Lab 3: Symmetric encryption & hashing

Objectives:

- To perform symmetric encryption and hashing.

Submission:

- A lab report and other resources as requested.

Instruction:

In this lab, you will perform symmetric encryption and hashing utilising one of the most widely used crypto tools called Openssl. It is widely used in Internet web servers, serving a majority of all web sites. OpenSSL contains an open-source implementation of the SSL and TLS protocols. The core library, written in the C programming language, implements basic cryptographic functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available.

In this lab, we will use its different utility functions to perform symmetric encryption and hashing. You will be given a set of tasks for using Openssl to perform these functions. For this you will also need to use an HEX editor like GHex in Ubuntu. Follow the instructions, complete the tasks and prepare a report as per instructed.

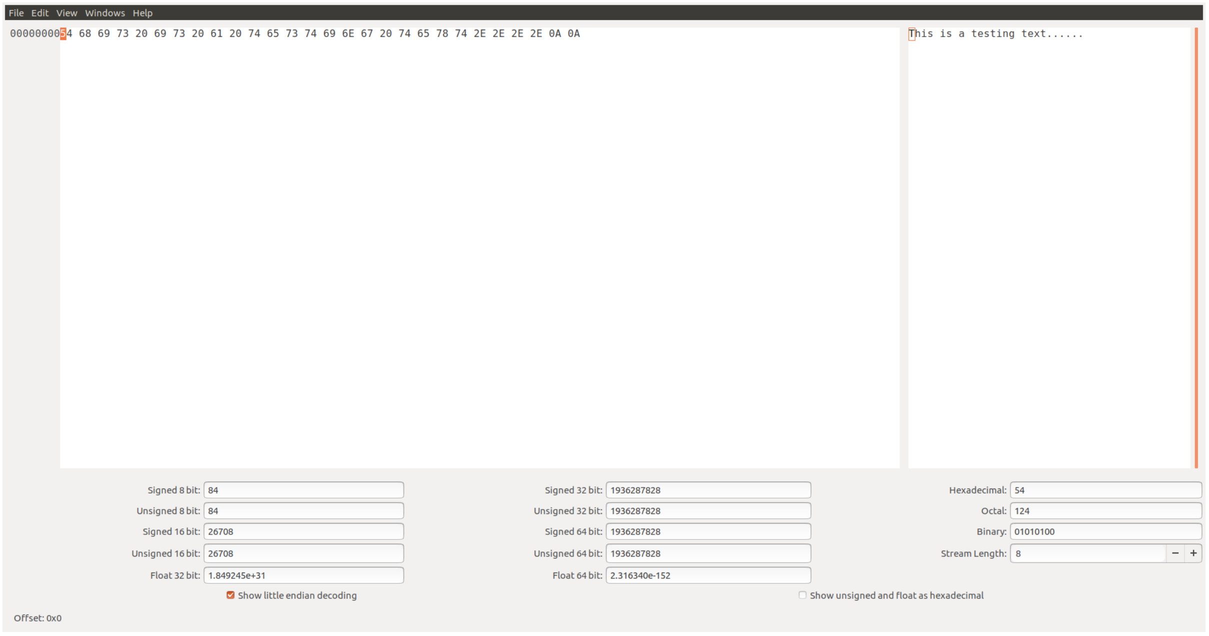
Setup:

At first, check if GHex is installed in your Ubuntu system either using the command **ghex** or application search bar.

If not installed, install ghex using the command: $ *sudo apt-get install ghex*

You can open a file with the ghex program using the command: *ghex filename &*

Once opened, the ghex window looks like the following figure.



