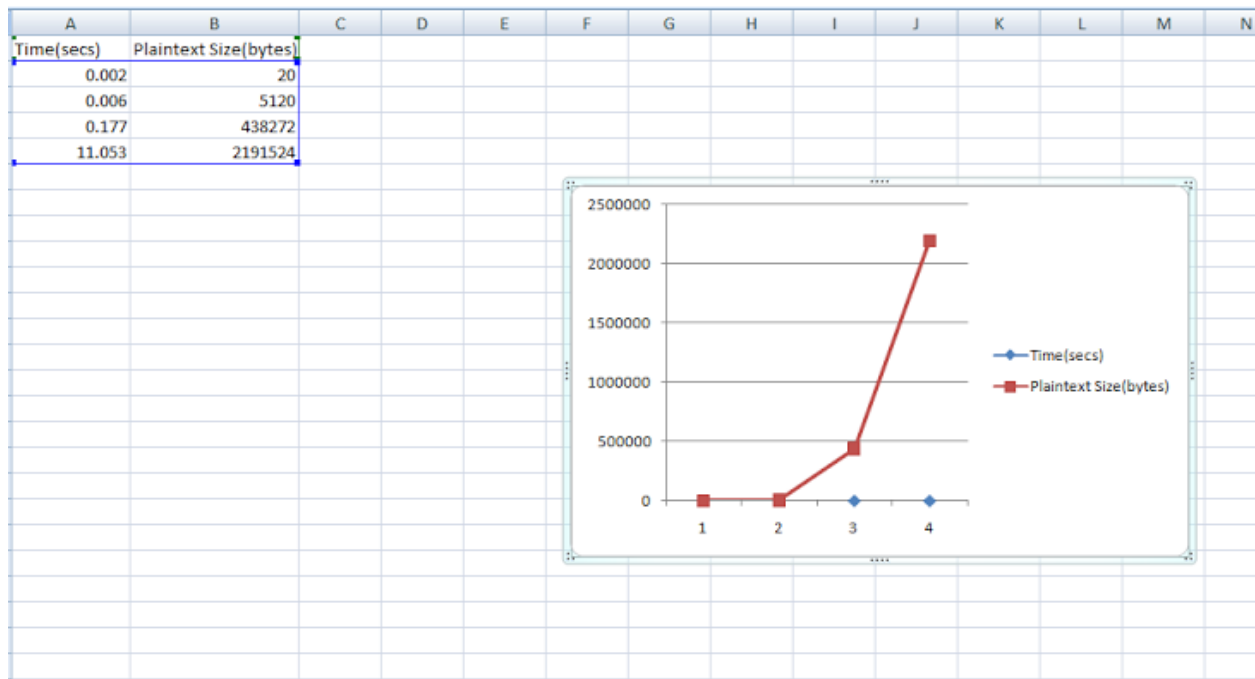
```

6. Simulation Results

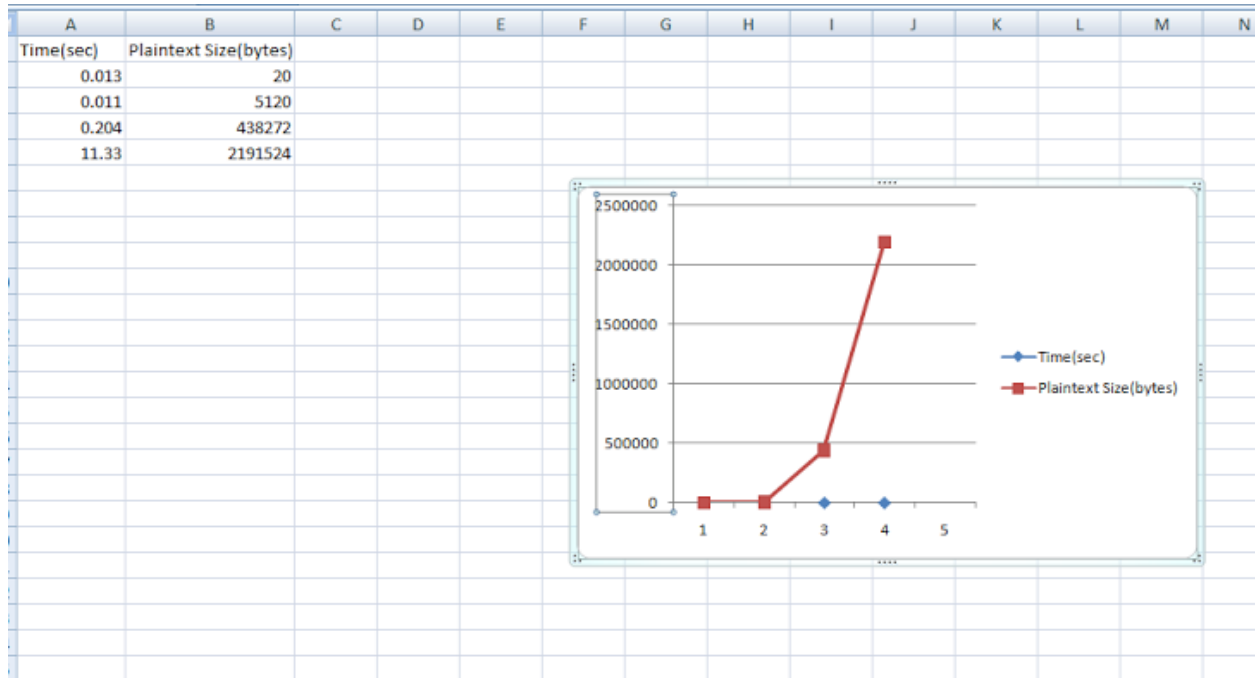
6.1 Performance Results with AES in CBC mode with IV=0



6.2 Performance Results with AES in CBC mode.



6.3 Performance Results with Triple DES in CBC mode.



AES does run faster than 3DES on comparable hardware. This is especially true when pure software encryption is used.

Part 2. Message Authentication and improved symmetric encryption

Schemes implemented:

Scheme based on pseudo-random functions as in Lecture 5 (use AES in CBC mode and IV=0 as a pseudo-random function)

We have implemented these schemes along with timestamping + message authentication

1) Timestamping and F-CBC-MAC (use F = AES) :

In this scheme, we perform AES encryption scheme (In CBC mode) on the last block of ciphertext to generate the MAC and also calculate the timestamping. Stepwise representation is given as follows:

Key Generation block:

- Keys k1 and k2 are generated in the key generation code. It is then concatenated with each other, written in the file “F_CBC_MAC_key.txt” and sent to the encryption and decryption blocks.

Encryption block:

- Individual keys are extracted in the encryption block.
- Plaintext is read and encoded in base16 form. It is then converted to **ciphertext** using key k1 and iv = 0.
- Ciphertext is encoded in the base16 form.
- The last block (block size = 128 bits) is then extracted from the ciphertext and converted to ascii format.
- Encryption is again carried out with key k2 and iv = 0 to compute the **MAC** and encoded to base16 form.
- Current local time and date are generated (**Timestamp**) and encrypted with key k1.
- Encrypted Timestamp, ciphertext and MAC are concatenated to form data which is written in the file **F_CBC_MAC_ciphertext.txt**.

Decryption block:

- In this block, the keys are again extracted individually.
- Ciphertext is read from the file **F_CBC_MAC_ciphertext.txt**. Timestamp, ciphertext and MAC are extracted from it.
- Timestamp is decrypted and it is checked if recent.
- If the timestamp is recent, MAC is again computed using key k2 and iv = 0 to compare it with the received MAC.
- If the message is verified, decryption is carried out and the plaintext is written in the output file **F_CBC_MAC_decryptedplaintext.txt**.

