CS 3800 Project 3
Encryption with Substitution Cipher and the Cipher Block Chaining Mode

Auraiporn Auksorn(aauksorn@cpp.edu)

(1) Introduction

The program first generates 8 bytes of random initialize vector, then it prompts for a message to be encrypted from a user. The program converts the message to an array of stream of ASCII characters, then the program determines when the end of the input is reached. If there is any padding should be applied, then the program will add the missing bytes to complete the last plaintext to be 8 bytes. As each plaintext is one block of size 8 bytes, and it is used to perform the cipher block chaining mode (XOR operation between plaintext and ciphertext). After that, the program performs encryption process with substitution cipher by using a function that takes two parameters of a number, k representing the substitution cipher shift, such that each ASCII character would be encrypted by the kth character ahead of it in the alphabet and iv (initialize vector). The result of ciphertext obtains by storing an initialize vector and the results of ciphertext (c0, c1, c2, …, cn) from substitution cipher into an array. Finally, the program displays both a message and encrypted message.

(2) Main Components

Encyption.java

- Encryption.java contains the following components to encrypt a message by using a substitution cipher with the Cipher Block Chaining mode.

    - A method to generates an initialize vector to perform XOR operation with the first block of plaintext.
    - A method to determine when the end of the input is reached, and if there is any padding should be applied, then the program will add the missing bytes to complete the last plaintext to be 8 bytes. In order to determine what should be the padding, the program first determine the length of the message and perform modulus operation, i.e., message.length() % block size, then it takes the result from modulus operation to subtract from default block size which is 8 bytes.
      The following is an example of how the padding algorithm in this program works.

```
        Let x be any ascii bytes in the range of ascii table; there are 256
        possible characters.
               - If the last plaintext has 8 bytes, the padding size returns 8.
        However, it simply determines the end of the string, but it does not add
        anything to the end of the last plaintext.
               - If the last plaintext has 7 bytes, the padding size returns 1.
        It means that there is one more byte needed to be filled to complete 8
        bytes, so the last plaintext will be [x,x,x,x,x,x,x,1].
               - If the last plaintext has 6 bytes, the padding size returns 2.
        It means that there are 2 more bytes needed to be filled to complete 8
        bytes, so the last plaintext will be [x,x,x,x,x,x,2,2].
               - If the last plaintext has 5 bytes, the padding size returns 3.
        It means that there are 3 more bytes needed to be filled to complete 8
        bytes, so the last plaintext will be [x,x,x,x,x,3,3,3].
               - If the last plaintext has 4 bytes, the padding size returns 4.
        It means that there are 4 more bytes needed to be filled to complete 8
        bytes, so the last plaintext will be [x,x,x,x,4,4,4,4].
               - If the last plaintext has 3 bytes, the padding size returns 5.
        It means that there are 5 more bytes needed to be filled to complete 8
        bytes, so the last plaintext will be [x,x,x,5,5,5,5,5].
               - If the last plaintext has 2 bytes, the padding size returns 6.
        It means that there are 6 more bytes needed to be filled to complete 8
        bytes, so the last plaintext will be [x,x,6,6,6,6,6,6].
               - If the last plaintext has 1 byte, the padding size returns 7.
       It means that there are 7 more bytes needed to be filled to complete 8
bytes, so the last plaintext will be [x,7,7,7,7,7,7,7].
```

- A method to perform substitution cipher with the Cipher Block Chaining mode by using two for loops. The outer loop keeps track of number of blocks, and the inner loop keeps track of numbers of elements in the array of byte. This method name is substitution_cipher_with_CBC, which accepts three parameters: a byte array of plaintext, a byte array of initialize vector, and an integer value of k.
- A method to encrypt the result of a byte array from XOR operation between plaintext and ciphertext by using substitution cipher, and it accepts two parameters: a byte array of iv, which is a pointer to the initializing vector, and a number, k representing the substitution cipher shift, such that each ASCII character would be encrypted by the kth character ahead of it in the alphabet.
- A method to perform XOR operation of two ASCII byte arrays.
- A method to convert an ArrayList of Byte to a byte array.
- An ArrayList that stores the final results of ciphertext which having the help from a method that converts byte arrays to ArrayList of Byte.
- A method to display ArrayList, which is used to display the ASCII byte values of the final ciphertext.
- A method to takes a byte array and display it into ASCII characters.
- A method that takes a byte array and display it in a binary representation.

  - A method that takes a byte array and display it in a hexadecimal representation.

(3) Challenges

3.1 Challenge #1:

Implementing CBC mode was challenging.

