Secret-Key Encryption Lab

Overview

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

Many common mistakes have been made by developers in using the encryption algorithms and modes. These mistakes weaken the strength of the encryption, and eventually lead to vulnerabilities. This lab exposes students to some of these mistakes, and ask students to launch attacks to exploit those vulnerabilities. This lab covers the following topics:

- Secret-key encryption
- Substitution cipher and frequency analysis
- Encryption modes, IV, and paddings
- Common mistakes in using encryption algorithms
- Programming using the crypto library

Lab Environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website: https://seedsecuritylabs.org/Labs_16.04/Crypto/Crypto_Encryption/

1 Task 1: Encryption using Different Ciphers and Modes

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
```

Please 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)

Submission for Task 1 (20 points)

(1) Which ciphers you have tried? (5 points)
(2) What are commands you used for those ciphers? Please write down (you can copy and paste) the commands or attach a screenshot of your commands in your report. (10 points)
(3) Can you successfully decrypt ciphertext which is encrypted by your selected ciphers? Please write down one command you used to decrypt your ciphertext. (5 points)

2 Task 2: Encryption Mode – ECB vs. CBC

The file *pic_original.bmp* can be downloaded from this lab's website, and it contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. 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 *bless* hex editor tool (already installed on our VM) to directly modify binary files. 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.

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

2. Display the encrypted picture using a picture viewing program (we have installed an image viewer program called *eog* on our VM). Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations

3. Select a picture of your choice, repeat the experiment above, and report your observations.

Submission for Task 2 (20 points)

(1) Please attach the screenshots of images which are encrypted using ECB and CBC modes respectively. Please note that the encrypted images are the ciphertext and you may want to change the header information before you can open the encrypted images. (5 points)
(2) What is difference between the image encrypted using ECB mode and that encrypted using CBC mode. Which mode is more secure? And why? (10 points)
(3) Please attach the picture of your choice, the encrypted picture using ECB and CBC modes. (5 points)

3 Task 3: Padding

For block ciphers, when the size of a plaintext is not a multiple of the block size, padding may be required. All the block ciphers normally use PKCS#5 padding, which is known as standard block padding. We will conduct the following experiments to understand how this type of padding works:

1. Use ECB, CBC, CFB, and OFB modes to encrypt a file (you can pick any cipher). Please report which modes have paddings and which ones do not. For those that do not need paddings, please explain why.

2. Let us create three files, which contain 5 bytes, 10 bytes, and 16 bytes, respectively. We can use the following "*echo -n*" command to create such files. The following example creates a file *f1.txt* with length 5 (without the *-n* option, the length will be 6, because a newline character will be added by *echo*):

   $ echo -n "12345" > f1.txt

   We then use "*openssl enc -aes-128-cbc -e*" to encrypt these three files using 128-bit AES with CBC mode. Please describe the size of the encrypted files.

   We would like to see what is added to the padding during the encryption. To achieve this goal, we will decrypt these files using "*openssl enc -aes-128-cbc -d*". Unfortunately, decryption by default will automatically remove the padding, making it impossible for us to see the padding. However, the command does have an option called "*-nopad*", which disables the padding, i.e., during the decryption, the command will not remove the padded data. Therefore, by looking at the decrypted data, we can see what data are used in the padding. Please use this technique to figure out what paddings are added to the three files.

   It should be noted that padding data may not be printable, so you need to use a hex tool to display the content. The following example shows how to display a file in the hex format:

```
$ hexdump -C  p1.txt
00000000   31  32  33  34  35  36  37  38  39  49  4a  4b  4c  0a      |123456789IJKL.|
$ xxd  p1.txt
00000000: 3132  3334  3536  3738  3949  4a4b  4c0a                   123456789IJKL.
```

Submission for Task 3 (20 points)

(1) Which modes have paddings and which ones do not? How did you know if a mode has a padding through the encrypted file? For those that do not need paddings, please explain why. Please also list the commands that you used in step 1. (5 points)
(2) Please describe the size of the encrypted files which are encrypted in step 2? (5 points)
(3) What paddings are added to the three files in step 2? Please attach the screenshots of decrypted files with paddings (i.e., decrypt the files with "*-nopad*"). (10 points)

4 Task 4: Error Propagation – Corrupted Cipher Text

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.

2. Encrypt the file using the AES-128 cipher.

3. Unfortunately, a single bit of the 55th byte in the encrypted file got corrupted. You can achieve this corruption using the bless hex editor.

4. Decrypt the corrupted ciphertext file using the correct key and IV.

Submission for Task 4 (20 points)

(1) Please answer the following question before you conduct this task: How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? (5 points)
(2) Please find out whether your answer is correct or wrong after you finish this task. Please provide justification. Specifically, please attach the screenshot of decrypted ciphertext in Step 4 and calculate how many bits are flipped compared with the plaintext. (15 points)

5 Task 5: Programming using the Crypto Library

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. Please the English word list attached in this lab assignment. The plaintext, ciphertext, and IV are listed in the following:

Plaintext (total 21 characters):	This is a top secret.
Ciphertext (in hex format):	1b4faa3403db410533b2e99b4a4dcf2a
	93a5af11b35988e62cb57d03e221bfc6
IV (in hex format):	aabbccddeeff00998877665544332211

Submission for Task 5 (20 points)

(1) Please write a program to find out the encryption key. Please attach the screenshot of your program. You can use any programming language that you are familiar with to conduct this task. (10 points)
(2) What is the key used in this task? (10 points)