2)Timestamping and HMAC (H = SHA2) :

In this scheme, we calculate HMAC using SHA2 (output = 256 bits) and also calculate the timestamping. Stepwise representation is given as follows:

Key Generation block:

- Keys key and hmakey are generated in the key generation code. It is then concatenated with each other, written in the file “**HMAC_SHA2_key.txt**” and sent to the encryption and decryption blocks.

Encryption block:

- Current local time and date are generated (**Timestamp**) and encoded to base16 form.
- Individual keys are extracted in the encryption block.
- Plaintext is then read and converted to **ciphertext** using key k1 and iv = 0.
- Ciphertext and timestamp are then concatenated to compute the HMAC with hmakey and iv = 0.
- Timestamp, ciphertext and HMAC are concatenated to form data which is written in the file **HMAC_SHA2_ciphertext.txt**.

Decryption block:

- In this block, the keys are again extracted individually.
- Ciphertext is read from the file **HMAC_SHA2_ciphertext.txt**. Timestamp, ciphertext and MAC are extracted from it.
- Timestamp is converted in the byte format and is checked if recent.
- HMAC is again computed using hmakey and iv = 0 to compare it with the received HMAC using HashVerificationFilter.
- IF the message is verified, decryption is carried out and the plaintext is written in the output file **HMAC_SHA2_decryptedplaintext.txt**.

3) Timestamping and F-CBC-MAC (use F = AES) :

In this scheme, we perform AES encryption scheme (In CBC mode) on the ciphertext produced from the encryption using key k1 and iv = 0. The 2nd encryption uses key k2 for the computation of ciphertext i.e. MAC and also calculate the timestamp. Stepwise representation is given as follows:

Key Generation block:

- Keys k1 and k2 are generated in the key generation code. It is then concatenated with each other, written in the file “**PRF_MAC_key.txt**” and sent to the encryption and decryption blocks.

Encryption block:

- Individual keys are extracted in the encryption block.
- Plaintext is read and encoded in base16 form. It is then converted to **ciphertext** using key k1 and iv = 0.
- Encryption scheme is again carried out on ciphertext with key k2 and iv = 0 to compute the **MAC**.
- Current local time and date are generated (**Timestamp**) and encrypted using key k1.
- Encrypted Timestamp, ciphertext and MAC are concatenated to form data which is written in the file **PRF_MAC_ciphertext.txt**.

Decryption block:

- In this block, the keys are again extracted from the file **PRF_MAC_key.txt**.
- Ciphertext is read from the file ciphertext.txt. Timestamp, ciphertext and MAC are extracted from it.
- Timestamp is converted in the byte format and is checked if recent.
- Encryption is carried out on the received ciphertext to compute the MAC which is then compared with received MAC.
- IF both the MACs match, the message is verified. Decryption is carried out and the plaintext is written in the output file **PRF_MAC_decryptedplaintext.txt**.

