Assignment 1 – Secret Key, One-Way Hash and MAC

1. Overview

The learning objective of this assignment is for students to get familiar with the concepts in the secret-key encryption, one-way hash functions and Message Authentication Code (MAC). After finishing the assignment, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initial vector (IV) in addition to gaining a deeper understanding of the concepts, students should be able to use tools and write programs to generate one-way hash value and MAC for a given message. Moreover, students will be able to use tools and write programs to encrypt/decrypt messages.

2. Work Environment

The easiest way to go about working in this course would be to install a Linux distribution (preferably Ubuntu) on your laptop/pc – either Dual Boot or use a VM. For this assignment, we are assuming that you are working in Linux. In this assignment, we will be using the *OpenSSL* library for encryption and *hexcurse* as a hex editor. With *hexcurse* we will be able to view and modify files of binary format. It allows the user to load data from any file, view and edit it in either hex or ascii.

You can use the following commands to navigate:

F2 or CTRL+s > Save
F3 or CTRL+o > Open
F4 or CTRL+g > Goto
F5 or CTRL+f > Find
F8 or CTRL+q > Exit

3. Part 1: Primer to *OpenSSL* library

a. Task 1: Generating Message Digest and MAC

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
```

b. Task 2: 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 *ciphertype* with a specific cipher type, such as -aes-128-cbc, -des-cbc, -des-ecb, 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)

c. Task 3: Encryption Mode – ECB vs. CBC

The file *picoriginal.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. 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 *hexcurse* 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.

d. Task 4: The Randomness of One-Way Hash

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

1. Create a text file of any length.
2. Generate the hash value H_1 for this file using a specific hash algorithm.
3. Flip one bit of the input file. You can achieve this modification using *hexcurse*.
4. Generate the hash value H_2 for the modified file.
5. Please observe whether H_1 and H_2 are similar or not. Please describe your observations in the assignment report. You can write a short program to count how many bits are the same between H_1 and H_2.

4. Part 2: Using *OpenSSL* in our programs

a. Task 1: One-Way Property vs. Collision-Free Property

In this task, we will investigate the difference between hash function's two properties: one-way property versus collision-free property. We 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 your own C programs to invoke the message digest functions in *OpenSSL's* crypto library. Sample code has been provided with this (*sampletask1.c*). 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. How many trials it will take you to break the one-way property using the brute-force method? You should repeat your experiment for multiple times, and report your average number of trials.
2. How many trials it will take you to break the collision-free property using the brute-force method? Similarly, you should report the average.
3. Based on your observation, which property is easier to break using the brute-force method?
4. (10 Bonus Points) Can you explain the difference in your observation mathematically?

Use *"gcc -o Opentest Opentest.c –lcrypto"* to compile your code.

b. Task 2: Programming using the 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/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. Sample code has been provided with this (*sampletask2.c*). 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 use the *words.txt* that we have given. The plaintext and ciphertext is in the following:

Plaintext (total 21 characters): This is a top secret.
Ciphertext (in hex format): 8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9

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 *hexcurse* 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. Use *"gcc -o enc enc.c -lcrypto –ldl"* to compile your code.

5. Part 3: Secure Email System using PKI (Design Problem)

You need to design a fully secure email system that meets these requirements, if possible:

- Authentication
- Authorization
- Confidentiality
- Data/Message Integrity
- Accountability
- Availability
- Non-repudiation

If you feel that any of the requirements cannot be met, then briefly explain why it would not be possible to meet it.

NOTE 1: anybody who wants to and is capable enough, will be able to intercept your emails. These emails are being sent over the Internet, and nothing is safe there.

NOTE 2: you can have a manager to keeps track of everything or you can design the protocol such that it remains secure. But in case you have a manager, how would you make sure that it remains secure (what if I hack into it or own it – even the developer/owner of the manager should not be able to read/modify any email)?

NOTE 3: this is a descriptive design problem, so you do **NOT** need to code anything. Please be clear and concise in your explanation. And there are no points for generic hand-wavy answers and long stories.

6. Part 4: Decrypt Vignere Cipher

A vignere cipher is a polyalphabetic substitution cipher that was commonly used in the past to hide messages. In this part of the assignment, you will be required to decrypt a ciphertext. Wikipedia is a good starting point for a description of the algorithm and for some hints on how to break it.

Please decrypt the ciphertext given in the *ciphertext.txt* file. For this part, along with code, write in your report the key, the plaintext, and a small paragraph describing the approach you used. You must code in C/C++ and not use any external libraries (just use the standard C/C++ libraries).

Hint: try to determine the length of the key first. After that, simple frequency analysis should do the trick.

7. Submission

Submit a report on LMS answering every question listed in the assignment along with the commands used. Also submit any code that you have to write (Part 2 and Part 4). Collaboration of **ANY** sort is not allowed, any cases of plagiarism will be reported to the Disciplinary Committee.