Đại học Quốc gia Thành phố Hồ Chí Minh
Trường Đại học Công nghệ Thông tin (UIT)

BÁO CÁO BÀI TẬP

Môn học: Mật mã học

Kỳ báo cáo: Buổi 02 (Session 02)

Tên chủ đề: Thuật toán mã hóa AES

Ngày báo cáo: 29/03/2023

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)
Lớp: NT219.N21.ANTN

STT	Họ và tên	MSSV	Email
1	Phạm Nguyễn Hải Anh	21520586	21520586@gm.uit.edu.vn
2	Nguyễn Nhật Quân	21522497	21522497@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá	Người đóng góp
1	Câu hỏi 01	100%	Nhật Quân
2	Câu hỏi 2		Hải Anh (phần lý luận đầu) Nhật Quân (phần code đánh giá)
3	Kịch bản 03	100%	Nhật Quân
4	Câu hỏi 04	40%	Hải Anh
5	Câu hỏi 05	40%	Hải Anh
6	Câu hỏi 6		Hải Anh (4 mode đầu, trừ CBC) Nhật Quân (4 mode sau)
7	Câu hỏi 7	30%	Hải Anh
8	Bài luyện tập 1		Quân
9	Bài luyện tập 3	70%	Hải Anh

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

1. Câu hỏi 01

Base on benchmarks file of AES algorithm, I wrote DES benchmarks program, I just changed key size from 16 byte to 8 byte (or DES::DEFAULT_KEYLENGTH) and iv size form 16 byte to 8 byte (or DES::BLOCKSIZE), and set CBC_Mode struct and DES class with Encryption typedef for cipher variable, these steps are actualized as below code:

```
SecByteBlock key(DES::DEFAULT_KEYLENGTH);
prng.GenerateBlock(key, key.size());

byte iv[DES::BLOCKSIZE];
prng.GenerateBlock(iv, sizeof(iv));

CBC_Mode< DES >::Encryption cipher;
cipher.SetKeyWithIV(key, key.size(), iv);
```

I used buf's size is 2048 for both DES and AES program.

After this loop, we can see the result of DES and AES program

```
do
{
    blocks *= 2;
    for (; i<blocks; i++)
        cipher.ProcessString(buf, BUF_SIZE);
    elapsedTimeInSeconds = timer.ElapsedTimeAsDouble();
}
while (elapsedTimeInSeconds < runTimeInSeconds);
```

Speed of AES encryption:

```
quanp  Cryptography_Lab2  4.292s  .\AES\benchmark_encrypt\benchmark_encrypt.exe
AES/CBC benchmarks...
  2.7 GHz cpu frequency
  1.3948 cycles per byte (cpb)
  1846.08 MiB per second (MiB)
quanp  Cryptography_Lab2  4.841s
```

Speed of DES encryption:

```
quanp  Cryptography_Lab2  4.841s  .\DES\benchmark_encrypt\benchmark_encrypt.exe
DES/CBC benchmarks...
  2.7 GHz cpu frequency
  27.0316 cycles per byte (cpb)
  95.2558 MiB per second (MiB)
quanp  Cryptography_Lab2  5.645s
```

the cycles per byte (cpb) measure the number of clock cycles required for the processor to encrypt one byte of data.

the MiB per second (MiB/s) metric can be used to measure the encryption speed of a system

In conclusion, cpb of AES is lower than DES's, this means AES can encrypt more data in a shorter amount of time than DES. MiB of AES is higher than DES's, a higher MiB/s value

indicates that more data can be encrypted in a shorter amount of time. It can be said that AES encryption is better than DES

2. Câu hỏi 02:

Edit the benchmarks.cpp sample to evaluate the AES debugging. First, I create CBC_AES encryption object and CBC_AES decryption object.

```
CBC_Mode< AES >::Encryption e;
e.SetKeyWithIV(key, sizeof(key), iv);
```

```
CBC_Mode< AES >::Decryption d;
d.SetKeyWithIV(key, sizeof(key), iv);
```

Then encrypt with only 1 time, no loop and read it:

```
// Encrypt the plaintext "buf" >> "buf"
e.ProcessString(buf, BUF_SIZE);

// Pretty print cipher
encoded.clear();
StringSource(buf, sizeof(buf), true,
        new HexEncoder(
            new StringSink(encoded)
        ) // HexEncoder
    ); // StringSource
cout << "cipher text: " << encoded << endl;
```

and decrypt with only 1 time, and read it:

```
// Pretty Print decrypt
d.ProcessString(buf, BUF_SIZE);
encoded.clear();
StringSource(buf, sizeof(buf), true,
        new HexEncoder(
            new StringSink(encoded)
        ) // HexEncoder
    ); // StringSource
cout << "recovered text: " << encoded << endl;
```

It works. Recovered text = plaintext:

```
PS D:\Move\UIT\HK4\MMH\ThucHanh\Lab2\Code> .\bt2.exe
key: 500151F0950C7612CDD0D9C8512BA5DD
iv: 674292FEACFE182092F8BF6E2B69EE54
plain text: F944B31F54881A93C36A5991EA87C86D357FC5F7604E22F7553F8254360E2EE3
cipher text: DFFDF39748315B8F4716A7E44EA32D783C1DCC2C0B9A952BD064294D4547C188
recovered text: F944B31F54881A93C36A5991EA87C86D357FC5F7604E22F7553F8254360E2EE3
```

However, if there are more than one time encryption, for example:

```
// Encrypt the plaintext "buf" >> "buf"
e.ProcessString(buf, BUF_SIZE);
e.ProcessString(buf, BUF_SIZE);
```

and decryp 2 times:

```
// Pretty Print decrypt
d.ProcessString(buf, BUF_SIZE);
d.ProcessString(buf, BUF_SIZE);
```

Now the recovered text and plaintext is no longer the same:

```
PS D:\Move\UIT\HK4\MMH\ThucHanh\Lab2\Code> .\bt2.exe
key: CCCBE7798F418ABA7A75E52187B2877D
iv: 7A650F3B14BB802628C045D0AEBFF897
plain text: 53E126FE2F39E7E9E613BB22F50E546DA82B7773B634213FB3A3553FF3D96301
cipher text: 35C86454DA0D5D6D190375C4DFC64AD630696FF8A2719F321649577443048572
recovered text: B478B323D01F7BCDF2AC4A5F5B37DA014323A5E10F4D9B9FCAF86A8A8A10C06E
```

But the execise just want the evalutation of the decryption, so the recovered message is not really important.

After that, we can compare speed of AES and DES decryption

Speed of AES decryption:

```
95.2558 MiB per second (MiB)
quanp  Cryptography_Lab2  45.645s  .\AES\benchmark_decrypt\benchmark_decrypt.exe
cipher text: 9557C991DA4FC23E66979245CAF9A6754A0AFFDB3C42580405546A1783CF3CA3
recovered text: D40ABF8CD66D43B7411E5B770CDF30D13262AC5F9EBCABE2335E3C2AC97CD1CD
AES/CBC benchmarks...
  2.7 GHz cpu frequency
  0.318005 cycles per byte (cpb)
  8097.11 MiB per second (MiB)
```

Speed of DES decryption:

```
quanp  Cryptography_Lab2  44.409s  .\DES\benchmark_decrypt\benchmark_decrypt.exe
DES/CBC benchmarks...
  2.7 GHz cpu frequency
  24.8314 cycles per byte (cpb)
  103.696 MiB per second (MiB)
quanp  Cryptography_Lab2  45.383s
```

Like exercise 1, we can see some proofs that AES is better than DES

cpb'AES < cpb'DES, MiB's AES > MiB's DES

3. Kịch bản 03

Use some libraries like this:

```
#include <locale>
#include <codecvt>
#include <string>
#include <io.h>
#include <fcntl.h>
#include <stdio.h>
```

The main idea is using the 'wstring' datatype which supports UTF-16 for input then converting 'wstring' to 'string' to encrypt the plaintext. The last step is converting the 'string' recovered text decrypted from ciphertext to 'wstring' to display UTF-16 output.