Tasks performed by each student

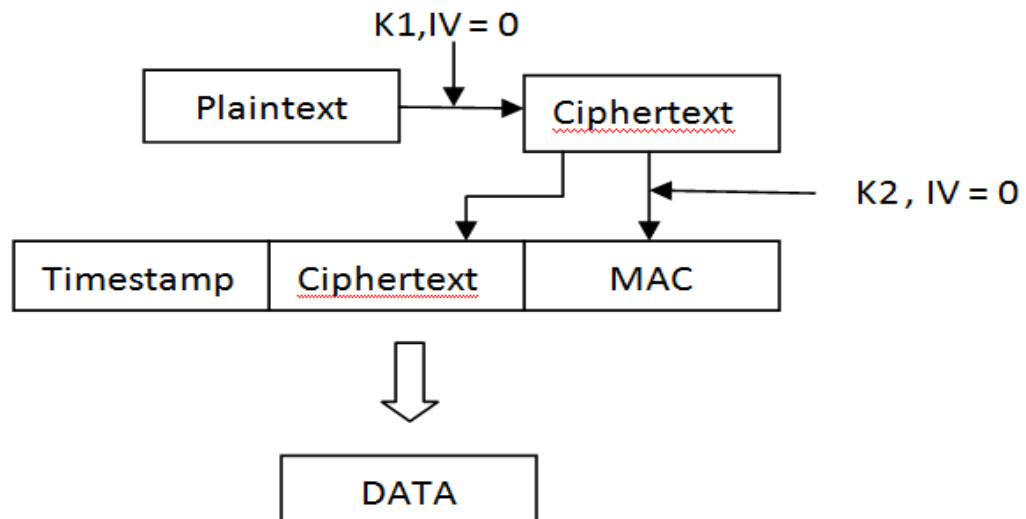
Ishani Parikh : timestamping and F-CBC-MAC (use F = AES)

Kedar Parab : timestamping and HMAC where H=SHA2

Madhura Pradhan : timestamping and the MAC based on pseudo-random functions

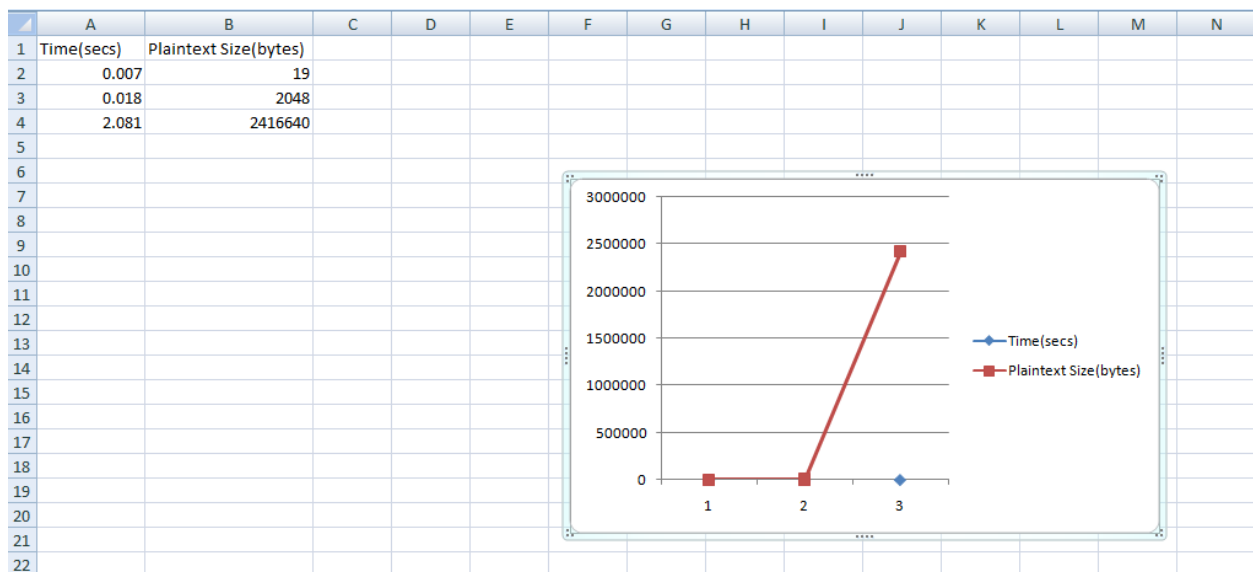
Insightful Performance Analysis

1. Pseudorandom MAC

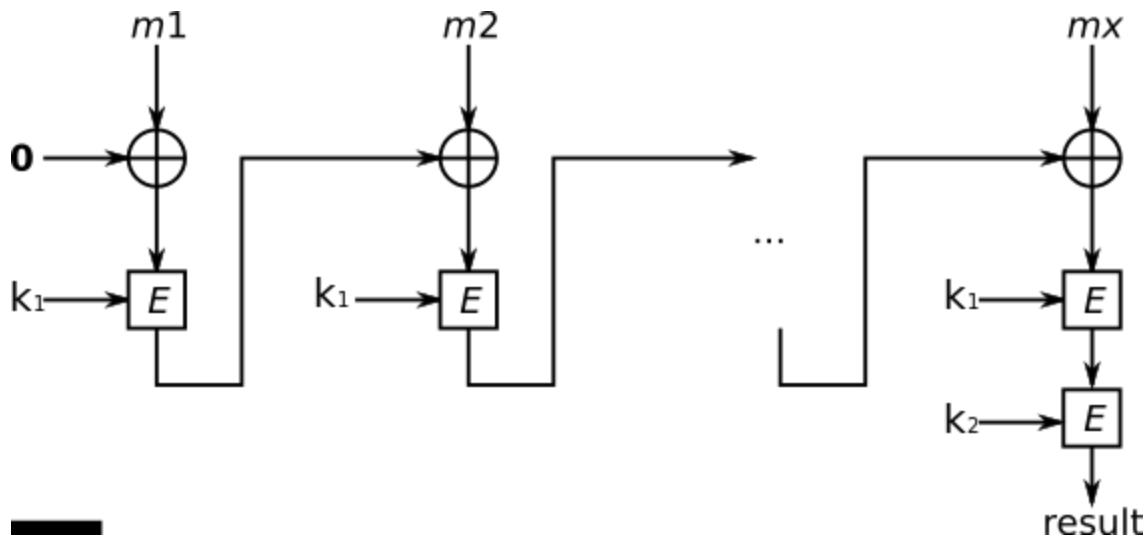


- In Pseudorandom MAC, encryption is performed with key k_2 and $iv = 0$ and corresponding MAC is computed. It is then concatenated with timestamp and ciphertext to form data which is sent to decryption block.
- If the plaintext is of large size, corresponding ciphertext and MAC will be larger which will require twice the amount of space.