It took me a while to figure out how to split an array into a chuck of size 8 as I first thought that I would have to create multiple arrays to store each plaintext. Moreover, there were a lot of components that I had to take in account such as the first XOR operation is taking the first 8 bytes of plaintext and initialize vector, but after that each plaintext will be calculated by XOR operation with each ciphertext, which is the result from previous operation. I also had to keep track the number of blocks that I have, and how it will be managed to follow the substitution cipher with the Cipher Block Chaining mode. Eventually, I figured it out by performing everything inside the two for loops. It works with any size of plaintext. I tested my code by using different strategies; for example, in normal scenario where the input is multiple of 8 bytes, and in scenario where the input requires padding. The code that I have works in both ways. The way I tested the program; I displayed the ASCII characters of input and output in hexadecimal and binary number. Then, I checked the result after performing XOR operation of each plaintext and ciphertext is correct. I also double checked if the substitution cipher is working correctly by observing that all the encrypted messages are decipherable.

3.2 Challenge #2:

I was not sure how would I determine the padding of the plaintext.

I understand that the project description says that “For this problem, you are allowed to select any reasonable system to determine that the end of the input is reached, and/or when padding should be applied to complete the block. You may select any output format, as long as it is unambiguous.” This project description could result to the output of the encrypted message to be wide and open, so we could expect different outcomes.
For this reason, I eventually came up with padding scheme that will first determine how many bytes are missing to complete 8 bytes of the last block of plaintext. Then, it simply adds the bytes of “numbers that are missing to fill the last block”.
For example, it the message entered has the length of 20 which is not divisible by 8. Then, it will add 4 more bytes to make the plaintext size to be 24, and it means that we now have 3 blocks to process. The last block would be the following.
Let x be any bytes of ASCII character from entered message, where ‘4’ is the character that will be filled to complete a block of 8 bytes.
The last block of plaintext if an initial plaintext has the length of 20 bytes is [x, x, x, x, 4, 4, 4, 4].
To conclude, the characters, that will be replaced in the last plaintext, are the numbers of missing bytes in the last plaintext. I think this algorithm would meet the requirement of the project because first it is reasonable as I could decrypt the message after the encryption process.

Moreover, it makes the output format unambiguous. The reason that I did not replace the missing bytes with printable ascii characters because it is harder to trace the code, and it would be harder to make the encrypted message be decipherable. Also, the XOR operation is going to give the result of ASCII within the range of ASCII table of 256 different characters which means that it will not necessarily stay in the range of printable characters.

(4) Results

<u>Sample Output when compile the program with usage:</u> `java Test`

```
The default k value that represents substitution cipher shift will be given the value of 20.
Please enter the message to be encrypted: helloworld
Bytes of ascii of entered message: [104, 101, 108, 108, 111, 119, 111, 114, 108, 100]
The size of entered message is: 10
Padding: 6 bytes
Ascii of IV is:       LRPIHFMA          The length of iv: 8
IV in Hexadecimal:    4C52504948464D41   IV in Binary:
1001100101001010100001001001100100010001101001101100000 1
The length of plain text is: 16
Bytes of plaintext(p0,p1,p2,p3,..pn): [104, 101, 108, 108, 111, 119, 111, 114, 108, 100, 6, 6, 6, 6, 6, 6]

************************  Begin cipher substitution with CBC mode ******************************
Split the plaintext into a block of 8 bytes, so there are 2 blocks of plaintext.
p0:                        68656C6C6F776F72 (in hexadecimal)
p0:                        hellowor
ivXORp0:                   24373C2527312233 (in hexadecimal)
ivXORp0:                   $7<%'1"3
c0:                        384B50393B453647 (in hexadecimal)
c0:                        8KP9;E6G

p1:                        6C64060606060606 (in hexadecimal)
p1:                        ld
p1 XOR c0:                 542F563F3D433041 (in hexadecimal)
p1 XOR c0:                 T/V?=C0A
c1:                        68436A5351574455 (in hexadecimal)
c1:                        hCjSQWDU

Plaintext(p0, p1, p2,...,pn) in hexadecimal and in binary:  (68656C6C6F776F726C64060606060606)
(110100011001011101100110110011011111110111110111111100101101100110010011011011011 0110110)
Ciphertext(iv, c0, c1, c2, ...,cn) in hexadecimal and in binary:
(4C52504948464D41384B50393B45364768436A5351574455)
(100110010100101010000100100110010001000110100110110000001111000100101110100001110011110111000101110110100011
1110100010000111101010101001110100011010111100010010101 01)
Ciphertext in bytes of ASCII is:  76 82 80 73 72 70 77 65 56 75 80 57 59 69 54 71 104 67 106 83 81 87 68 85

********************************** Result *******************************************************
A message to be encrypted is:     helloworld
An encrypted message is:          LRPIHFMA8KP9;E6GhCjSQWDU
*************************************************************************************************

Note:
Size of original message is:  10           Size of ciphertext is: 24
Padding size is:              6           Plaintext size is:     16
```

