CS 656 LAB 1
Symmetric Encryption

Notes:

- This is an individual lab.
- The code and other answers you submit MUST be entirely your own work, and you are bound by the WSU Academic Integrity Policy (`https://www.wichita.edu/about/student_conduct/ai.php`). You MAY consult with other students about the conceptualization of the tasks and the meaning of the questions, but you MUST NOT look at any part of someone else's solution or collaborate with anyone. You may consult published references, provided that you appropriately cite them in your reports and programs, as you would do in an academic paper.
- Read the entire document carefully before you start working on the lab.

GOOD LUCK!

1 Overview

The learning objective of this lab is to get familiar with the concepts in the secret-key (symmetric) encryption. From this lab, students will gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initialization vector (IV). Moreover, students will be able to use tools and write programs to encrypt/decrypt messages. This lab covers the following topics:

- Symmetric encryption
- Block cipher encryption modes
- Programming using the crypto library

Lab Environment. This lab has been tested on Ubuntu 20.04. You will be given access to the remote Ubuntu environment with necessary packages installed. You may also run your own virtual machine by downloading it from the SEED labs website (see the details here: `https://github.com/seed-labs/seed-labs/blob/master/manuals/vm/seedvm-manual.md`).

2 Submission

Submit a PDF document with your answers to the questions in this lab. Your report should have a subheading for each question, and your answers should be inside the corresponding subheading. If applicable, list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.

Note: *Your report must contain your Name (Lastname, Firstname) and WSU ID. Use the following format for your report's PDF filename:* `lab1report_YOUR_WSU_ID.pdf`*. For example, if your ID is C656S656, your report should be* `lab1report_c656s656.pdf`*. There is a 10% reduction of points if your report does not follow the correct filename format and/or missing name/ID inside the document.*

3 Lab Tasks

3.1 Task 1: Encryption using Different Ciphers and Modes [10 Points]

In this task, we will play with various encryption algorithms and modes. You can use the following `openssl enc` command to encrypt/decrypt a file. To see the manuals, you can type `man openssl` and `man enc`.

```
$ openssl enc -ciphertype -e  -in plain.txt -out cipher.bin \
              -K  00112233445566778889aabbccddeeff \
              -iv 0102030405060708
```

Replace the `ciphertype` with a specific cipher type, such as `-aes-128-cbc`, `-bf-cbc`, `-aes-128-cfb`, etc. In this task, you should try at least 3 different ciphers. You can find the meaning of the command-line options and all the supported cipher types by typing "`man enc`". We include some common options for the `openssl enc` command in the following:

```
  -in <file>      input file
  -out <file>     output file
  -e              encrypt
  -d              decrypt
  -K/-iv          key/iv in hex is the next argument
  -[pP]           print the iv/key (then exit if -P)
```

Deliverable. List the ciphers and modes you used in this task and why you picked those ciphers/modes.

3.2 Task 2: Encryption Mode – ECB vs. CBC [20 Points]

The files `task2_original_1.bmp` and `task2_original_2.bmp` contains simple pictures. We would like to encrypt these pictures, so people without the encryption keys cannot know what is in the picture. Encrypt the files using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

- Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. However, For the `.bmp` file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate `.bmp` file. We will replace the header of the encrypted picture with that of the original picture. We can use the `GHex` or `bless` hex editor tool to directly modify binary files. Let us assume the original image is `p1.bmp` and encrypted image is `p2.bmp`. We can also use the following commands to get the header from `p1.bmp`, the data from `p2.bmp` (from offset 55 to the end of the file), and then combine the header and data together into a new file (say, `new.bmp`).

```
$ head -c 54 p1.bmp  > header
$ tail -c +55 p2.bmp > body
$ cat header body > new.bmp
```

Deliverable.

