**CIS 458 System Security**

**Lab 02 – Hash and MAC**

**Time Due:** Friday, 9/20/2019 in Blackboard

Please answer all questions in the Answer Sheet; put your answers in a Word or PDF or a text file; submit your work by the due time through Blackboard to receive full credits for this assignment.

**Educational Objectives**

The learning objective of this lab is for students to gain hands-on experience with one-way hash functions and Message Authentication Code (MAC). After finishing this lab, in addition to gaining a better understanding of the concepts, students will be able to

* Use the OpenSSL commands in the message digest (dgst) family.
* Program with the APIs of OpenSSL in the message digest (MD) family to generate one-way hash values and MAC for a given message or messages.

**Lab Environment**

This lab will be performed on a DC computer. Actually, any Linux system will be fine.

**Section 1 Introduction**

The cryptographic libraries implemented by OpenSSL project will be used in this lab. Detailed description about OpenSSL and the libraries can be found from the following link:

<https://www.openssl.org/>

By default, OpenSSL is installed in most Linux distributions. Many applications have been using OpenSSL libraries and APIs.

A list of OpenSSL commands implemented in the current version (version 1.1.1) can be found from the following link:

<https://www.openssl.org/docs/man1.1.1/man1/>

The usage of each command will be shown when you click on the corresponding hyper link.

**Section 2 OpenSSL commands**

The command **openssl dgst** is commonly used to perform digest (hash and MAC) operations. More specifically, you can use the following command to generate a hash value of a file:

openssl dgst *-algorithm* *filename*

where

*-algorithm:* is the algorithm of hash function.

*filename:* is the file name of the file for whose contents you want to generate a hash value using the algorithm -*algorithm*.

The option -hmac can be used to generate a keyed hash with a chosen algorithm. For example, the following command will generate hash value with the key “*MyPassword*”:

openssl dgst *-algorithm* -hmac “*MyPassword*” *filename*

More details about this command can be found by typing man openssl dgst or man openssl and then man dgst or from the following link:

<https://www.openssl.org/docs/man1.1.1/man1/dgst.html>

The following command will show you all hash functions supported by the openssl dgst command:

openssl list -digest-commands

**Question 01:** Give at least three hash functions supported by the openssl dgst.

**Section 3 Hash Generation with OpenSSL Commands**

In this section, you will get familiar with the OpenSSL commands for generating hashes and keyed hashes with different hash functions. In order to do so, please do the following:

* Create a file of your choice. Any file should work. For example, you can copy over a file you have such as an image, a text file and so on.
* Generate a hash value of the file using the command introduced in the previous section. Please play with at least three hash functions supported by the openssl dgst command.

**Question 02:** What are the hash functions you used to generate hash values. Did they work as expected?

* Generate a keyed hash value of the file using the command introduced in the previous section. Please play with at least three hash functions supported by the openssl dgst command.

**Question 03:** What are the hash functions you used to generate the keyed hash values? What is/are the key(s) you used?

**Section 4 The Randomness of One-Way Hash**

One of the properties of a cryptographic hash function is collision resistance. In other words, given any pair of messages *m1* and *m2*, *m1 ≠ m2*, then *H(m1) ≠ H(m2)*.

You will test this property in this section with md5 and sha256. In order to do so, we want to do the following exercise with md5 and sha256:

* Generate a text file of your choice. You can copy and paste an existing one.
* Generate the hash value *h1* for this file using a specific hash algorithm (md5 or sha256).
* Make minor changes to the file (change one letter, for example).
* Generate the hash value *h2* for the changed file.
* Please examine whether *h1* and *h2* are similar or not for each algorithm.

**Question 04:** Please describe your observations briefly. Please attach a screenshot to support your observations.

**Section 4 Message Authentication**

As discussed in the class, message authentication/data integrity check can be depicted in the following diagram:



Please write a script, or a program if you choose so, to implement the message authentication protocol.

You can work with a partner for this part or on your own. If you work with a partner, one works as the sender and writes the sender’s part of the script, the other works as the receiver and writes the receiver’s part of the script. If you work on your own, you need to write both. Then test your scripts by generating a MAC as the sender, then verify/authenticate the message as the receiver. In your scripts, you don’t need to write the message transmission part. Use cp or other method to send the message and MAC to receiver.

