One-Way Hash Function and MAC Lab

Overview

The learning objective of this lab is for students to get familiar with one-way hash functions and Message Authentication Code (MAC). After finishing the lab, 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. This lab covers a number of topics described in the following:

- One-way hash function and MAC
- The collision-resistance property
- Collision attacks

Lab Environment. Please use the same virtual machine that you used for Lab 1, which can be downloaded from the SEED website: https://seedsecuritylabs.org/Labs_16.04/Crypto/Crypto_Encryption/

1 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
```

Please replace the dgsttype with a specific one-way hash algorithm, such as -md5, -sha1, -sha256, etc. In this task, you should try **at least 3 different algorithms**, and describe your observations. You can find the supported one-way hash algorithms by typing "man openssl".

Submission for Task 1 (20 points)

(1) Please list hash algorithms and commands that you use in this task. (10 points)

(2) Please also list the corresponding outputs for each command. Please also describe your observation. (10 points)

2 Task 2: Keyed Hash and HMAC

In this task, we would like to generate a keyed hash (i.e. MAC) for a file. 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
```

Please generate a keyed hash using HMAC-MD5, HMAC-SHA256, and HMAC-SHA1 for any file that you choose.

Submission for Task 2 (20 points)

Please try several keys with different length and answer following questions:

(1) What are commands that you use in this task? What are the corresponding hash values? (10 points)

(2) Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why? (10 points)

3 Task 3: 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 `H1` for this file using a specific hash algorithm.
3. Flip one bit of the input file. You can achieve this modification using `ghex` or `Bless`.
4. Generate the hash value `H2` for the modified file.
5. Please observe whether `H1` and `H2` are similar or not. Please describe your observations in the lab report. You can write a short program to count how many bits are the same between `H1` and `H2`.

Submission for Task 3 (20 points)

(1) What is the text file you generate in step 1? (5 points)

(2) Which hash algorithm you are using? (5 points)

(3) Which bit was modified in this task? Please attach the screenshots of original file in binary (using ghex or Bless) and of modified file. (5 points)

(4) Please describe your observation. Are `H1` and `H2` similar? How many bits are the same between `H1` and `H2`? (5 points)

4 Task 4: One-Way Property versus 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 a program to invoke the message digest functions in openssl's crypto library. Please feel free to use any programming language that you are familiar with to conduct this task. For example, if you plan to write a C program, a sample code can be found from http://www.openssl.org/docs/crypto/EVP_DigestInit.html. If you plan to use Python to write your program, the package PyCryptodome could be used to generate hash value: https://pycryptodome.readthedocs.io/en/latest/index.html .

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.

Task 4.1: One-Way Property

One-way property (also known as preimage resistant) is that given the hash value h it is computationally infeasible to find a message y such that the hash value of y is identical to the given h (i.e., H(y) = h). In this task, you will write a program to break the one-way property using the brute-force method. Please design an experiment to find out the following: How many trials it will take you to break the one-way property using the brute-force method?

Hints: You can first select a message and generate the hash value (denoted as h) of this message using a specific hash algorithm (such as MD5 or SHA 256). Then you can randomly generate string and obtain the hash value h1 of the random string. If the first 24 bits of h1 is identical to the first 24 bits of h, then you break the one-way property. How many random strings you have tried to break the one-way property? You should repeat your experiment for multiple times, and report your average number of trials.

Submissions for Task 4.1 (20 points)

(1) Theoretically, how many attempts on average are needed to brute force one-way property (you can check the slide 13 of Module 07 to find the answer)? Please note that we will reduce the length of the hash value to 24 bits. (5 points)

(2) Please explain how you design your program and attach the screenshot of your program. (10 points)

(3) How many trails on average it will take you to break the one-way property using brute-force method? (5 points)

Task 4.2: Collision-Free Property

Collision-free property (also known as strong collision resistant) is that it is computationally infeasible to find any pair (x, y) such that H(x) =H(y). In this task, you will write a program to break the collision-free property using brute-force method. Please design an experiment to find out the following: How many trials it will take you to break the collision-free property using the brute-force method?

Hints: You can first randomly generate a string, obtain the corresponding hash value, and store the first 24 bits of hash value. Then you continue to randomly generate strings and store the first 24 bits of hash values until you get a 24-bits value that has already been stored. That is, you find two strings which have same hash values (the first 24 bits of hash values are identical). Similarly, you should report the average.

Submissions for Task 4.2 (20 points)

(1) Theoretically, how many attempts on average are needed to brute force collision-free property (you can check the slide 13 of Module 07 to find the answer)? Please note that we will reduce the length of the hash value to 24 bits. (5 points)

(2) Please explain how you design your program and attach the screenshot of your program. What are two random string that have same first 24 bits of hash values? (10 points)

(3) How many trails on average it will take you to break the collision-free property using brute-force method? (5 points)

Bonus Points (10 points)

(1) Based on your observation, which property is easier to break using the brute-force method?

(2) Can you explain the difference in your observation mathematically?