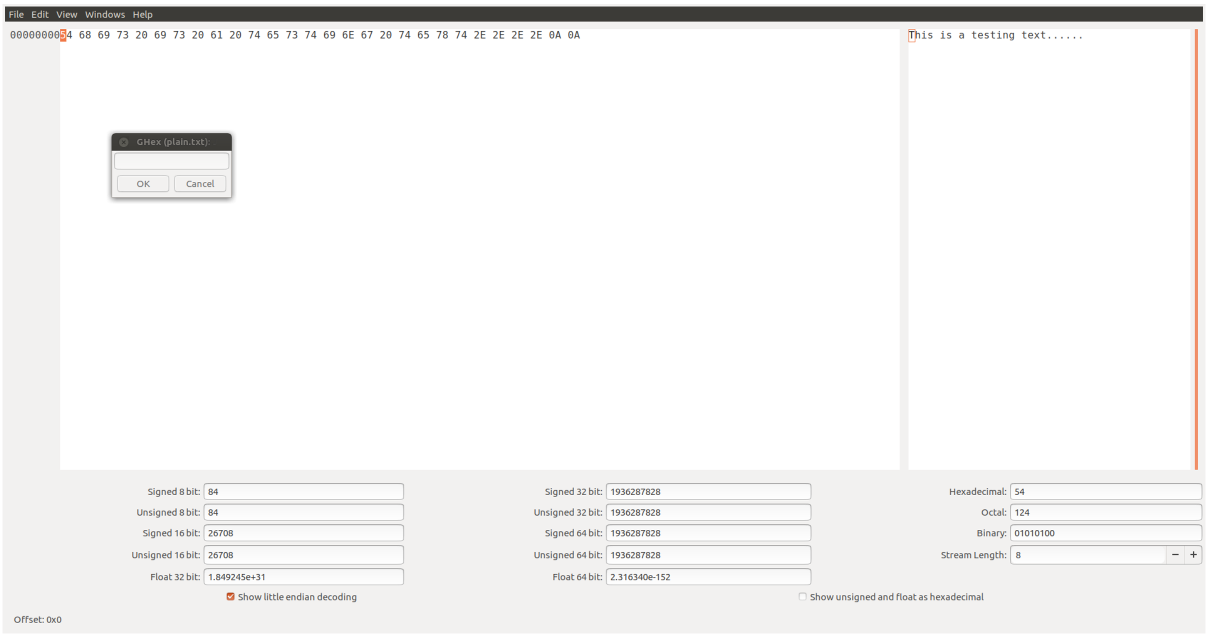
```
0000000054 68 69 73 20 69 73 20 61 20 74 65 73 74 69 6E 67 20 74 65 78 74 2E 2E 2E 2E 0A 0A        This is a testing text......
```

Signed 8 bit: 84
Unsigned 8 bit: 84
Signed 16 bit: 26708
Unsigned 16 bit: 26708
Float 32 bit: 1.849245e+31
Signed 32 bit: 1936287828
Unsigned 32 bit: 1936287828
Signed 64 bit: 1936287828
Unsigned 64 bit: 1936287828
Float 64 bit: 2.316340e-152
Hexadecimal: 54
Octal: 124
Binary: 01010100
Stream Length: 8
[✓] Show little endian decoding
[ ] Show unsigned and float as hexadecimal
Offset: 0x0

In the screenshot, a text file with the text "This is a testing text……" has been opened. The middle area shows the hexadecimal value of the file contents whereas the rightmost are shows the corresponding text contents in the file.

Create any text file using any editor (e.g. gedit text editor or nano) in Ubuntu and save it in your preferred location. Now, open the file using ghex. Familiarise yourself with the interface. Check and count how many hex bytes are displayed in the middle area. You can traverse the hex value by arrow keys in the keyboard or just clicking the mouse in the preferred location. You can change the content of any hex value by just typing into the value.

You can also go to the preferred byte position by clicking the *Goto Byte* in the Edit menu. It will show a Dialog box like the following figure. You can type in a decimal or hexadecimal value and Click OK.



```
0000000054 68 69 73 20 69 73 20 61 20 74 65 73 74 69 6E 67 20 74 65 78 74 2E 2E 2E 2E 0A 0A
```

```
This is a testing text......
```

```
Signed 8 bit: 84                 Signed 32 bit: 1936287828      Hexadecimal: 54
Unsigned 8 bit: 84               Unsigned 32 bit: 1936287828    Octal: 124
Signed 16 bit: 26708             Signed 64 bit: 1936287828      Binary: 01010100
Unsigned 16 bit: 26708           Unsigned 64 bit: 1936287828    Stream Length: 8
Float 32 bit: 1.849245e+31       Float 64 bit: 2.316340e-152
[✓] Show little endian decoding  [ ] Show unsigned and float as hexadecimal
Offset: 0x0
```

Openssl should be installed by default in Ubuntu. However, just check it by typing this command in the terminal: $ openssl. If it shows something like this: $OpenSSL>, Openssl is installed in your PC. Type q to quit the Openssl interface. If not, install Openssl using the following command: $ *sudo apt-get install openssl*.