- It also takes more time to compute the mac comparing with other 2 schemes

Simulation result for this scheme is shown below:

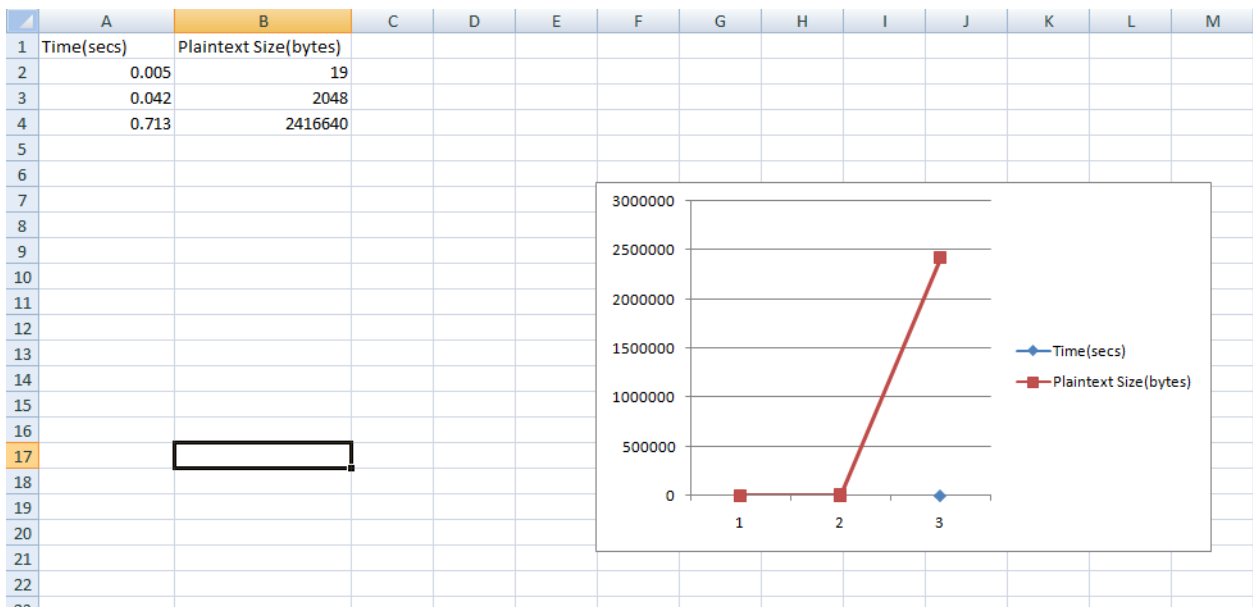


2. F-CBC-MAC

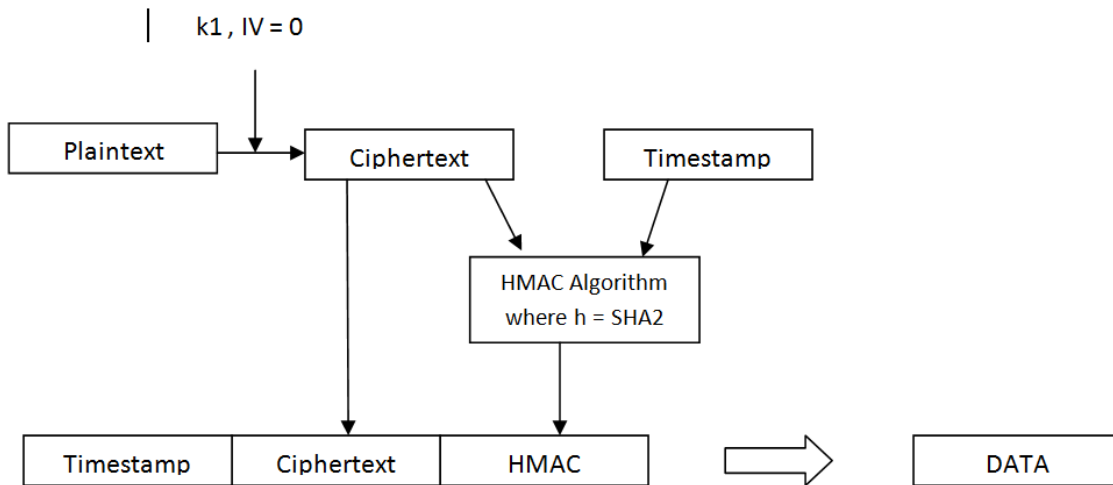


- Here, message m (comprising of blocks $m1 \parallel m2 \parallel \dots \parallel mx$ of size 128 bits) is encrypted in CBC mode, with initialization vector 0 and key $k1$, where F can be any encryption scheme.
Encryption is again carried out on the ciphertext of last block mx with key $k2$.
Resulting ciphertext is the MAC.
- Here, space required to send the message is less than that of Pseudorandom MAC.

Simulation result for this scheme is shown below:

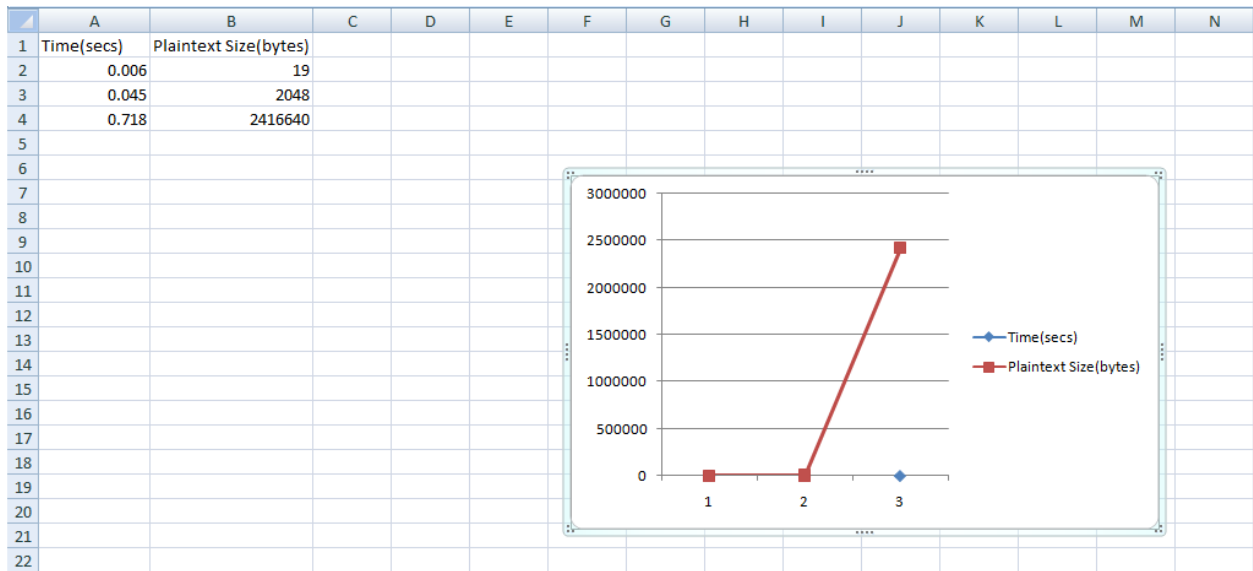


3. HMAC-SHA2