Now, let's go through the details:

a. The first thing we need is declaring the variables

```
std::wstring wplain = L"你好";
std::wstring_convert<std::codecvt_utf8<wchar_t>> converter;
std::string plain = converter.to_bytes(wplain);


string cipher, encoded, recovered;
std::wstring wencoded;
```

We have both wstring and string datatype. To convert wstring to string datatype, I have used 'wstring_convert' and 'codecvt_utf8' class to create converter. After using it, I have had 'plain' string datatype converted from wstring

b. Then, we just use the DES algorithm to encrypt the plaintext.
c. In addition, we need to use _setmode() funtion with 'stdout' mode and 'std::wcout' to display our UTF-16 paintext.

```
_setmode(_fileno(stdout), _O_U16TEXT); // <=== Windows madness
try
{
    std::wcout << "plain text: " << wplain << std::endl;
```

d. At the last step, I used 'wstring_convert' and 'codecvt_utf8_utf16' class to create converter1 and used it to convert cipher text in 'encoded' variable and recovered text to UTF-16 and display them with wcout.

```
); // StringSource
std::wstring_convert<std::codecvt_utf8_utf16<wchar_t>> converter1;
wencoded = converter1.from_bytes(encoded);

std::wcout << "cipher text: " << wencoded << endl;
```

```
    std::wstring wrecovered = converter1.from_bytes(recovered);

    std::wcout << "recovered text: " << wrecovered << endl;
```

(anything printed by 'cout' below _setmode() function can not be displayed on terminal so that I have to also convert ciphertext to UTF-16 and use 'wcout')

e. And this is a result:

```
quanp  Cryptography_Lab2  100ms  ERROR  .\UTF-16_inupt\aes_cbc_sample.exe
key: 78B7C84E414BFB7D4D728EDEE05B95E6
iv: 4B9EF25BCDEFCF67F3B3B43BD1B976ED
plain text: ةجفص
cipher text: E3C0A2D11082E3B7EA592DDBBDCFBABB
recovered text: ةجفص
quanp  Cryptography_Lab2  66ms
```

4. Bài tập 04 (bt4.cpp):

The most of the code depents on the aes_cbc_sample.cpp file. Instead of static plaintext “string plain = “CBC Mode””, edit into:

```
string plain;
cout << "Type plaintext: "; std::getline(std::cin, plain);
```

The whole code remains the same. Run the program:

```
PS D:\Move\UIT\HK4\MMH\ThucHanh\Lab2\Code> .\bt4.exe
Type plaintext: CBC Mode
key: C2B36A0ED5AD2560C7475CD86FD8B3C4
iv: 81D11B5049FF8922D4A1D517504996AD
plain text: CBC Mode
cipher text: F1353E22A8E25CB214DD840930429EB1
recovered text: CBC Mode
```

5. Bài tập 05 (bt5.cpp):

For short, I just copy parts of the bt5.cpp from lab1 and change DES to AES.

The key is now a string named keystring, can type from keyboard. Cast the string stringkey to the SecByteBlock type by using reinterpret_cast:

```
string keystring;
cout << "Type 8-byte key: ";
std::getline(std::cin, keystring);
// Convert string  to SecByteBlock
// &keystring[0]: the beginning of the data
// keystring.size: the size of the array
SecByteBlock key(reinterpret_cast<const byte*>(&keystring[0]), keystring.size());
```

With iv, use the byte from CryptoPP library, because the CryptoPP supports it with cin and no need to convert:

```
byte iv[8];
for (size_t i = 0; i < 8; i++){
    cout << i << ": ";
    std::cin >> iv[i];
}
```

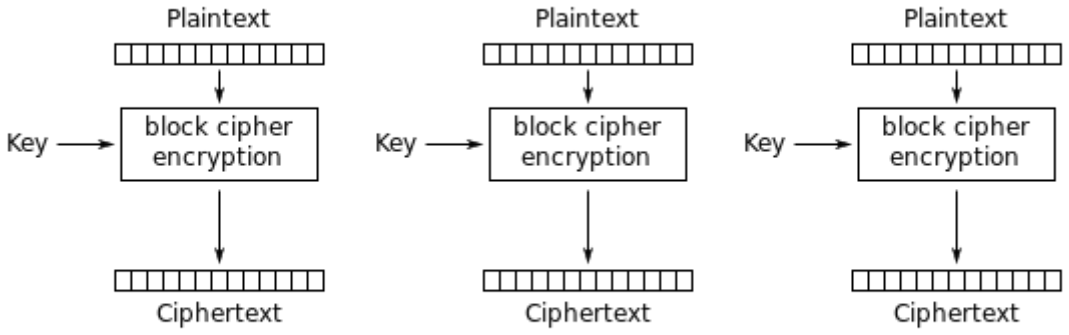
The remain is the same. Run the program:

```
PS D:\Move\UIT\HK4\MMH\ThucHanh\Lab2\Code> .\bt5.exe
Type 8-byte key: abcd1234
5: 6
6: 7
key: abcd1234
iv: 3132333435363738
plain text: CBC Mode Test
cipher text: 2DF117ABC1C6055A7BB9555EC8D02437
recovered text: CBC Mode Test
```

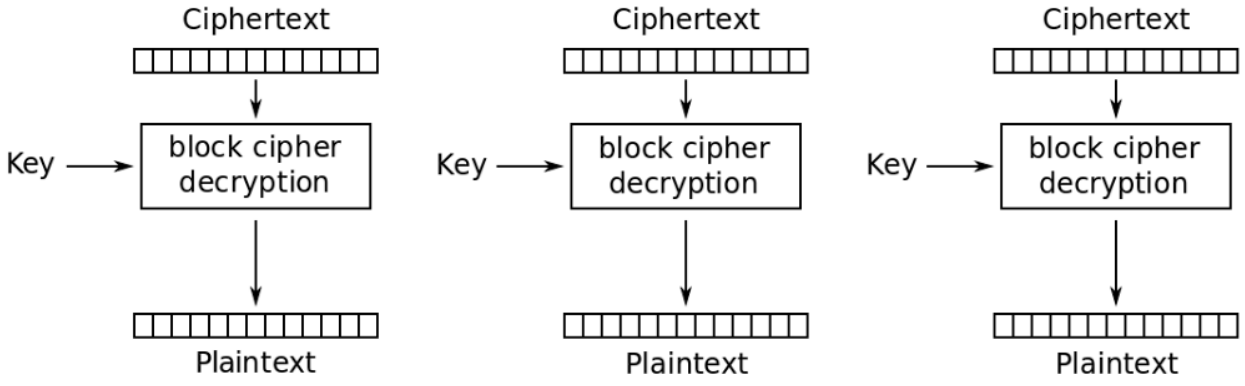
6. Bài tập 06:

Mode 1: ECB (file bt6_ecb.cpp)

Concept: ECB is considered as the most easiet mode of all Block cipher modes, and, consequently, most vulnerable mode. It does not have IV; the plaintext is divided into many plaintext blocks, but unlike CBC, in encryption direction, plaintext blocks are independent and are not XORed with something; the same happens to ciphertext in the reverse direction.



Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

About code:

ECB does not use IV, so “SetKey” is used instead of “SetKeyWithIV” for encryption and decryption, and CBC_Mode change to ECB_Mode:

```
ECB_Mode< AES >::Encryption e;
e.SetKey(key, sizeof(key));
```

```
ECB_Mode< AES >::Decryption d;
d.SetKey(key, sizeof(key));
```

The remain code is the same as the aes_cbc_sample.cpp. Run the program:

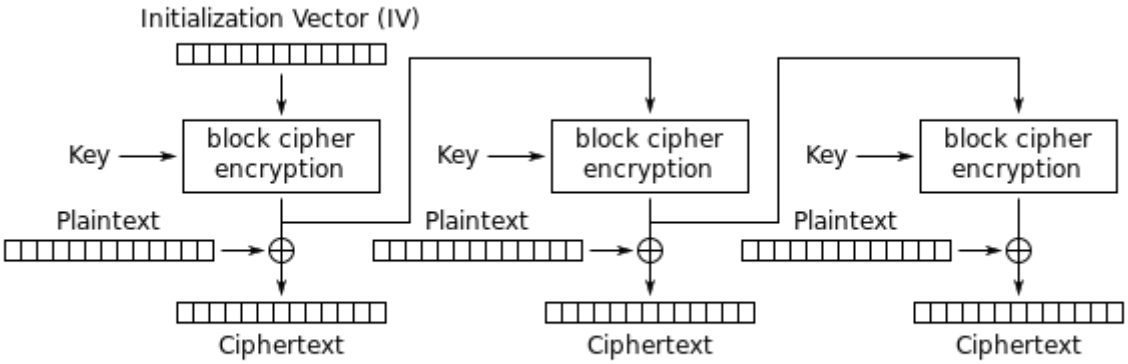
```
PS D:\Move\UIT\HK4\MMH\ThucHanh\Lab2\Code> .\bt6_ecb.exe
key: 25A3845D1AEA378B9FFA30D642835DDC
plain text: ECB Mode Test
cipher text: 6880F9D2A02C014041DF75E9BD29D008
recovered text: ECB Mode Test
```

Mode 2: CBC

Mode CBC is used from "bai tap 1" to "bai tap 5". The mechanism of CBC is describe detailed in the lab.

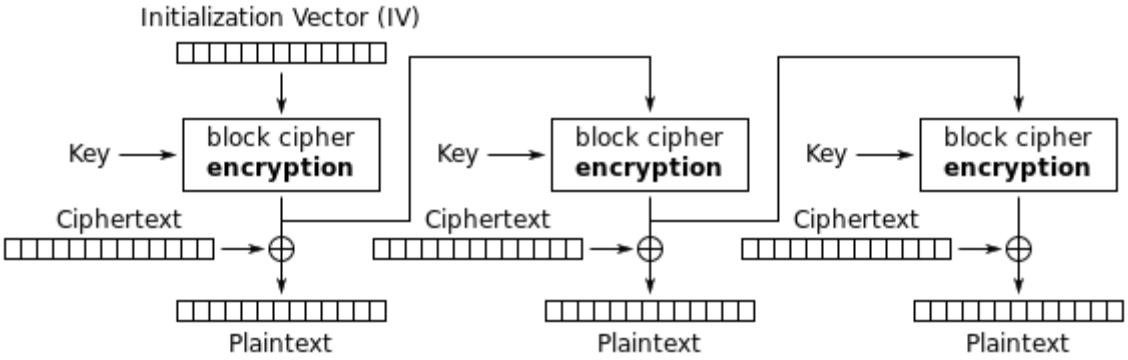
Mode 3: OFB (file bt6_ofb.cpp):

OFB is different to CBC in that in encryption, instead of XOR plaintext with IV like CBC, OFB creates a first keystream block by computing the key with IV, then XOR the first keystream block with first plaintext block to form the first ciphertext block. The first keystream block is then used with the key to compute the second keystream block.



Output Feedback (OFB) mode encryption

In decryption, the keystream block is XORed with plaintext. Note that the keystream is generated by the same encryption function in the encryption, or we can say encryption and decryption function is the same.



Output Feedback (OFB) mode decryption

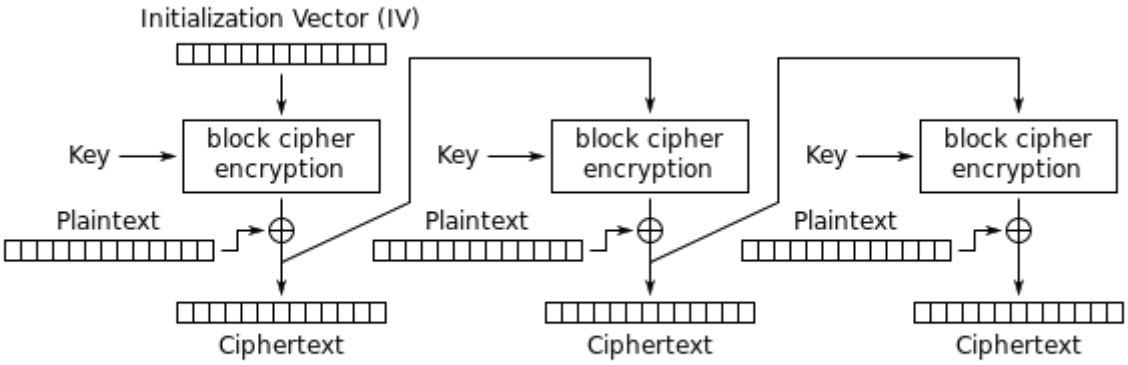
About code:

Most part remains the same as in aes_cbc_sample.cpp. Just change the CBC mode to OFB mode

```
PS D:\Move\UIT\HK4\MMH\ThucHanh\Lab2\Code> .\bt6_ofb.exe
iv: D6AD40DAC3914286BA41A7DD13D009AC
key: 2FE3D90A9ABA952A1D3D345822A0AA9C00000000000000000000000000000000
plain text: OFB Mode Test
cipher text: 04243966C45F4B2BDC02B00F37
recovered text: OFB Mode Test
```

Mode 4: CFB (bt6_cfb.cpp):

In encryption, CFB creates a output block cipher from IV and Key. The encryption is like OFB in CFB simplest form, which uses the entire output block cipher;



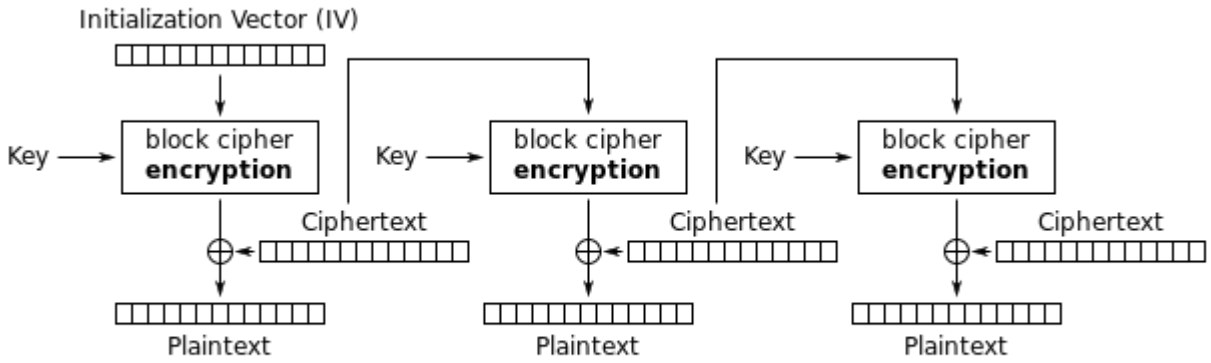
Cipher Feedback (CFB) mode encryption

in other cases, only a number s of most significant bits from the block cipher are used, which called “segments”:

$$
\begin{aligned}
I_0 &= \mathrm{IV}. \\
I_i &= \left((I_{i-1} \ll s) + C_i\right) \bmod 2^b, \\
\text{(Encrypt)}\quad C_i &= \mathrm{MSB}_s\left(E_K(I_{i-1})\right) \oplus P_i, \\
\text{(Decrypt)}\quad P_i &= \mathrm{MSB}_s\left(E_K(I_{i-1})\right) \oplus C_i,
\end{aligned}
$$

In the pic above, Ci is used for encryption, Pi is for decryption.