**Question 05:** Attach your script of the sender’s part.

**Question 06:** Attach your script of the receiver’s part.

**Question 07:** Briefly describe how the script at the receiver’s side works and how to test your script.

**Section 5 One-Way And Strong Collision Resistance**

As discussed in class, a good cryptographic hash function (*H*) must meet the following requirements:

* **One-way resistance:** For any given hash value *h*, it is computationally infeasible to find a message *x* such that *H(x) = h*.
* **Collision resistance (collision free):** It is computationally infeasible to find any pair of messages (*x, y*) such that *x* *≠ y* and *H(x) = H(y)*.

However, it is theoretically impossible to design such a hash algorithm to meet these properties because the size of hash value is limited. In practice, it is computationally inhibitive, however, to break these two properties when the size of hash value is large.

In this section, we will use the brute-force method to see how long it takes to break each of these properties. At the same time, we will observe the difference between these two properties. Instead of using OpenSSL commands, you are required to write your own programs to invoke the message digest functions (APIs) in OpenSSL crypto library. This activity will help you gain hands-on experience with OpenSSL APIs. The message digest functions (APIs) in OpenSSL crypto library and a sample C program to invoke those functions to compute the hash value for a message can be found from the following link:

<https://www.openssl.org/docs/manmaster/man3/EVP_DigestInit.html>

Please read the web page and understand the functions and the sample code.

Note:

* C is not required. You can choose the language you are more familiar with to write your program.
* You can use the following command to compile a C program using OpenSSL functions.

gcc -o *hcracker* -l crypto *hash\_cracker.c*

where gcc is the C compiler, *hcracker* is the executable name, *hash\_cracker.c* is the name of your C program.

Since most of the hash functions are quite strong against brute-force attack on those two properties, it will take you years to break them. To make the lab feasible, we reduce the length of the hash value to 24 bits (three characters). We can use any one-way hash functions that are included in the OpenSSL library, but we use the first 24 bits of the hash value only in this lab. In other words, we are using a modified one-way hash function. Please design an experiment and implement your design to find the following:

**Question 08:** How many trials it will take you to break the one-way property with the brute-force method? To make your answer reliable, you should repeat your experiment for multiple times (e.g. 5-10 times) and report the average number of trials. Use screenshot to demonstrate your work if you have.

**Question 09:** How many trials it will take you to break the collision resistance property with the brute-force method? Similarly, to make your answer reliable, you should repeat your experiment for multiple times (e.g. 5-10 times) and report the average number of trials. Use screenshot to demonstrate your work if you have.

**Question 10:** Based on your observation, which property is easier to break by using the brute-force method? Please justify your answer. Use screenshot to demonstrate your work if you have.

**Question 11:** (20 points) Please turn in your program code to Blackboard.

**Congratulations!** You have completed this lab for the class. Hope you enjoyed doing this lab. Please let your instructor know if you have any comments.

**Answer Sheet**

========================== **Required Questions** ===========================

**Question 01:** Give at least three hash functions supported by the openssl dgst.

**Question 02:** What are the hash functions you used to generate hash values. Did they work as expected?

**Question 03:** What are the hash functions you used to generate the keyed hash values? What is/are the key(s) you used?

**Question 04:** Please describe your observations briefly. Please attach a screenshot to support your observations.

**Question 05:** Attach your script of the sender’s part.

**Question 06:** Attach your script of the receiver’s part.

**Question 07:** Briefly describe how the script at the receiver’s side works and how to test your script.

**Question 08:** How many trials it will take you to break the one-way property with the brute-force method? To make your answer reliable, you should repeat your experiment for multiple times (e.g. 5-10 times) and report the average number of trials. Use screenshot to demonstrate your work if you have.

**Question 09:** How many trials it will take you to break the collision resistance property with the brute-force method? Similarly, to make your answer reliable, you should repeat your experiment for multiple times (e.g. 5-10 times) and report the average number of trials. Use screenshot to demonstrate your work if you have.

**Question 10:** Based on your observation, which property is easier to break by using the brute-force method? Please justify your answer. Use screenshot to demonstrate your work if you have.

**Question 11:** (20 points) Please turn in your program code to Blackboard.