- In this scheme, we concatenate ciphertext and timestamp and perform SHA2 hash function with key hmackey. Resulting HMAC is then appended to timestamp and ciphertext to form data. This data is then sent to the decryption block.
- In SHA2 hash function, the output remains the same (output = 256 bits) for any size of plaintext.
- Unlike F-CBC-MAC , HMAC is a one-way compression function that transforms the input into fixed length output i.e it is difficult to get back the input if the fixed length hash output is provided.
- It is secure against variable length messages too.

Simulation result for this scheme is shown below:

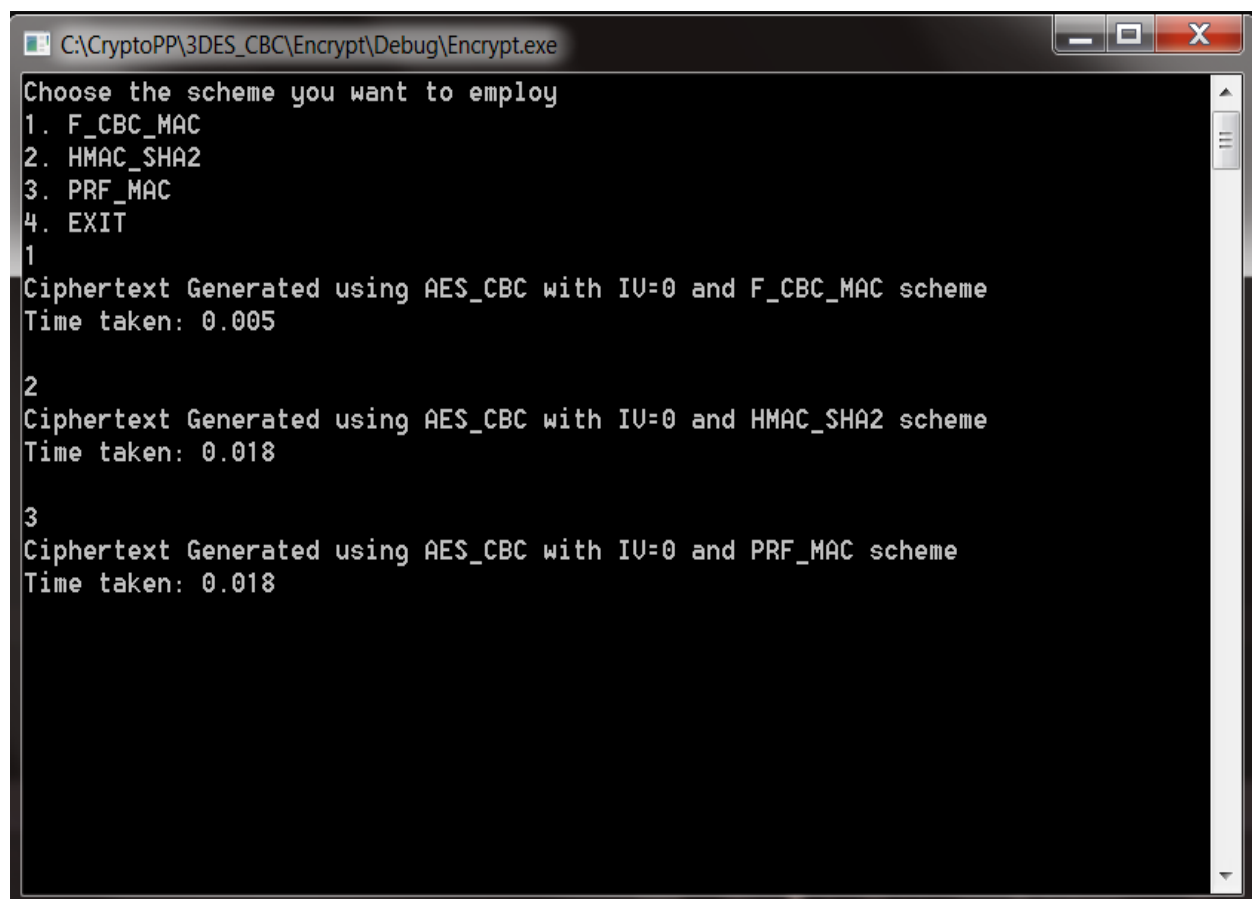


Test cases and screenshots:

1. KeyGen.exe

```
C:\CryptoPP\3DES_CBC\KeyGen\Debug\KeyGen.exe
F_CBC_MAC KEY GENERATED !!!!!
