**CS473: Network Security**

**Lab 3 - Spring 2024**

**Due Date: Task 1: 27th March at 2:35pm**

**Task 2-4: 28th March at 11:55 pm**

**Lead TA: Zoha Hayat**

**Total Marks: 50**

This lab has two sections. The first section deals with one way hash functions and MAC and the second section deals with symmetric key cryptography. The second section is an extension of the OpenSSL demo given in class.

**Instructions:**

**1. OpenSSL**

* OpenSSL is installed by default on Linux systems, MacOS and VM’s so you need to use them for this lab. You can use the HW1 VM for this lab. Alternatively, you can also use the Programming Studio or SA Lab PC’s for this lab. If you do not have a VM on your windows systems, you can use the following links to install it:

1. <https://www.youtube.com/watch?v=v1JVqd8M3Yc>
2. <https://www.geeksforgeeks.org/how-to-install-ubuntu-on-virtualbox/>

**2. Hex Editors:** You can either use bless editor, hexedit or any hex editor that you can find online or in the app stores. Alternatively, a Python file **corruptBit.py** has been provided to you. This is more efficient to use than other hex editors. **This will be used for Tasks in Section B.**

* It will be used to corrupt a single bit in a binary file and test its effects on different Cipher Types. In this file, we pick a byte offset from the binary file and randomly flip a bit from that byte to simulate the bit's corruption.
* **Usage:**

python corruptBit.py <input\_file\_path>.bin <byte\_offset> <corrupted\_file\_path>.bin

* Below is a demonstration given:

**Example:**

python corruptBit.py ciphertext\_ecb.bin 10 corrupted\_ecb.bin

**Output**

Full Hex Dump Before Corruption:

00000001 F0 E7 9A 7D F8 D9 EE ED 5A **[1B]** 31 29 DE 4E 3F 0E

00000017 CD 08 B7 6E 50 87 8E 2C A5 E4 DB E6 57 D5 63 4A

00000033 A4 CF 05 5B 0C B1 69 30 2A A4 B6 35 D0 7A ED EB

00000049 3F 23 FB FB 92 0A A4 50 21 8B BB 1D 7C 93 0D C8

Corruption Details:

Bit flip details:

Original byte in bits: 0 0 0 1 1 0 1 1

Corrupted byte in bits: 0 0 1 1 1 0 1 1

Bit at position 5 flipped from 0 to 1

Full Hex Dump After Corruption:

00000001 F0 E7 9A 7D F8 D9 EE ED 5A **[3B]** 31 29 DE 4E 3F 0E

00000017 CD 08 B7 6E 50 87 8E 2C A5 E4 DB E6 57 D5 63 4A

00000033 A4 CF 05 5B 0C B1 69 30 2A A4 B6 35 D0 7A ED EB

00000049 3F 23 FB FB 92 0A A4 50 21 8B BB 1D 7C 93 0D C8

**3. PyCryptodome**

* Library can be installed using: **pip install -U PyCryptodome**. This allows you to use the AES block ciphers in openssl commands.

**Submission Guidelines:**

1. You have to submit Section A Task 1 during lab timings by **2:35 pm on 27th March, 2024**. Submit a pdf file with the format Name\_RollNumber\_Lab3\_Task1.pdf on LMS which contains the explanations for the task.
2. For the remaining lab, submit a pdf file following the format Name\_RollNumber\_Lab3.pdf on LMS by **28th March, 2024** at **11:55 pm** containing the required explanations for each task. Also add a zipped folder with all the files you used for each task.
3. Please make sure that your submissions include appropriate explanations for your reasoning and also include all files that you used in the tasks. Please organise your files according to the task number.
4. You are not required to submit screenshots of your tasks.

