Assignment 2
CS 370/ECE 499
Fall 2018

1 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. This project has a total of 5 tasks and 45 Points along with a bonus question for 5 points.

2. Submission Guidelines

You need to submit responses for the questions related to tasks 3.1 -- 3.5. Please submit your assignment as a single PDF file. Any submissions other than a single PDF file will not be graded.

3 Lab Environment and 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.

3.1 [5 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 -sha256, -sha512etc. In this exercise, you should try at least 3 different algorithms. 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 SHA512 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 SHA512 (SHA256).

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.

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 SHA512 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?

3.2 [5 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-SHA512 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?

3.3 [10 pts] 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 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, 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? Record both encrypted pictures and the original picture for your report for both CBC and ECB modes.

3.4 [10 pts] Observation Task: CBC Encryption using different IVs

IVs or initialization vectors act as a random seed to the block cipher modes of encryption like CBC. In this task you will understand the role of IVs.

- Step 1: Create any file with one line of text and name it plain.txt, the content of the file doesn't matter.
- Step 2: Now, using the command in the Q3.3, encrypt the file using aes-128-cbc scheme and write the output to the file 'encrypt1.bin'. You are free to choose the Key and IV in this step, but keep a note of them.
- Step 3: Now using the same Key and IV in the step 2, encrypt the file again and write the output to 'encrypt2.bin'.

- Step 4: Now using the same Key in Step 2 and a different IV, encrypt the file again and write your output to ‘encrypt3.bin’.

Now based on your observations of the above steps and the 3 different files that you generated, answer the following questions:

1. Does the contents of ‘encrypt1.bin’ match the contents of ‘encrypt2.bin’? Explain why or why not.
2. Does the contents of ‘encrypt1.bin’ match the contents of ‘encrypt3.bin’? Explain why or why not.

3.5 [20 pts] These set of questions are based on a real world vulnerability in an open source File Storage application called OwnCloud. Please refer to this blog post for the details about the vulnerability and answer the following questions in brief (less than 5 lines):

1. Please summarize the attack mentioned in the above blog post. Less than 150 words.
2. What encryption mechanism is being used in the above mentioned cryptosystem?
3. What shortcoming in the above mentioned cryptosystem and encryption scheme leads to the above mentioned vulnerability?
4. Give an example of an cryptosystem or encryption scheme that could have been used to avoid this vulnerability.
5. [Bonus 5 pts] Since the above vulnerability only allows for an attacker to insert 16 bytes of code, it is relatively harder to perform a real world attack with that code. However, an attacker can subvert the code flow to re-use the code blocks in program being attacked. Explain how does these kinds of attacks work and what are they called?