1. Using EBC mode perform the following tasks for both of the given images.

   (a) Display the encrypted pictures using any picture viewing software and add a screen shot to your report. You may use the `eog` image viewer program already installed in the VM.
   (b) Can you derive any useful information about the original picture from the encrypted picture? Briefly explain your observations (no more than 5 sentences) in terms of the ECB encryption mechanism.

2. Using CBC mode perform the following tasks for both of the given images.
   (a) Display the encrypted pictures using any picture viewing software and add a screen shot to your report.
   (b) Can you derive any useful information about the original picture from the encrypted pictures? Briefly explain your observations (no more than 5 sentences) in terms of the CBC mechanism.

3.3 Task 3: Error Propagation – Corrupted Cipher Text [20 Points]

To understand the error propagation property of various encryption modes, we would like to do the following exercise:

1. Create a text file that is at least 1000 bytes long. You are given a sample plaintext file named `task3_plaintext.txt`. You can either use this file or create your own.
2. Encrypt the file using the AES-128 cipher.
3. Let us assume that a single bit of the 55th byte in the encrypted file got corrupted. You can achieve this corruption using the `GHex` or `bless` hex editor (i.e., arbitrarily edit a byte). The editors are already installed on the remote machine.
4. Decrypt the corrupted ciphertext file using the correct key and IV.

Deliverable. Please answer the following questions in your lab report:

- Provide screen shots of your original file, the corrupted file, and the output of the decryption under both ECB and CBC.
- How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB and CBC, respectively?

  Answer this question *before* you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. Provide justification of your initial thoughts and final findings.

3.4 Task 4: Brute-force Attack using the Crypto Library [50 Points]

In this task, you are given a plaintext and a ciphertext, and your job is to find the key that is used for the encryption. You do know the following facts:

- The `aes-128-cbc` cipher is used for the encryption.
- The key used to encrypt this plaintext is an English word shorter than 16 characters; the word can be found from a typical English dictionary. Since the word has less than 16 characters (i.e. 128 bits), pound signs (#: hexadecimal value is `0x23`) are appended to the end of the word to form a key of 128 bits.

Your goal is to write a program to find out the encryption key. Consider the plaintext, ciphertext, and IV are listed in the following:

```
Plaintext (total 21 characters): This is a top secret.
Ciphertext (in hex format): 764aa26b55a4da654df6b19e4bce00f4
                            ed05e09346fb0e762583cb7da2ac93a2
IV (in hex format):         aabbccddeeff00998877665544332211
```

You need to pay attention to the following issues:

- In this task, you are supposed to write your own program to invoke the crypto library. No credit will be given if you simply use the `openssl` commands to do this task. Sample code can be found from the following URL:

  ```
  https://www.openssl.org/docs/man1.1.1/man3/EVP_CipherInit.html
  ```

- When you compile your code using `gcc`, do not forget to include the `-lcrypto` flag, because your code needs the `crypto` library. See the following example for the code filename `bfattack.c`:

  ```
  $ gcc -o bfattack bfattack.c -lcrypto
  ```

- For your convenience, we provide a starter C file `task4_starter_bfattack.c` with some code skeleton. Although we use C as an example, ***you can use any programming language to complete this task***. Note that OpenSSL is natively written in C. You are welcome to use other programming languages for which OpenSSL wrappers exist. For example, there exists OpenSSL wrapper in Python: `https://pyopenssl.org/en/stable/api.html`.

Note. To receive full marks, your code should also be able to launch the attack against a different ciphertext encrypted with a different word in the English dictionary after submission.

Deliverable. You are given the following files:
`task4_plaintext.txt`
`task4_ciphertext.txt`
`task4_iv.txt`
`task4_wordlist.txt`
`task4_starter_bfattack.c`.
Use the following filename format for your program: `lab1_task4_YOUR_WSU_ID.<extension>`
For example, if you use C and your WSU ID is C656S656, your source file would be `lab1_task4_c656s656.c`. Upload a copy of your code. In your report, include a brief explanation of your implementation and the secret key that you found.