**Section A**

**Task 1: Generating Message Digest and MAC (10 Marks)**

In this task, we delve into the exploration of various one-way hash algorithms. To access the manual for OpenSSL commands, simply type "man openssl" in your terminal. You can utilise the following openssl command structure to generate the hash value for a specific file:

**openssl dgst dgsttype filename**

Ensure to replace 'dgsttype' with your desired one-way hash algorithm, which are "-sha1", and "-sha256". In this task, you have to experiment with these two algorithms using two .txt files of size 10 bytes and 40 bytes and carefully document your observations. You can refer to the supported one-way hash algorithms by utilising the command "man openssl". You are supposed to do the following for this task:

1. Provide the digest generated and the digest size in bits for each of the hash functions you experimented with, given a particular file. (Tip: Remember, two hexadecimal digits represent one byte. For instance, "a64d56" spans three bytes.)
2. Investigate whether the digest size varies for the files of different lengths and complete the following table.

|  | **Digest + Digest Size (10 bytes)** | **Digest + Digest Size (40 bytes)** | **Varies by file length Yes/No?** |
| --- | --- | --- | --- |
| **SHA-1** |  |  |  |
| **SHA-256** |  |  |  |

**Submission for Task 1:** Please submit the filled table and the .txt files you used in your task.

**Task 2: Keyed Hash and HMAC (10 Marks)**

In this task, our objective is to generate a keyed hash, also known as a Message Authentication Code (MAC), for a given file. HMAC (Hash-based Message Authentication Code) is a commonly used method for creating MACs, which uses a cryptographic hash function in combination with a secret key.

**Background:**

HMAC : Since we haven’t covered HMAC in the lectures, a brief introduction to HMAC from Ross Anderson’s Security Engineering book is as follows:

“Hash functions have many other uses. One of them is to compute MACs. A naive method would be to hash the message with a key: MACk(M) = h(k,M). However the accepted way of doing this, called HMAC, uses an extra step in which the result of this computation is hashed again. The two hashing operations are done using variants of the key, derived by exclusive-or’ing them with two different constants. Thus HMACk(M) = h(k⊕B,h(k⊕A,M)). A is constructed by repeating the byte 0x36 as often as necessary, and B similarly from the byte 0x5C. If a hash function is on the weak side, this construction can make exploitable collisions harder to find [888]. HMAC is now FIPS 198-1.”

You are encouraged (but not required) to read further on HMACs.

We can accomplish creating HMACs by utilising the "-hmac" option within OpenSSL, even though it's currently undocumented, it is indeed supported. Below is an example command demonstrating how to generate a keyed hash for a file using the HMAC-MD5 algorithm. The string following the "-hmac" option represents the key:

**openssl dgst -md5 -hmac "abcdefg" filename**

For this task, we'll generate keyed hashes using HMAC-SHA256, and HMAC-SHA1 for any **one** selected text file. Please experiment with **3 keys of different lengths** for each algorithm to observe how they affect the resulting MAC. You are supposed to do the following for this task:

1. Explore generating keyed hashes with HMAC-SHA1, and HMAC-SHA256 algorithms using 3 different key lengths.
2. Answer the following questions:
   * Do we need to adhere to a fixed key size in HMAC?
   * If so, what is the required key size?
   * Pick one of the keys you used, and write out the variants of k⊕A and k⊕B that are used in the HMAC. (Just writing the expression should suffice. No need for the actual computation). (Refer to the reading material section for this).

**Submission for Task 2:** Please submit details of the different sized keys you used for this task, the .txt file you used and your answers to the above questions.

**Task 3: Checksums (10 Marks)**

In this task, you are provided with a .txt file along with two corresponding checksums: one generated using SHA-256 and the other using HMAC-MD5. Your objective is to verify whether the file's integrity is intact by comparing these checksums using OpenSSL. Use the template table given below for this task.

