Assignment 2 One-way hash

Hampus Nilsson

Spring Term 2023

1 Tasks

1.1 Task 2.1

In this task, we will experiment 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
Replace the dgsttype with a specific one-way hash algorithm, such as -md5, -sha1, -sha256. In this task, you should try these three algorithms (MD5, SHA1 and SHA256). Create a file consisting of your KTH email address in the format username@kth.se, with UTF-8 encoding. Generate message digests for this file with all three algorithms

1.1.1 Question 1

- **Question:** Describe your observations. What differences do you see between the algorithms?
- **Answer:** The hashed output is output in hexadecimals. The main difference between them is number of characters used. md5 clearly uses the least amount of characters as hashed output while sha256 uses the most characters. **Number of characters used:**
  md5: 32
  sha1: 40
  sha256: 64

1.1.2 Question 2

- **Question:** Write down the digests generated using the three algorithms.
- **Answer:**
  md5: 11a95d322edca7ff3e541b693938e771
  sha1: 4026664c77ae4c8d47829e29a854edb8daaddd1d
  sha256: 5527d8a41e0f3cf019cacef28f3fb8bf411c1e42e9f69b7fe575b168e6f0c055

1.2 Task 2.2

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
Generate a keyed hash using HMAC-MD5, HMAC-SHA256, and HMAC-SHA1 for the file created in Section 2.1. Try several keys of different length

1.2.1 Question 3

- **Question:** Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why?

- **Answer:** The entered key can be of any length however the hashing algorithm will process the key to the point where it would achieve 64 bytes in byte-length. In a sense, there is a fixed size of 64 bytes but the user may enter any key without problem. If the key was not to reach a byte length of 64 then the remaining bytes would be appended as 0x00. If the entered key would exceed 64 bytes the the algorithm would first hash the key in itself and the use the hashed key for hashing the input data. [RFC 2104]

1.2.2 Question 4

- **Question:** Now use the string IV1013-key as the secret key and write down the keyed hashes generated using the three algorithms.

- **Answer:**
  md5: 80781e0a87a24ad22da7c909fb21671d
  sha1: fda0509ce521186bd936b72b0314cc89a90232f5
  sha256: 6cd68419f115d8cdf242d14e85792e90f9e16fb73bb39b3df75d4ef1ecbcd2b7

1.3 Task 2.3

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

1. Generate the hash value H1 for the file created in Section 2.1.

2. Flip the first bit of the input file (e.g. 01100001 $\rightarrow$ 11100001). You can achieve this modification using a binary editor such as ghex or Bless.

3. Generate the hash value H2 for the modified file.

4. How similar are H1 and H2?

1.3.1 Question 5

- **Question:** Describe your observations. Count how many bits are the same between H1 and H2 for MD5 and SHA256 (writing a short program to count the same bits might help you). In the report, specfiy how many bits are the same

- **Answer:**
  md5: 67 bits are the same
  sha256: 125 bits are the same

1.4 Task 2.4

In this task, we will investigate collision resistance of the hash function. The task is limited to weak collision resistance. We will use the brute-force method to see how long it takes to break this property. Instead of using openssl's command-line tools, you are required to write your own Java program. Get familiar with the provided sample code and feel free to use its parts in your solution. The code uses the MessageDigest class from the java.security library. You can find more information about this class at the official web documentation:
https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html.
Since most of the hash functions are quite strong against the brute-force attack on collision resistance, it could 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. In this task, work with the SHA256 one-way hash function, but use only the first 24 bits of the hash value. Namely, we are using a modified one-way hash function. Please design an experiment to answer the following questions:

1.4.1 Question 6

- **Question:** Investigate how many trials it will take to break the weak collision property using the brute- force method. Below is a list of five messages. For each message, report how many trials it took before you could find a message with the same hash. IV1013 security, Security is fun, Yes, indeed, Secure IV1013, No way
  Answer:
  IV1013 security: 28872840
  Security is fun: 10015372
  Yes, indeed: 9777773
  Secure IV1013: 23746118
  No way: 1139565