Sample Output when compile the program with usage:

`Syntax: java Test <<k value>>`

In this case is: `java Test 128`

```
k value is: 128
Please enter the message to be encrypted: helloworld
Bytes of ascii of entered message: [104, 101, 108, 108, 111, 119, 111, 114, 108, 100]
The size of entered message is: 10
Padding: 6 bytes
Ascii of IV is:       KIKYVMRZ           The length of iv: 8
IV in Hexadecimal:    4B494B59564D525A   IV in Binary:
10010111001001100101110110011010110100110110100101011010
The length of plain text is: 16
Bytes of plaintext(p0,p1,p2,p3,..pn): [104, 101, 108, 108, 111, 119, 111, 114, 108, 100, 6, 6, 6, 6, 6, 6]

*************************  Begin cipher substitution with CBC mode ******************************
Split the plaintext into a block of 8 bytes, so there are 2 blocks of plaintext.
p0:                     68656C6C6F776F72 (in hexadecimal)
p0:                     hellowor
ivXORp0:                232C2735393A3D28 (in hexadecimal)
ivXORp0:                #,'59:=(
c0:                     A3ACA7B5B9BABDA8 (in hexadecimal)
c0:                     £¬?µ?º½?

p1:                     6C64060606060606 (in hexadecimal)
p1:                     ld♠♠♠♠♠♠
p1 XOR c0:              CFC8A1B3BFBCBBAE (in hexadecimal)
p1 XOR c0:              ??¡?¿¼»?
c1:                     4F4821333F3C3B2E (in hexadecimal)
c1:                     OH!3?<;.

Plaintext(p0, p1, p2,...,pn) in hexadecimal and in binary:  (68656C6C6F776F726C64060606060606)
(110100011001011101100110110011011111110111101111111001011011001100100110110110110110110)
Ciphertext(iv, c0, c1, c2, ...,cn) in hexadecimal and in binary:
(4B494B59564D525AA3ACA7B5B9BABDA84F4821333F3C3B2E)
(10010111001001100101110110011010110100110110100101011010111111111111111111111111111010001111111111111111111
1111110101100111111111111111111111111110100111111111111111111111111111110110101111111111111111111111111111011100
11111111111111111111111111011101011111111111111111111111111101111011111111111111111111111111111010100010011111001
0001000001110011111111111100111011101110)
Ciphertext in bytes of ASCII is:  75 73 75 89 86 77 82 90 -93 -84 -89 -75 -71 -70 -67 -88 79 72 33 51 63 60
59 46

********************************* Result *******************************************************
A message to be encrypted is:   helloworld
An encrypted message is:        KIKYVMRZ£¬?µ?º½?OH!3?<;.
*************************************************************************************************

Note:
Size of original message is:  10        Size of ciphertext is: 24
Padding size is:              6         Plaintext size is:     16
```