Key used for HMAC-MD5: alphabetagamma

Checksum from HMAC-MD5: 9e5f127cd6d1df4b07faf5c7945d4d6f

Checksum for sha256: 0c4bfd1603904b1cc9239f58c78fed177ee6230233174d8df42f36cc3bc27cc0

|  | **Command Used** | **Correct or Not (Yes/No) + Checksum values** |
| --- | --- | --- |
| **SHA-256** |  |  |
| **HMAC-MD5** |  |  |

**Submission for Task 3:** Please submit the filled table given above.

**Section B**

**Task 4A: Encryption Mode – Corrupted Cipher Text (10 Marks)**

To explore the behaviour of different encryption modes when a single bit of the encrypted data is corrupted, you need to do the following:

1. Create a text file that is at least 64 bytes long.

2. Encrypt the file using AES-128 cipher.

3. Corrupt a single bit of the 24th byte in the encrypted file (you’ll be using **corruptBit.py**).

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

5. Answer the following questions for each encryption mode in the table below:

a. How much information can you recover by decrypting the corrupted file if the encryption mode is ECB, CBC, or OFB, respectively? Give your answer in the form of an approximate percentage or number of blocks. For example, if your text file has a total of 4 sentences and more than 3 sentences are being recovered, you can say “more than 50% information is being recovered” and give relevant reference to your decrypted text.

b. Explain your observations and why you think this is happening.

c. Discuss the implications of these differences.

| **Cipher Type** | **a) Information Recovery** | **b) Observations after task** | **c) Implications** |
| --- | --- | --- | --- |
| **ECB** |  |  |  |
| **CBC** |  |  |  |
| **OFB** |  |  |  |

**Submission for Task 4A**:

- Create separate folders for each encryption mode within the task 4A folder.

- For each encryption mode, submit the following:

1. The OpenSSL key, IV, and encrypt/decrypt commands used.

2. The text file created for encryption.

3. The original ciphertext file.

3. The corrupted ciphertext file.

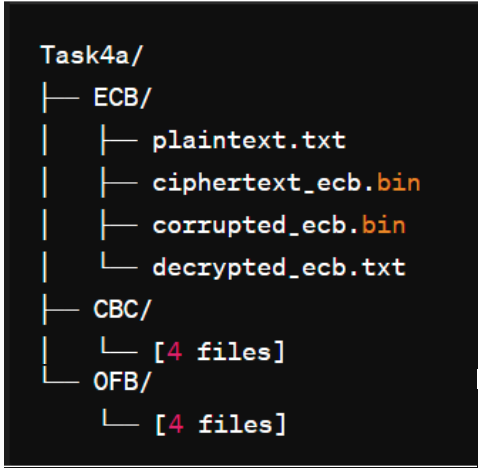
4. The decrypted text file.

5. The filled table for both Task 4A.

- Place all relevant files (as mentioned above) into their respective folders.

- Include these folders in your final submission.

Note: Here is an example of how Task4A folder is supposed to be organised:

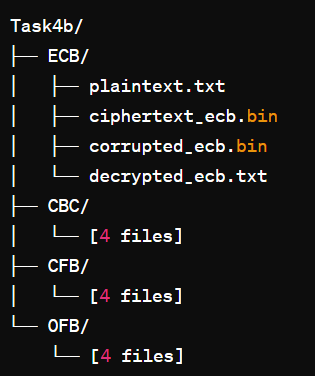


**Task 4B: (10 Marks)**

For this task, you need to explore the corruption of bytes in two blocks instead of one. Follow the steps 1 to 5 from Task 4A but instead corrupt the 24th **and** 50th byte, and compare the ECB and CBC modes **ONLY**.

| **Cipher Type** | **a) Information Recovery** | **b) Observations after task** | **c) Implications** |
| --- | --- | --- | --- |
| **ECB** |  |  |  |
| **CBC** |  |  |  |

**Submission for Task 4B:** Follow the same submission guidelines as in Task 4A. The folder structure is as follows:



**Optional Task:**

You can try to follow the MD5 Collisions Demo here: https://www.mathstat.dal.ca/~selinger/md5collision/

This part is only for your own learning and requires no submission.

There is no bonus for completing this task.