With this setup, perform the following tasks.

Task – 1: AES encryption using different modes (2 Marks)

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, -aes-128-cfb, -bf-cbc, etc. 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)
```

In this task, you should encrypt using AES with three different modes for a given text file. At first, create a text file, add several lines of texts and save it to a preferred location. Complete this task by performing the encryption and generating an output file containing the encrypted output. Perform the decryption of the generated encrypted files to test that the encryption works well.

In your report, add the commands you have used to encrypt the files in three different modes. During your submission, add the three encrypted output file as well for me to check.

Task – 2: Encryption mode - ECB vs CBC (3 Marks)

The file pic original.bmp contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Encrypt the file 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. You can use ghex to directly modify binary files.
- Display the encrypted picture using any picture viewing software. Can you derive any useful information about the original picture from the encrypted picture? Explain your observations in your report. Also, attach the encrypted images during the submission

Task – 3: Encryption mode – corrupted cipher text (3 Marks)

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

- Create a text file that is at least 64 bytes long. Use the gedit text editor to create and save the file. gedit also allows you to count the byte number in the file.
- Encrypt the file using the AES-128 cipher.
- Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. You can achieve this corruption using a hex editor.
- Decrypt the corrupted file (encrypted) using the correct key and IV.

Answer the following questions with the commands used to encrypt and decrypt the file: (1) How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, 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. (2) Explain why. (3) What are the implication of these differences?

Add the answers in your report. Also, attach the plain and encrypted text files during the submission.

Task – 4: Padding (3 Marks)

For block ciphers, when the size of the plaintext is not the multiple of the block size, padding may be required. In this task, you will study the padding schemes by designing an experiment.

Use ECB, CBC, CFB, and OFB modes to encrypt a file (you can pick any cipher). Report which modes have paddings and which ones do not. For those that do not need paddings, explain why. Also, add the command utilised for this experiment in your report. Attach any created files during the submission.

Task – 5: Generating message digest (3 Marks)

In this task, we will play with various 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
```

Replace the *dgsttype* with a specific one-way hash algorithm, such as -md5, -sha1, -sha256, etc. You can find the supported one-way hash algorithms by typing *man openssl*.

For this task, create a text, add some texts and save it to a preferred location. Then, use the above command to create digest for this file for different algorithms. In this task, you should try at least 3 different algorithms, and describe your observations.

In your report, add the commands used to perform this task and attach the generated hash for each algorithm. In addition, add the text file during your submission.

Task – 6: Keyed hash and HMAC (3 Marks)

In this task, we would like to generate a keyed hash (i.e. MAC) for a file. In cryptography, an HMAC (sometimes expanded as either keyed-hash message authentication code or hash-based message authentication code) is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key.

We can use the -hmac option (this option is currently undocumented, but it is supported by openssl). The following example generates a keyed hash for a file using the HMAC-MD5 algorithm. The string following the -hmac option is the key.

```
% openssl dgst -md5 -hmac "abcdefg" filename
```

Create a text file and have some texts in to it. Generate a keyed hash using HMAC-MD5, HMAC-SHA256, and HMAC-SHA1 for the created file. Try several keys with different length. Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why?

Add the commands and their corresponding HMAC for the file in your report. Attach the text file during your submission.

Task – 7: Keyed hash and HMAC (3 Marks)

To understand the properties of one-way hash functions, we would like to do the following exercise for MD5 and SHA256:

- Create a text file of any length.
- Generate the hash value H_1 for this file using a specific hash algorithm.
- Flip one bit of the input file. You can achieve this modification using ghex.
- Generate the hash value H_2 for the modified file.
- Observe whether H_1 and H_2 are similar or not. Describe your observations in the lab report. **(Bonus: 2 Marks)** Write a short program to count how many bits are the same between H_1 and H_2.

Add the commands and the outputs of H_1 and H_2 in your report along with the text file.

Submission

The lab report containing the commands and reactions (and screenshots) as requested in each task along with the files required to execute each command and generate the same output as in the report.