Like OFB, the encryption and decryption function is the same. However, in decryption, the first ciphertext block is used to be the input in the second decryption



Cipher Feedback (CFB) mode decryption

About code: Similar to CBC code, just change where “CBC” appears to CFB

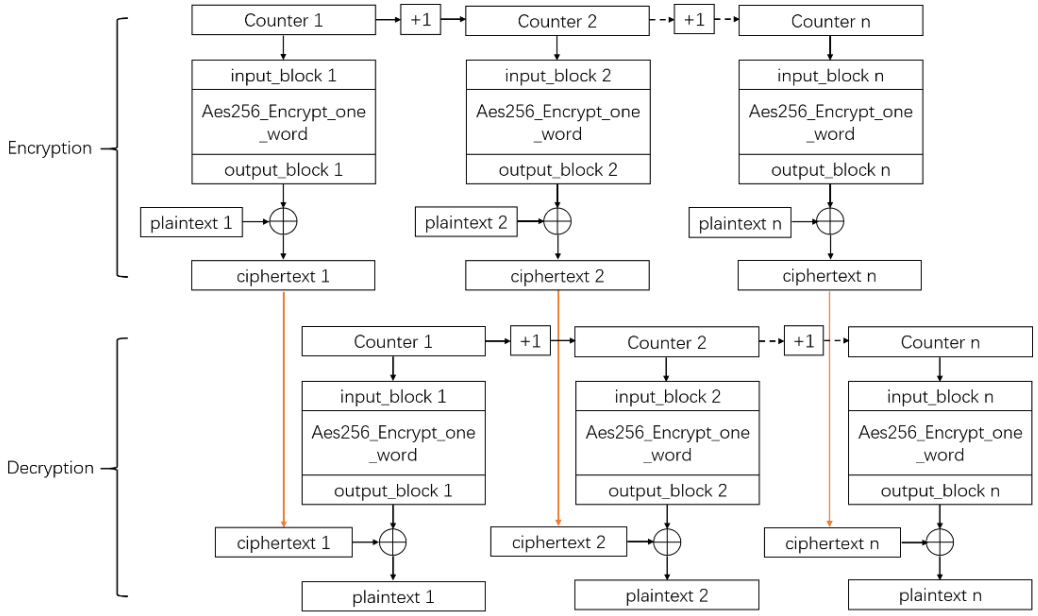
Run the program:

```
PS D:\Move\UIT\HK4\MMH\ThucHanh\Lab2\Code> .\bt6_cfb.exe
iv: 6D767D6FCF91DFB5DD1AE57CA4DEC418
key: 3E751DACDD7308098F7919B82C4AF007000000000000000000000000000000000
plain text: CFB Mode Test
cipher text: 55CD66EB75CC5765C68C7D0968
recovered text: CFB Mode Test
```

Mode 5: CTR

CTR mode is quite similar to OFB mode, but CTR mode replace IV to Counter. The counter can be any function which produces a sequence which is guaranteed not to repeat for a long time. First at all, the Counter 1 is encrypted like a plaintext with AES algorithm, after that the output XOR with the first plaintext block and we will get ciphertext1. subsequent ciphertext is encrypted similarly with Counter 2, 3, ...n. The decryption seems reversed

Because of parallel encryption and decryption too (e.g. ciphertext2 doesn’t have to wait for encryption of first block), I guess this system is faster than CBC, OFB, CFB.

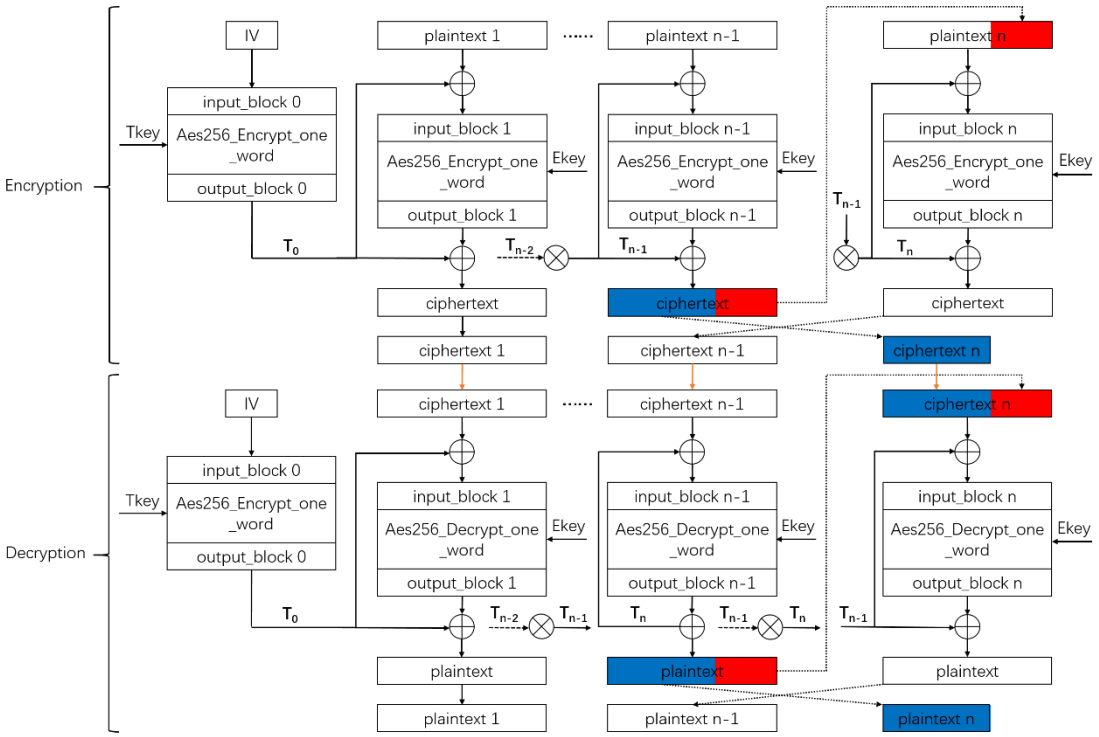


Run the program:

```
quanp  Cryptography_Lab2  471ms  .\AES\CTR\CTR.exe
plain text: CTR Mode Test
key: DB94ED62279FECEFA37C252C8860EC8F
iv: 3332DC718576165C23243A3A0941A234
cipher text: 56845D52F9B0A6A54E9630F437
recovered text: CTR Mode Test
```

Mode 6: XTS

In XTS mode of AES Encryption, firstly, IV is encrypted through AES algorithm with Tkey (tweakable key) and the output, I call T_0, T_0 XOR first block of plaintext (plaintext 1), T_0 XOR output of AES encryption using encryption key of plaintext 1, we will get a result, I call C_1, C_1 is ciphertext 1, C_1 XOR plaintext 2, C_1 XOR output of AES encryption using encryption key of plaintext 2. If we continue, the last block of plaintext will be encrypted. Moreover, I think this mode is not padding for last block, instead of using last bits of ciphertext n-1. The decryption seems reversed



Run the program:

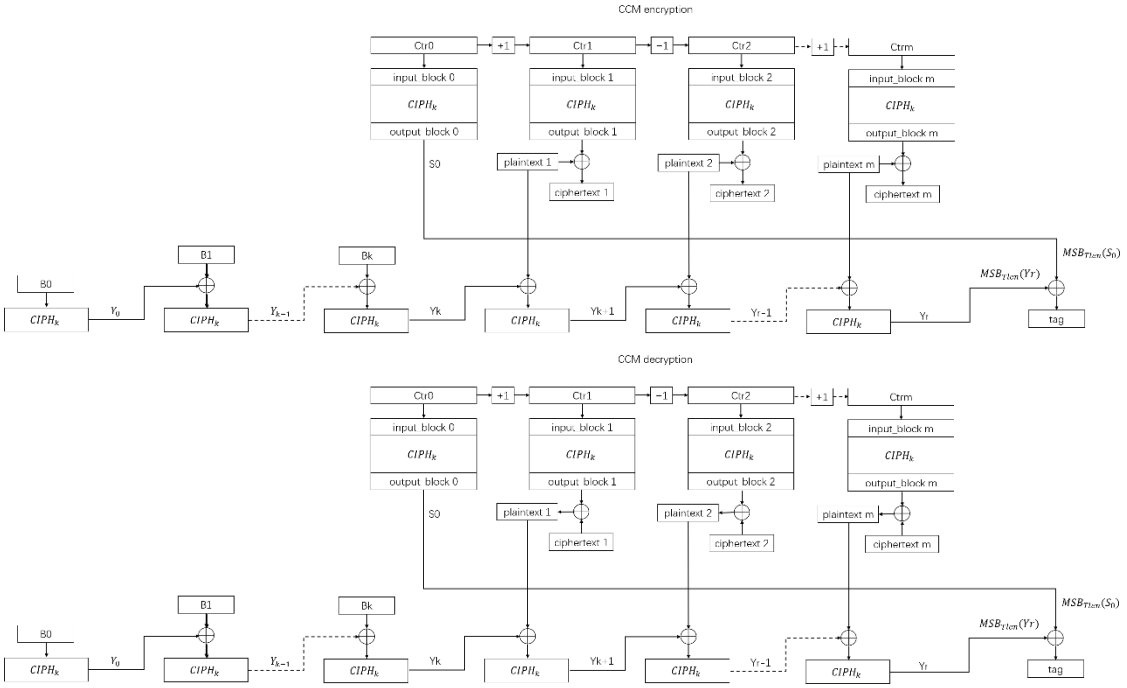
```
quanp  Cryptography_Lab2  37ms

                                                .\AES\XTS\XTS.exe
plain text: XTS Mode Test XTS Mode Test
key: A9020F999C0B49FC2E4E5935F2EF91C6FF440FAA318E47BA711A6B6C54F416FD
iv: E447C888EDF602CF00FDD9AB2C046BF7
cipher text: B55689A87714C342E7D3B137BA9FB076228F532629850D6C0D0D7A
recovered text: XTS Mode Test XTS Mode Test
quanp  Cryptography_Lab2  45ms
```

If I use the shorter plaintext:

```
                                                .\AES\XTS\XTS.exe
plain text: XTS Mode Test
XTS: message is too short for ciphertext stealing
quanp  Cryptography_Lab2  44ms  ERROR
```

Mode 7: CCM

This mode uses Counter mentioned above to encrypt, the encryption is quite similar to the CTR mode but it has some parts to check the integrity which I don't understand clearly. Generally, after encrypting, there will be a tag used to check the integrity. The decryption seems reversed



This mode uses Authentication for both Encryption and Decryption

```
{
    CCM< AES, TAG_SIZE >::Encryption e;
    e.SetKeyWithIV( key, key.size(), iv, sizeof(iv) );
    e.SpecifyDataLengths( 0, pdata.size(), 0 );

    StringSource ss1( pdata, true,
        new AuthenticatedEncryptionFilter( e,
            new StringSink( cipher )
        ) // AuthenticatedEncryptionFilter
    ); // StringSource
}
```

```
    AuthenticatedDecryptionFilter df( d,
        new StringSink( rpdata )
    ); // AuthenticatedDecryptionFilter
```

Moreover, It also checks the data's integrity

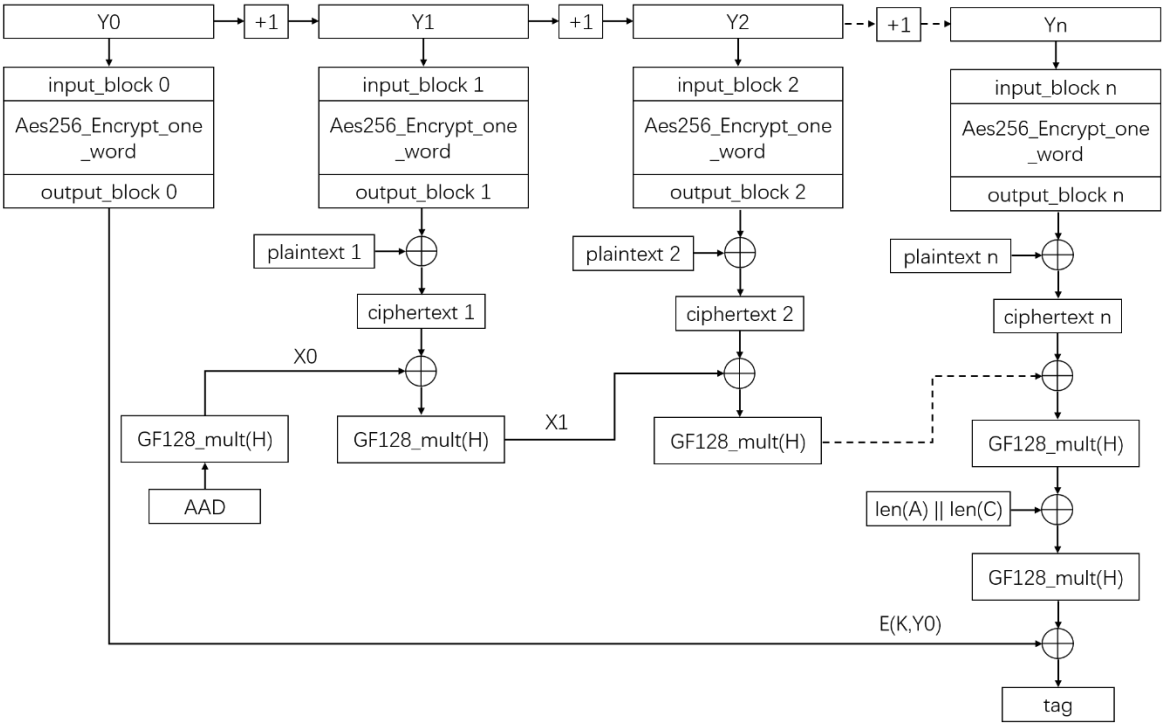
```
    //  opportunity to check the data's integrity
    if( true == df.GetLastResult() ) {
        cout << "recovered text: " << rpdata << endl;
    }
}
```

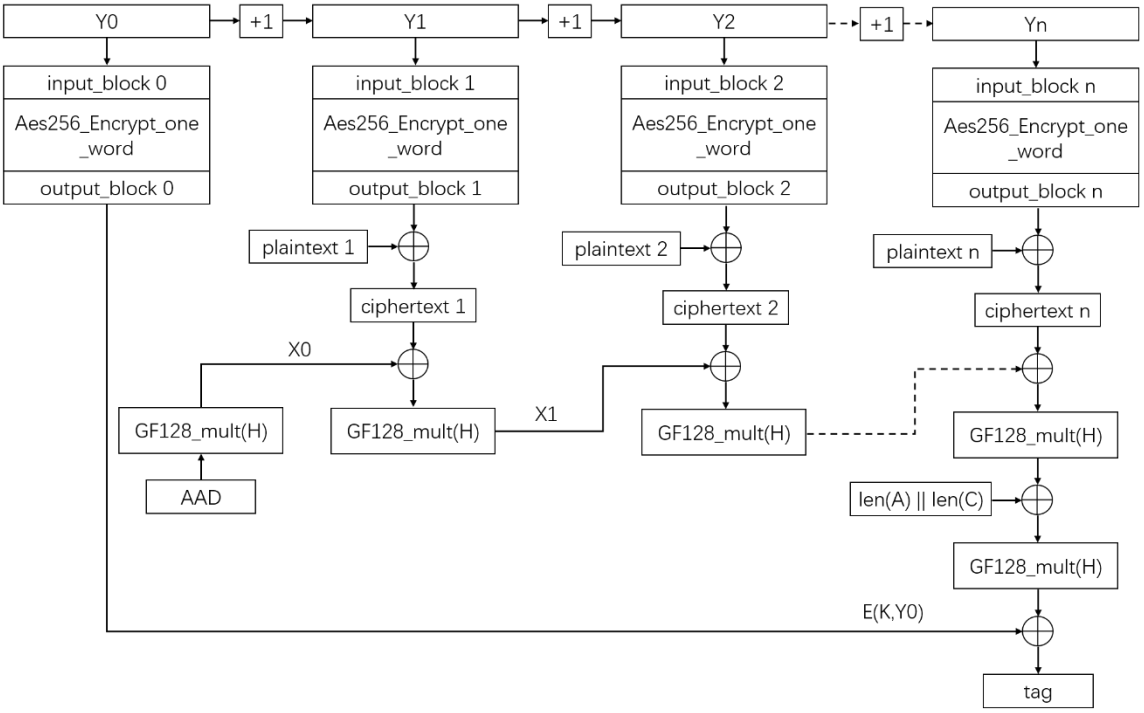
Run the program:

```
quanp  Cryptography_Lab2  450ms

                                        .\AES\CCM\CCM.exe
plain text: CCM Mode Test
key: B29743D325C7AE9BE7C5F5A9C00E1FBC
iv: C4DC5F1589B3C1384BBBEF45
cipher text: EFC3B1EFF3B1D107D16D90183D50A303B46B008C99
recovered text: CCM Mode Test
quanp  Cryptography_Lab2  443ms
```

Mode 8: GCM

In this mode, I see the same of the encryption with above mode. But The tag creation is different.