HMAC_SHA2 KEY GENERATED !!!!!
PRF_MAC KEY GENERATED !!!!!
```

2. Encrypt.exe



```
C:\CryptoPP\3DES_CBC\Encrypt\Debug\Encrypt.exe
Choose the scheme you want to employ
1. F_CBC_MAC
2. HMAC_SHA2
3. PRF_MAC
4. EXIT
1
Ciphertext Generated using AES_CBC with IV=0 and F_CBC_MAC scheme
Time taken: 0.005
2
Ciphertext Generated using AES_CBC with IV=0 and HMAC_SHA2 scheme
Time taken: 0.018
3
Ciphertext Generated using AES_CBC with IV=0 and PRF_MAC scheme
Time taken: 0.018
```

3. Decrypt.exe

```
C:\CryptoPP\3DES_CBC\Decrypt\Debug\Decrypt.exe
Choose the scheme you want to employ
1. F_CBC_MAC
2. HMAC_SHA2
3. PRF_MAC
4. EXIT
1
Timestamp Sun Apr 20 20:24:30 2014
Received F_CBC_MAC tag : D82D53B2DEE90DD5D87F64F99E24CCB6
tag generated from the ciphertext : D82D53B2DEE90DD5D87F64F99E24CCB6
Tag verified
Decryption Done using AES_CBC and IV=0!!

2
Timestamp Sun Apr 20 20:23:45 2014
Received HMAC tag: FE01811DA3C41BE60A68BD3B9F28BB564EFFC7C9E1928F9C3452E3A3611A2
4FA
tag generated from ciphertext: FE01811DA3C41BE60A68BD3B9F28BB564EFFC7C9E1928F9C3
452E3A3611A24FA
Tag verified.
Decryption Done using AES_CBC and IV=0!!

3
Timestamp Sun Apr 20 20:23:46 2014
Received PRF_MAC tag 6D86D1F672A29525DADBCEB1BE8FA18765F14692D38AD49BFE28C787131
3BBEC73EB75D318EBBF965B854F17BB44A13A
Tag generated from ciphertext: 6D86D1F672A29525DADBCEB1BE8FA18765F14692D38AD49BF
E28C7871313BBEC73EB75D318EBBF965B854F17BB44A13A
Tag verified
Decryption Done using AES_CBC and IV=0!!
```

We have also provided a Readme.docx file.