Sample output of a long message:

```
k value is: 250
Please enter the message to be encrypted: Encrypt this input using substitution cipher with the
CipherBlockChainingmode.
The size of entered message is: 79
Padding: 1 bytes
The length of plain text is: 80
*************************  Begin cipher substitution with CBC mode ******************************
Split the plaintext into a block of 8 bytes, so there are 10 blocks of plaintext.
p0:                     456E637279707420 (in hexadecimal)
p0:                     Encrypt
ivXORp0:                07252C282E223073 (in hexadecimal)
ivXORp0:                %,(."0s
c0:                     011F2622281C2A6D (in hexadecimal)
c0:                     ☺▼&"(∟*m

p1:                     7468697320696E70 (in hexadecimal)
p1:                     this inp
p1 XOR c0:              75774F510875441D (in hexadecimal)
p1 XOR c0:              uwOuD↔
c1:                     6F71494B026F3E17 (in hexadecimal)
c1:                     oqIK☻o>↨

p2:                     7574207573696E67 (in hexadecimal)
p2:                     ut using
p2 XOR c1:              1A05693E71065070 (in hexadecimal)
p2 XOR c1:              →♣i>q♠Pp
c2:                     14FF63386B004A6A (in hexadecimal)
c2:                     ¶ÿc8k Jj

p3:                     2073756273746974 (in hexadecimal)
p3:                      substit
p3 XOR c2:              348C165A1874231E (in hexadecimal)
p3 XOR c2:              4?▬Z↑t#▲
c3:                     2E861054126E1D18 (in hexadecimal)
c3:                     .?►T↕n↔↑

p4:                     7574696F6E206369 (in hexadecimal)
p4:                     ution ci
p4 XOR c3:              5BF2793B7C4E7E71 (in hexadecimal)
p4 XOR c3:              [òy;|N~q
c4:                     55EC73357648786B (in hexadecimal)
c4:                     Uìs5vHxk

p5:                     7068657220776974 (in hexadecimal)
p5:                     pher wit
p5 XOR c4:              25841647563F111F (in hexadecimal)
p5 XOR c4:              %?▬GV?◄▼
c5:                     1F7E104150390B19 (in hexadecimal)
c5:                     ▼~►AP9♂↓

p6:                     6820746865204369 (in hexadecimal)
p6:                     h the Ci
p6 XOR c5:              775E642935194870 (in hexadecimal)
p6 XOR c5:              w^d)5↓Hp
c6:                     71585E232F13426A (in hexadecimal)
c6:                     qX^#/‼Bj

p7:                     70686572426C6F63 (in hexadecimal)
p7:                     pherBloc
p7 XOR c6:              01303B516D7F2D09 (in hexadecimal)
p7 XOR c6:              ☺0;Qm⌂-
c7:                     FB2A354B67792703 (in hexadecimal)
c7:                     û*5Kgy'♥

p8:                     6B436861696E696E (in hexadecimal)
```

```
p8:                    kChainin
p8 XOR c7:             90695D2A0E174E6D (in hexadecimal)
p8 XOR c7:             ?i]*♫↨Nm
c8:                    8A63572408114867 (in hexadecimal)
c8:                    ?cW◄Hg

p9:                    676D6F64652E2001 (in hexadecimal)
p9:                    gmode. ☺
p9 XOR c8:             ED0E38406D3F6866 (in hexadecimal)
p9 XOR c8:             í♫8@m?hf
c9:                    E708323A67396260 (in hexadecimal)
c9:                    2:g9b`

******************************** Result ******************************************************
A message to be encrypted is: Encrypt this input using substitution cipher with the CipherBlockChainingmode.
An encrypted message is:      BKOZWRDS☺▼&"(∟*moqIK☻o>↨¶ÿc8k Jj.?►T↕n↔↑Uìs5vHxk▼~►AP9♂↓qX^#/‼Bjû*5Kgy'♥?cW◄Hg2:g9b`
************************************************************************************************

Note:
Size of original message is:  79        Size of ciphertext is: 88
Padding size is:              1         Plaintext size is:     80
```

<u>The algorithm that I used to obtain the above output is as followed:</u>

- Obtain a number, k representing the substitution cipher shift, from the command line argument. If k value is not given from the command line argument, the default k value is equal to 20.
- Prompt a message from standard input to obtain plaintext.
- Generate the 8-bytes initialize vector (IV) in the top register.
- XOR the 8-bytes first plaintext block with data value in the top register (IV).
- Encrypt the result of XOR operation with underlying block cipher with key k by the substitution cipher shift such that each ASCII character would be encrypted by the kth character ahead of it in the alphabet.
- Feed ciphertext block into top register and continue the operation till all plaintext blocks are processed.
- Store an initialize vector and all ciphertext blocks in an array and print the ciphertext on the standard output.

(5) Evaluation

With no background in network security, I strived for when I first started working on the last project of this class, and I have been optimizing the code by trying to understand the functionality of the encryption to ensure that this project would meet the requirement. I did my best to try to understand the topics that we have been discussed in class and used the knowledge from our class to apply to this project. I think I have learned about network security more in details while accomplishing this project. Especially, I developed a better understanding that CBC is a symmetric algorithm which operates in block mode, and it works on fixed-size blocks of data. Even though most block cipher modes require the length of plaintext to be a multiple of the block size of the underlying encryption algorithm, sometimes the length of plaintext will not be multiple of the block size. Therefore, I had the opportunity to learn and implement the padding scheme. I also learned that to improved encryption process we need IVs to be unpredictable and random, but I initially hard coded an initialize vector. The more time I have spent and worked on this project; I have developed a better understanding of the network security.