UNIVERSITY OF MARYLAND
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ENEE 457
Computer Systems Security
Instructor: Charalampos Papamanthou

Programming Project 2: Symmetric Encryption Lab

Out: 09/13/18 Due: 09/21/18 11:59pm

Instructions

1. Strictly adhere to the University of Maryland Code of Academic Integrity.
2. Submit your solutions as a pdf document at Canvas. Include your full name in the solutions document. Name the solutions document as x-project2.pdf, where x is your last name.

Partly taken from SEED labs `http://www.cis.syr.edu/~wedu/seed/`

1 Overview

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms and encryption modes. Moreover, students will be able to use tools and write programs to encrypt/decrypt messages.

2 Lab Setup

Installing OpenSSL. In this lab, we will use `openssl` commands and libraries. We have already installed `openssl` binaries in our VM. It should be noted that if you want to use `openssl` libraries in your programs, you need to install several other things for the programming environment, including the header files, libraries, manuals, etc. We have already downloaded the necessary files under the directory `/home/seed/openssl-1.0.1`. To configure and install `openssl` libraries, go to the `openssl-1.0.1` folder and run the following commands.

```
You should read the INSTALL file first:
  % sudo ./config
  % sudo make
  % sudo make test
  % sudo make install
```

Installing a hex editor. In this lab, we need to be able to view and modify files of binary format. For this purpose, you can install `Bless` by using the following command:

```
% sudo apt-get install bless
```

3 Lab Tasks

3.1 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`, `-aes-128-cfb`, `-bf-cbc`, etc. In this task, you should try at least 3 different ciphers and three different modes. 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)
```

3.2 Task 2: Encryption Mode – ECB vs. CBC

The file `pic_original.bmp` contains a simple picture. You can find the picture here: `http://www.cis.syr.edu/~wedu/seed/Labs_12.04/Crypto/Crypto_Encryption/files/pic_original.bmp`. 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. You can use a hex editor tool (e.g. `Bless`) to directly modify binary files.
2. Display the encrypted picture using any picture viewing software. Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.

3.3 Task 3: Encryption Mode – Corrupted Cipher Text

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

1. Create a text file that is at least 64 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. You can achieve this corruption using a hex editor.

4. Decrypt the corrupted file (encrypted) using the correct key and IV.

Please answer the following questions: (1) How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. (2) Please explain why. (3) What are the implication of these differences?

3.4 Task 4: Dictionary Attack using the OpenSSL Crypto Library

So far, we have learned how to use the tools provided by `openssl` to encrypt and decrypt messages. In this task, we will learn how to use `openssl`'s crypto library to encrypt/descrypt messages in programs.

OpenSSL provides an API called EVP, which is a high-level interface to cryptographic functions. Although OpenSSL also has direct interfaces for each individual encryption algorithm, the EVP library provides a common interface for various encryption algorithms. To ask EVP to use a specific algorithm, we simply need to pass our choice to the EVP interface. A sample code is given in `http://www.openssl.org/docs/crypto/EVP_EncryptInit.html`. Please get yourself familiar with this program, and then do the following exercise.

You are given a plaintext and a ciphertext, and you know that `aes-128-cbc` is used to generate the ciphertext from the plaintext, and you also know that the numbers in the IV are all zeros (not the ASCII character '0'). Another clue that you have learned is that the key used to encrypt this plaintext is an English word shorter than 16 characters; the word that can be found from a typical English dictionary. Since the word has less than 16 characters (i.e. 128 bits), space characters (hexadecimal value 0x20) 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 this key. You can download a English word list from the Internet but you can also find one here: `http://www.cis.syr.edu/~wedu/seed/Labs_12.04/Crypto/Crypto_Encryption/files/words.txt`.

The plaintext and ciphertext is in the following:

```
Plaintext (total 21 characters): This is a top secret.
Ciphertext (in hex format): 8d20e5056a8d24d0462ce74e4904c1b5
                            13e10d1df4a2ef2ad4540fae1ca0aaf9
```

Note 1: If you choose to store the plaintex message in a file, and feed the file to your program, you need to check whether the file length is 21. Some editors may add a special character to the end of the file. If that happens, you can use a hex editor tool to remove the special character.

Note 2: 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.

Note 3: To compile your code, you may need to include the header files in `openssl`, and link to `openssl` libraries. To do that, you need to tell your compiler where those files are. In your `Makefile`, you may want to specify the following:

```
INC=/usr/local/ssl/include/
LIB=/usr/local/ssl/lib/
```

```
all:
	gcc -I$(INC) -L$(LIB) -o enc yourcode.c -lcrypto -ldl
```

3.5 Task 5: Frequency Analysis Attack

Alice and Bob formed a committee to design a secure encryption scheme. Mallory wants to be able to spy on their communications, so she joined their standards committee and suggested they encrypt files by splitting the file into blocks of size equal to the length of the key, and then compute a XOR of each block of the plaintext with the key to get the ciphertext.

Their encryption program in C is given below. It reads the plaintext from stdin and outputs the ciphertext to stdout:

```
#include <stdio.h>
#include <unistd.h>

int main() {
    char key[8] = {'K', 'E', 'Y', ' ', 'H', 'E', 'R', 'E'};
    int keylen = 8;
    char *buf = malloc(keylen * 512);
    int nread;

    while (nread = read(0, buf, sizeof(buf))) {
        int i;
        for (i = 0; i < sizeof(buf); i ++)
            buf[i] ^= key[i % keylen];
        write(1, buf, nread);
    }
}
```

Your task is to help Mallory spy on their communications by writing a program to crack this encryption scheme. You are given two ciphertexts 1.enc and 2.enc that you need to find the key and plaintext for. In 1.enc, the length of the key is 8 (i.e., the key is 8 bytes). In 2.enc, the length of the key could be any value less than 30. The bytes of the key can be any value from 0x00 to 0xff.

Hints: First you will probably want to code and test a program to break the following simpler encryption scheme:

Given a plaintext and a 1-byte key, XOR every byte of the plaintext with the key to get the ciphertext. This encryption scheme can be attacked by trying to decrypt the ciphertext with all 256 possible keys, and picking the key that gives a character frequency distribution most similar to english text. The following array will probably be useful:

```
double freq[] = {0.0651738, 0.0124248, 0.0217339, 0.0349835,
                 0.1041442, 0.0197881, 0.0158610, 0.0492888,
                 0.0558094, 0.0009033, 0.0050529, 0.0331490,
                 0.0202124, 0.0564513, 0.0596302, 0.0137645,
                 0.0008606, 0.0497563, 0.0515760, 0.0729357,
                 0.0225134, 0.0082903, 0.0171272, 0.0013692,
                 0.0145984, 0.0007836, 0.1918182};
```

In the above array, `freq[0]` contains what fraction of characters in average english text are the letter 'a', `freq[1]` is for the letter b and so on.

4 Submission

Students need to submit a detailed lab report to describe what they have done and what they have observed. Report should include the evidences to support the observations. Evidences include source code with appropriate comments, packet traces, screenshots of outputs, etc.