Lab 2: OpenSSL Hash Algorithms & Implementation

1 Overview

The key objective of this lab is to understand the range of hashing methods used, analyse the strength of each of these methods, and in the usage of salting. Overall the most popular hashing methods are: MD5 (128-bit); SHA-1 (160-bit); SHA-256 (256-bit); SHA-3 (256-bit), bcrypt (192-bit) and PBKDF2 (256-bit). The methods of bcrypt, scrypt and PBKDF2 use a number of rounds, which significantly reduces the hashing rate. This makes the hashing processes much slower, and thus makes the cracking of hashed passwords more difficult. We will also investigate the key hash cracking tools such as hashcat and John The Ripper.

2 Objectives

The learning objective of this lab is for students to get familiar with the concepts of cryptographic hash function. After finishing the lab, students should be able to gain first-hand experience on hash algorithms (unkeyed or symmetric keyed). Moreover, students will be able to use tools and write commands to hash/authenticate messages and files.

3 Lab Environment

In this lab, we will use `openssl` commands such as dgst, req, verif

The systanx of openssl command is:

```
openssl command [options] [arguments]
```

Installing a hex editor. In this lab, we need to be able to view and modify files of binary format. You can install a hex editor. It allows the user to load data from any file, view and edit it in either hex or ascii.

4 Hashing using OpenSSL

Hash functions are mathematical algorithms designed to take data as input and generate a fixed-size, unique string of characters, also known as the hash. Moreover, they are designed to be fast, and are very hard to reverse; it is very hard to recover the data that created any given hash, based on the hash alone. Another important property of the hash function is that even the smallest change done to the input data yields a completely different hash.

You can find the meaning of the command-line options and all the supported dgst types by typing `"openssl dgst -help"`. We include some common options for the `openssl dgst` command in the following:

```
dgst [options] [file...]
  -*                  Any supported digest
 -out outfile        Output to filename rather than stdout
 -hex                Print as hex dump
```

```
-binary              Print in binary form
 file... files to digest (default is stdin)
```

In this exercice, we have two objectives:

1. Part 1: Hashing a Text File with OpenSSL
2. Part 2: Verifying Hashes

4.1 Part 1: Hashing a Text File with OpenSSL

Type the command below to list the contents of any text file on the screen: **cat filename**

From the terminal window, issue the command below to hash the text file. The command will use SHA2-256 as the hashing algorithm to generate a hash of the text file. The hash will be displayed on the screen after being computed by the OpenSSL.

```
openssl sha256  filename
```

OpenSSL displays the hashing algorithm used, SHA–256, followed by the name of the file used as input data. The SHA–256 hash itself is displayed after the equal ('=') sign.

Hash functions are useful for verifying the integrity of the data regardless of whether it is an image, a song, or a simple text file. The smallest change results in a completely different hash. Hashes can be calculated before and after transmission, and then compared. If the hashes do not match, then this indicates that the data was modified during transmission (or storing). A hashing algorithm with a longer hash value bit-length, such as SHA2-512, can also be used. To generate a SHA2-512 hash of any file, use the command below:

```
openssl *  filename
```

where * represents the implemented hash function. List the available digest algorithms in OpenSSL such as sha512 can be obtained by using the following command:

```
openssl list -digest-algorithms
```

3. Let us modify any byte in the text file and recalculate the hash. Now that the file has been modified and saved, run the same command again to generate a SHA2-256 hash of the file. Is the new hash value different compared to the previous one? Please explain how is it different?

4. Use sha256sum and sha512sum to generate SHA-2-256 and SHA-2-512 hash of any new file two times. Do the hashes generated with the same hash algorithm match? Explain.

5. Hash the string "Hello from the other side" using SHA3-512

```
echo -n  "Hello from the other side" | openssl dgst -sha3-512
```

6. Determine the number of hex characters in the following hash values.
    - MD5 hex chars:
    - SHA-1 hex chars:
    - SHA-256 hex chars:
    - SHA-384 hex chars:
    - SHA-512 hex chars:
7. How does the number of hex characters relate to the length of the hash?

Note: SHA–2 is the recommended standard for hashing. Though SHA-2 has not yet been effectively compromised, computers are tending to become more and more powerful. It is expected that this natural evolution will soon make it possible for attackers to break the SHA-2. **SHA-3 is the newest hashing algorithm that will eventually be the replacement for SHA-2 family of hashes.**

4.2 Part 2: Verifying Hashes

As mentioned before, a common use for hashes is to verify the file's integrity. List the required steps to verify the integrity of any file (software) that you downloaded from the Internet. Explain.

The MD5 signatures of a set files are presented in the following:

- MD5(1.txt)= 5d41402abc4b2a76b9719d911017c592
- MD5(2.txt)= 69faab6268350295550de7d587bc323d
- MD5(3.txt)= fea0f1f6fede90bd0a925b4194deac11
- MD5(4.txt)= d89b56f81cd7b82856231e662429bcf2

Detect which file(s) have been modified?

4.3 Part 3: Working with keyed Hashes

Some well-known message authentication codes, like CMAC or HMAC, are implemented in OpenSSL. For instance, you can compute the HMAC of a message with the following command:

```
openssl dgst -hashAlgorithm -mac HMAC -macopt hexkey:..... file
```

In addition, you can compute the CMAC of a message with the following command:

```
openssl dgst -mac cmac -macopt cipher:* -macopt hexkey:$mykey file
```

The option -macopt is used to specify the parameters of HMAC or CMAC, like the key and the block cipher (case of CMAC), with the syntax -macopt name:value. The length of the key must match the one required by the specified block cipher. If you want to compute the CMAC or HMAC of a file (not the standard input), you only need to provide the filename.

where * represents here the block cipher such as aes-128-cbc.

8. Create a MAC value by using the HMAC-SHA384 of a file "rand.txt" and a specific key (369bd7d655).
9. Create a MAC value by using the HMAC - SHA512 of he string "Hello from the other side".