GCM mode does not only use Authenticated Encryption and Decryption Filter too, but it also has a DEFAULT_FLAGS value that I don't understand clearly.

```
const int TAG_SIZE = 12;
const int DEFAULT_FLAGS = 0;
```

But if I change DEFAULT_FLAGS to another value like 12, the result will be:

```
28
29      const int TAG_SIZE = 12;
30      const int DEFAULT_FLAGS = 12;
31
32      // Plain text

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS

cipher text: B55689A87714C342E7D3B137BA9FB076228F532629850D6C0D0D7A
recovered text: XTS Mode Test XTS Mode Test
 quanp   Cryptography_Lab2   445ms
                                                    .\AES\XTS\XTS.exe

plain text: XTS Mode Test
XTS: message is too short for ciphertext stealing
 quanp   Cryptography_Lab2   444ms   ERROR

                                                    .\AES\GCM\GCM.exe
plain text: GCM Mode Test
key: 13B9827CD4F0F028728B6F9FB1C3CD9E
iv: 3E3A48D68B28ADEA653C7AD241471F3E
cipher text: AB30AC92A92214F382D30D118738B65DC82E85EF99B0BDC353
recovered text: GCM Mode Test8▌]╚.àn▒▟┤S0
 quanp   Cryptography_Lab2   464ms
```

And it has a tag to check the data's integrity.

Run the program:

```
quanp  Cryptography Lab2  31ms  .\AES\GCM\GCM.exe
plain text: GCM Mode Test
key: C2A9BC977609DE19665BE3024C4F4A9C
iv: 44050CAAA9B2A4F8F1FC7A98190E8C3D
cipher text: 9B2608C43131D3361D2B046F7846792E66E814884AFABFA006
recovered text: GCM Mode Test
```

7. Bài tập 07 (file bt7.cpp):

ECB has a weakness that if the plaintext has a part of 1 block appears in other blocks, the ciphertext will consequently have some blocks that have repeated parts.

In the scenario that the key, iv is 16 bytes. I type a string "ECB Mode XXXXXXXECB Mode XXXXXXX" for plaintext (32 characters, 1 char = 1 byte) with hoping that ciphertext has 2 same segments for the repeated string "ECB Mode XXXXXXX". Demo:

```
PS D:\Move\UIT\HK4\MMH\ThucHanh\Lab2\Code> .\bt7.exe
Type plaintext: ECB Mode XXXXXXXECB Mode XXXXXXX
key: 0FEDDEEFE3640BA5BA7D0CAC77B0D97B
plain text: ECB Mode XXXXXXXECB Mode XXXXXXX
cipher text: D4FF78447DA66171A02D4F175C044C87D4FF78447DA66171A02D4F175C044C877C865AEEC07A1208157B8A9E13F61037
recovered text: ECB Mode XXXXXXXECB Mode XXXXXXX
```

The "D4FF78447DA66171A02D4F175C044C87" is repeated (32 characters, 1 char = 4 bit => total = 128 bits, the same length to "ECB Mode XXXXXXX"). I don know why there are still 32-bits in the end which is different than "D4FF78447DA66171A02D4F175C044C87", perhaps because of padding mechanism.

If plaintext has repeated parts in different blocks, the ciphertext will have blocks which contain same parts of characters. If the plaintext is large enough, this vulnerability can let the ciphertext show various different repeated strings to reverse most parts of plaintext.

8. Bài luyện tập 03 (file bai_luyen_tap_3.cpp):

It is almost the same as "Bài luyện tập 3" for DES in lab 1, except that the key length in AES does not accept 8 bytes:

```
terminate called after throwing an instance of 'CryptoPP::InvalidKeyLength'
  what():  AES/CBC: 8 is not a valid key length
```

so I try 16-byte key length. Therefore, the iv must be extended to 16 bytes.

```
Type 16-byte key: abcd111122223333
plaintext: CBC Mode Test demo CBC Mode Test demo CBC Mode Test demo
Type array IV:
0: 0
1: 1
2: 2
3: 3
4: 4
5: 5
6: 6
7: 7
8: 8
9: 9
10: 1
11: 2
12: 3
13: 4
14: 5
15: 6
```

More over, now I know why just only a part of "Eve cipher text" is the same as "cipher text": it is because of the padding machanism of CryptoPP.

```
cipher text: τ*&z‼µ┤R»╢±½Ö"E◘¿3╜¥┼=nsyY
Eve cipher text: τ*&z‼µ┤R»╢±½Ö"E╒╩`å(
≤z
Recovered text: CBC Mode Test demo CBC Mode Test demo CBC Mode Test demo
```

The following lines is the AES weakness, as repeated from "Bài luyện tập 3" for DES.

Reference:
https://www.cs.jhu.edu/~rubin/courses/sp03/papers/voydock.kent.pdf

One weakness of CBC is that if the attacker know the IV, he can implement a chosen plaintext attack.

Scenario: the attacker can catch the transmission between a user X and X's mail box on host H amd even can mail to X. Each uniqe IV and key is used for the session (the paper used the word "association") and not hidden, but all packets use the same IV.

The attacker can:

- Know the IV of the session.

- see the encrypted messages.

Assume:

- Session (or association): A

- IV of the session A: IV_A

- first block of the i-th packet: U_i

- ciphertext: E(X) with X is the result of the XOR 2 byte strings.

Logic of the attack:

- Because the attacker can know ciphertext, it means he knows $E(IV_A \ xor \ U_i)$

- Because he can send message to user X, it means he can determine the message, including the first patterns K_i that will create the first block the i-th packet, which means he knows $E(IV_A \ xor \ K_i)$ and will find K_m through bruteforce until $E(IV_A \ xor \ K_m) = E(IV_A \ xor \ U_i)$ with all i packets.

- When he has the first plaintext block U_1, he can replace IV by U_1 to find second, third, ... n-th plaintext blocks of the packets he has caught.

Demo program description:

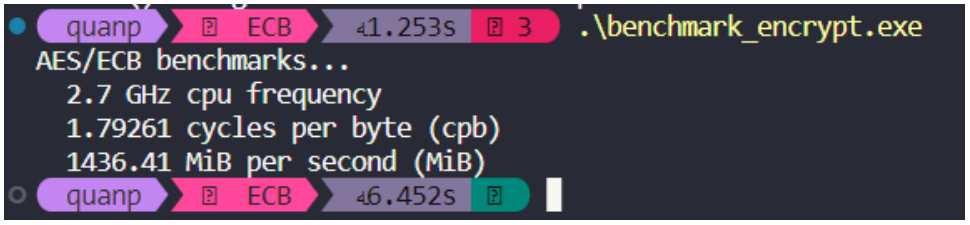
- IV (var iv), key (var keystring), plaintext (var plain) is input from keyboard. Only one IV and one key are used throughout the program.

- The pattern K_1 (var eP1, which means eavesdropper first Plaintext block) is "CBC Mode Test Demo CBC Mode Test Demo CBC Mode Test Demo".

