Programming Assignment 2
CS370/ECE499

Overview
The learning objective of this project is for students to get familiar with using secret-key encryption and one-way hash functions in the openssl library. After finishing the assignment, in addition to improving their understanding of secret-key encryption and one-way hash function concepts, you should be able to use tools and write programs to generate one-way hash value and encrypt/decrypt messages.

Submission Guidelines

You need to submit responses for the questions related to tasks.

- For Tasks you have to write your own code in the language of your choice and implement it.
- Also, the report should be a pdf including all the images and write-up for the questions.
- Deadline for this assignment is **11:59 PM 17th November, 2016**
- You will submit a single tar archive that will include the pdf and all the code required for this assignment.
- Please include a Makefile along with your submission that compiles your code.
- You will have to work on the programming assignments in C language because you will use the OpenSSL library to complete them. Flip server (access.engr.oregonstate.edu) has openssl installed in them, so you can use them to work on your assignment.
- If you choose to work on other platforms, you are on your own to install and use the library.

Tasks

Installing OpenSSL: In this assignment, you will use openssl commands and libraries. Please install the latest version of openssl compatible with your platform. 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.

1 [10 pts] Observation Task: One-way Hash and their randomness.
In this exercise, you will generate message digests using a few different one-way hash algorithms. You can use the following openssl dgst command to generate the hash value for a file. To see the manuals, you can type man openssl and man dgst.

$ *openssl dgst dgsttype filename*

Please replace the dgsttype with a specific one-way hash algorithm, such as -sha1, -sha256, etc. In this exercise, you should try at least 3 different algorithms (sha1, sha256 and sha512). You can find the supported one-way hash algorithms by typing "man openssl".

To understand the properties of one-way hash functions, do the following exercise for SHA1 and

SHA256:

1. Create a text file with the following text -- 'The cow jumps over the moon'
2. Generate the hash value H_1 for this file using SHA1 (SHA256). (Add this value in your writeup)
3. Flip one bit of the input file. You can achieve this modification using hex editors like ghex or Bless.
4. Generate the hash value H_2 for the modified file.(Add this value in your writeup)
5. Please observe whether H_1 and H_2 are similar or not. How many bits are the same between H_1 and H_2.
6. Flip bits 1, 49, 73, and 113 and record the number of bits that are different between H_1 and H_2 when using SHA1 and SHA256 in a table. What trend do you see in table? Does this trend change if you flip different bits in the file or flip multiple bits?

2 [10 pts] Observation Task: Keyed Hash and HMAC

In this exercise, you will generate a keyed hash (i.e. MAC) for a file. You can use the -hmac option The following example command generates a keyed hash for a file using the HMAC-SHA256 algorithm. The string following the -hmac option is the key.

$ *openssl dgst –sha256 -hmac "abcdefg" filename*

Please generate a keyed hash using HMAC-SHA256, and HMAC-SHA1 for the text file that you created in the task above. In each case try keys of length 128, 160 and 256 bits and record your output. Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why?

3 [35 pts] Coding Task: Weak versus Strong Collision Resistance Property

In this task, we will investigate the difference between hash function's two properties: weak collision resistance property versus strong collision-resistance property. You will use the brute-force method to see how long it takes to break each of these properties. Instead of using openssl's command-line tools, you are required to write own programs to invoke the message digest functions in openssl's crypto library. A sample C code can be found from https://www.openssl.org/docs/manmaster/man3/EVP_DigestInit.html. Please get familiar with this sample code.

Since most of the hash functions are quite strong against the brute-force attack on those two properties, it will take us years to break them using the brute-force method. To make the task feasible, we reduce the length of the hash value to 24 bits. We can use any one-way hash function, but we only use the first 24 bits of the hash value in this task. Namely, we are using a modified one-way hash function. Please design an experiment to find out the following:

1. [15 pts] How many trials will it take you to break the weak collision resistance property using the brute-force method? You should repeat your experiment multiple times (100 or more depending on how long each trial takes), and tabulate your findings on number of trials in each experiment along with the statistics (average, median).

2. [15 pts] How many trials will it take you to break the collision-free property using the brute-force method? Similarly, to the above task repeat the experiment multiple times and and tabulate your findings on number of trials in each experiment along with the statistics (average, median).

3. [5 pts] Based on your observation, which property is easier to break using the brute-force method? Can you explain the difference in your observation mathematically?

4 [10pts] Observation Task: Encryption using different ciphers and modes

In this exercise, you 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. 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)

Obtain a simple picture in .bmp format. Encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Codebook) 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, you have to set it correctly, so the encrypted file can be treated as a legitimate .bmp file. Replace the header of the encrypted picture with that of the original

picture. You can use a hex editor tool (e.g. ghex or 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? Include both encrypted pictures and the original picture for your report for both CBC and ECB modes.

5 [35 pts] Coding Task: Encrypting with OpenSSL

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/decrypt 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 C code is available at the link below. Please get yourself familiar with this program, and then do the following exercise.
https://www.openssl.org/docs/manmaster/man3/EVP_EncryptInit.html.

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. An English word list *words.txt* is provided. 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 plaintext 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/ or the actual location in your system
LIB=/usr/local/ssl/lib/ or the actual location in your system
all:
```

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