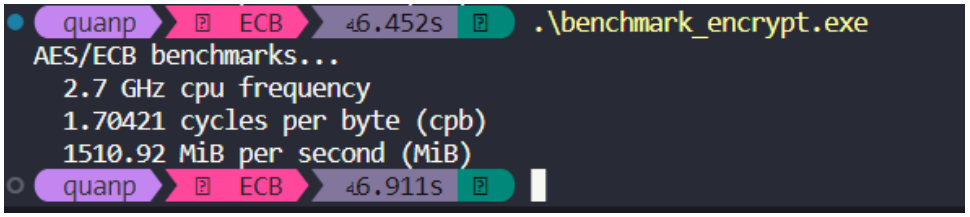
9. Bài Luyện tập 01:

- Mode ECB:

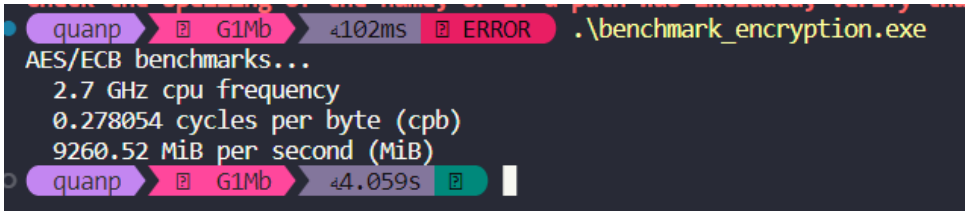
+ Data < 64 bit



+ UTF-16 Data



+ Data > 1Mb



- Mode CBC:

+ Data < 64 bit

```
quanp  L64bit  1.261s  ERROR
                                        .\benchmark_encrypt.exe
AES/CBC benchmarks...
  2.7 GHz cpu frequency
  2.45564 cycles per byte (cpb)
  1048.58 MiB per second (MiB)
quanp  L64bit  4.225s
```

+ UTF-16 data

```
                                        .\benchmark_encrypt.exe
AES/CBC benchmarks...
  2.7 GHz cpu frequency
  2.50475 cycles per byte (cpb)
  1028.02 MiB per second (MiB)
quanp  UTF-16  4.253s
```

+ Data > 1Mb

```
quanp  G1Mb  3ms  .\benchmark_encryption.exe
AES/CBC benchmarks...
  2.7 GHz cpu frequency
  1.43137 cycles per byte (cpb)
  1798.92 MiB per second (MiB)
quanp  G1Mb  4.912s
```

- Mode CFB:

+ Data < 64bit

```
quanp  L64bit  119ms  .\benchmark_encrypt.exe
AES/CFB benchmarks...
  2.7 GHz cpu frequency
  3.50665 cycles per byte (cpb)
  734.297 MiB per second (MiB)
quanp  L64bit  6.205s
```

+ UTF-16 data

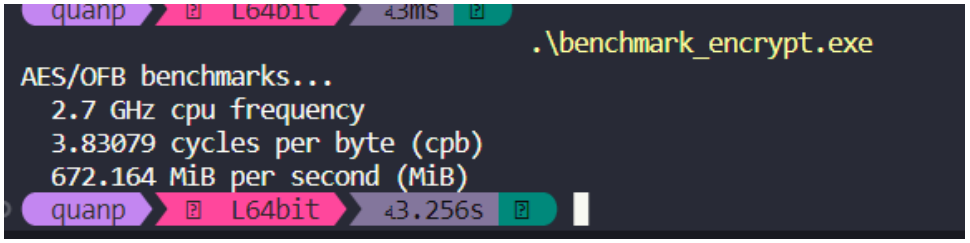
```
quanp  UTF-16  2ms  .\benchmark_encrypt.exe
AES/CFB benchmarks...
  2.7 GHz cpu frequency
  3.3986 cycles per byte (cpb)
  757.642 MiB per second (MiB)
quanp  UTF-16  6.281s
```

+ Data > 1Mb

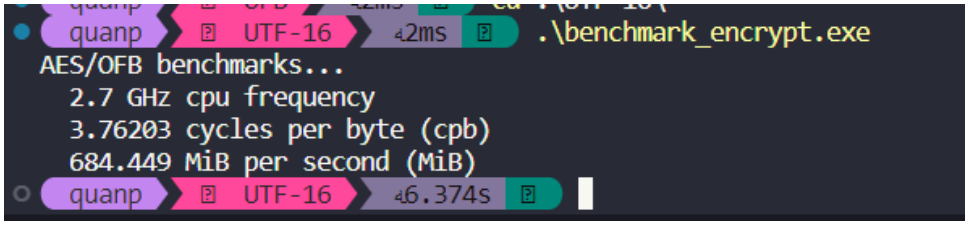
```
quanp  G1Mb  2ms  .\benchmark_encryption.exe
AES/CFB benchmarks...
  2.7 GHz cpu frequency
  1.43621 cycles per byte (cpb)
  1792.86 MiB per second (MiB)
quanp  G1Mb  5.17s
```

- Mode OFB:

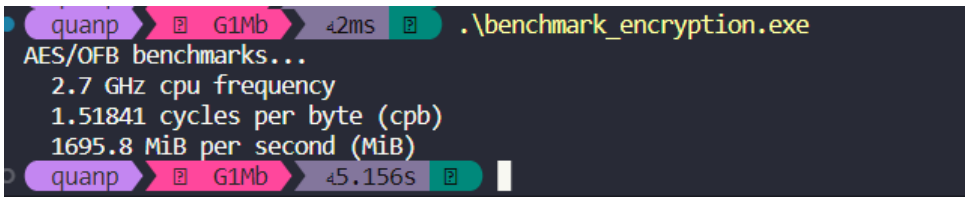
+ Data < 64bit



+ UTF-16 data

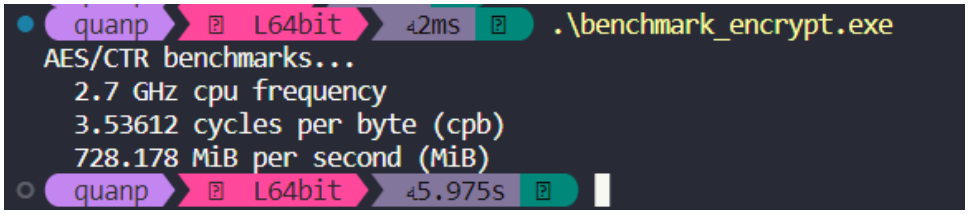


+ Data > 1Mb

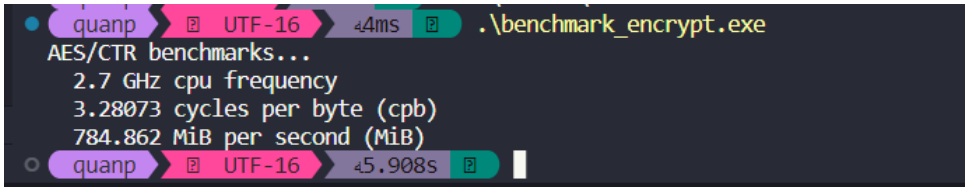


- Mode CTR:

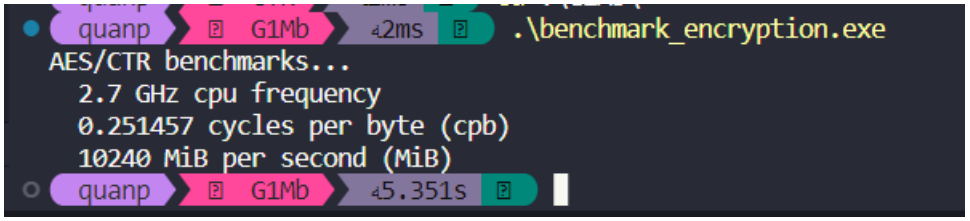
+ Data < 64bit



+ UTF-16 data

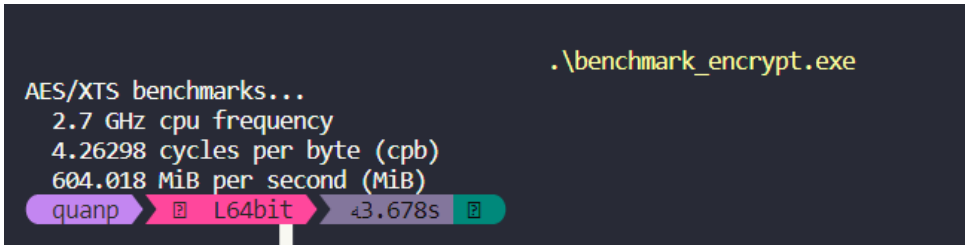


+ Data > 1Mb

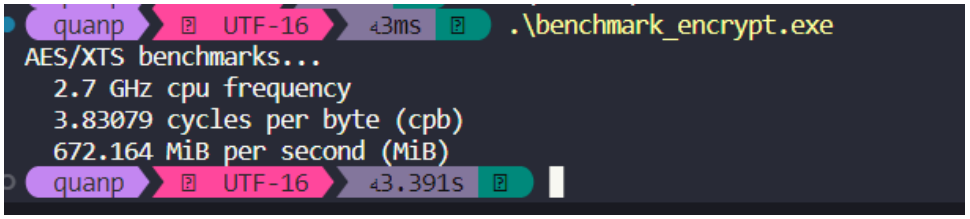


- Mode XTS:

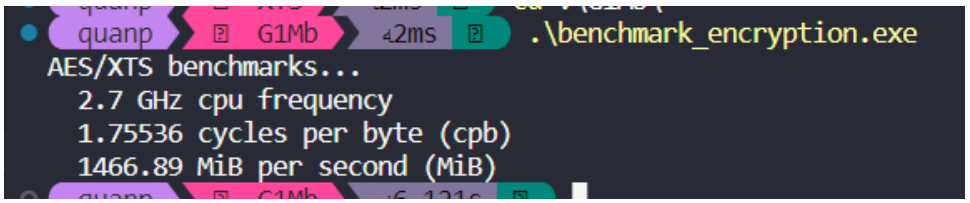
+ Data < 64bit



+ UTF-16 data

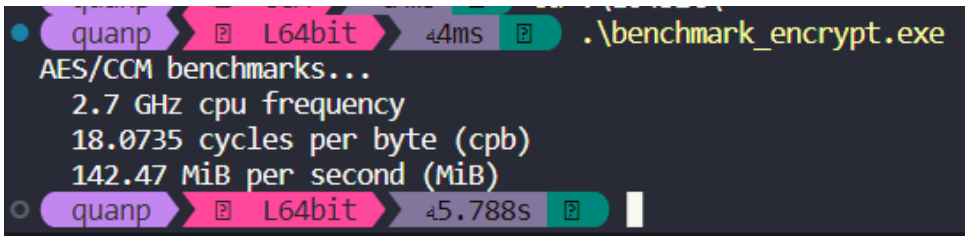


+ Data > 1Mb

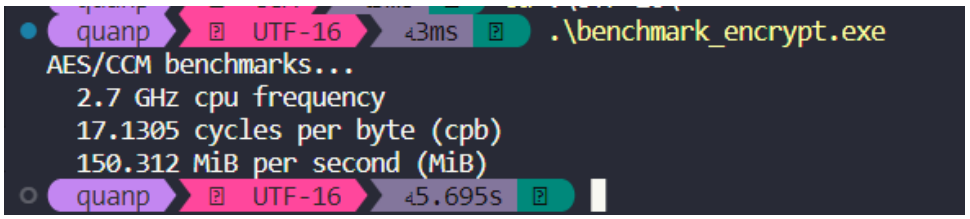


- Mode CCM:

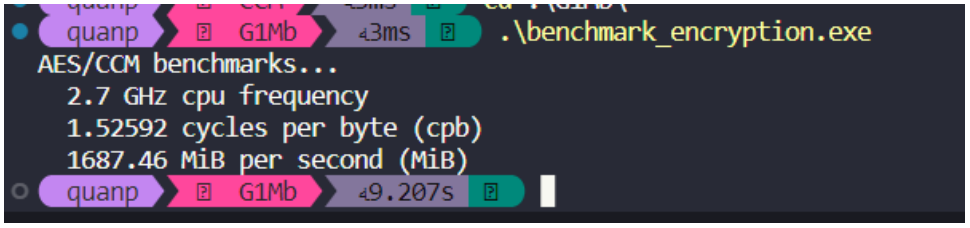
+ Data < 64bit



+ UTF-16 data

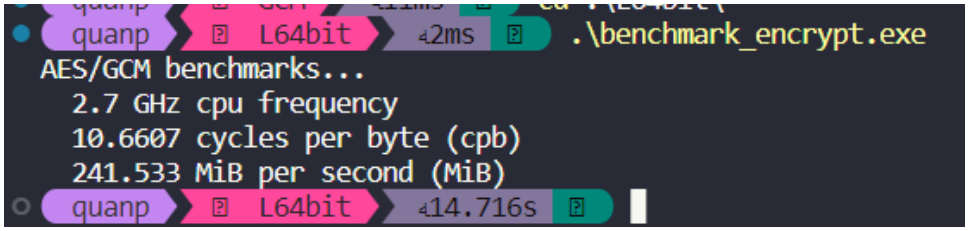


+ Data > 1Mb

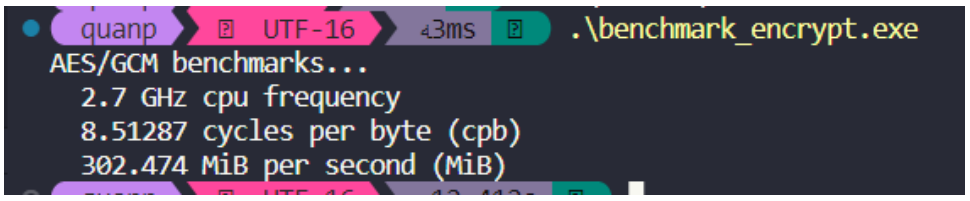


- Mode GCM:

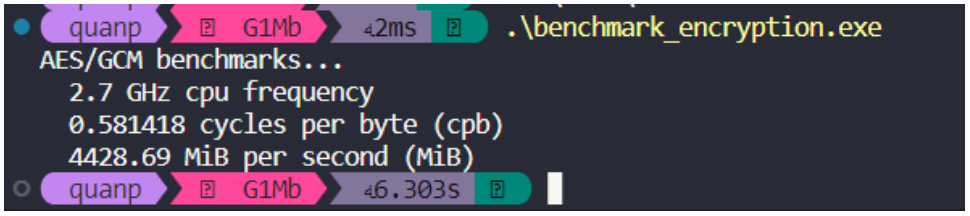
+ Data < 64bit



+ UTF-16 data



+ Data > 1Mb



File .exe: https://github.com/PNg-HA/MatMaHoc_exe.git

Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này

YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File .PDF. Tập trung vào nội dung, không mô tả lý thuyết.
- Nội dung trình bày bằng Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)– cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.
- Đặt tên theo định dạng: [Mã lớp]-SessionX_GroupY. (trong đó X là Thứ tự buổi Thực hành, Y là số thứ tự Nhóm Thực hành/Tên Cá nhân đã đăng ký với GV).

  *Ví dụ: [*NT101.K11.ANTT*]-Session1_Group3.*
- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Không đặt tên đúng định dạng – yêu cầu, sẽ **KHÔNG** chấm điểm.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá: Sinh viên hiểu và tự thực hiện. Khuyến khích:

- Chuẩn bị tốt.
- Có nội dung mở rộng, ứng dụng trong kịch bản/câu hỏi phức tạp hơn, có đóng góp xây dựng.

Bài sao chép, trễ, … sẽ được xử lý tùy mức độ vi phạm.

HẾT