IV1013 One-Way Hash Report

Veronica Nadler

Spring Term 2024

1 Introduction

2 Tasks

2.1 Generating Message Digest and MAC

In the first task, we try out the 'openssl dgst' command line script which is used to generate *message digests*, or *hashes*. The message to encrypt is created as a text file containing the string "nadler@kth.se". The hashes are created with three different algorithms; MD5, SHA1 and SHA256.

```
openssl dgst -md5 email.txt
openssl dgst -sha1 email.txt
openssl dgst -sha256 email.txt
```

Q1: Describe your observations. What differences do you see between the algorithms?

The length of the hash increases with each algorithm. MD5 has the shortest hash with 32 characters, then comes SHA1 with 40 and the longest is SHA256 with 64. Upon analyzing the hashes some, it's clear that the values range from 1-9 and a-f which corresponds to hexadecimal. One hexadecimal value is 4 binary digits so from this one can deduce that $32 \cdot 4 = 128$, $40 \cdot 4 = 160$ and $64 \cdot 4 = 256$ which corresponds to the algorithms hash lengths of 128 bits, 160 bits and 256 bits respectively.

Q2: Write down the digests generated using the three algorithms.

The hashes created with the message "nadler@kth.se" with the different algorithms:

Algorithm	Hash
MD5	4228727bb4944216a29e5dd46629c766
SHA1	a35496217448bda3bfd17482d53295f3e3eb1738
SHA256	be0a4968754984b2c4b2c4458a676aaa696f078523b5465cd630e4885d9285ed

2.2 Keyed Hash and HMAC

This tasks concerns eyes hashes, or message authentication code (MAC). The keyed hash involves combining a message with a secret key to produce a fixed-size hash value. Now we are instead using the command

```
openssl dgst <hash-algo> -hmac <key> <file>
```

for the different algorithms and and trying for different lengths of key. I tried inputting the following for the different algorithms

```
openssl dgst -md5 -hmac "abcdefg" email.txt
openssl dgst -md5 -hmac "abcdefghijklmno" email.txt
```

Q3: Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why?

It is not a requirement to used a fixed key size, however the hash length will not vary regardless of key length. The user can input whatever key length they desire. If it's longer than the hash functions block size, it must be processed before it can be used which is done by hashing the key down to an appropriate length. If the key is shorter, it is padded with zeroes. The block size is the same for all three algorithms we use: 64 bytes, or 512 bits. Using a longer key can enhance security by increasing the difficulty of brute-force attacks against the key but also requires the system to do extra work affecting performance.

Q4: Use "IV1013-key" as the secret key.

```
openssl dgst -md5 -hmac "IV1013-key" email.txt
openssl dgst -sha1 -hmac "IV1013-key" email.txt
openssl dgst -sha256 -hmac "IV1013-key" email.txt
```

Algorithm	Hash
MD5	1b48bccc24d0727900c94099fee79709
SHA1	6e1fee47602f97f72cd5aa98ecca11b11b6823b1
SHA256	fd0ea8c886aee21ec738b9ca0e29c03df7498fbeff361dab7735efc59525ce74

Table 1: key="IV1013-key"

2.3 The Randomness of One-way Hash

In this task we are generating two hash values for two files, using MD5 and SHA256 and comparing them. The first file is the same file used in the previous tasks, "email" and the other is derived from this file by flipping its first bit and saving as a new file, "modified-email".

The first bit is flipped with the software Bless. The first hexadecimal value $6e = 01101110$ was translated to $ee = 11101110$.

To compare the bits, I wrote a program in Java to avoid doing this by hand

```java
private static String hexToBinary(String hexString) {
    BigInteger bigInteger = new BigInteger(hexString, 16);
    String binary = bigInteger.toString(2);

    int padding = (hexString.length()*4) - binary.length();
    if (padding > 0) {
        binary = "0".repeat(padding) + binary;
    }
    return binary;
}
private static int countBits(String h1, String h2){
    int commonBits = 0;
    for(int i=0; i<h1.length(); i++){
        if(h1.charAt(i)==h2.charAt(i)){
            commonBits++;
        }
    }

    return commonBits;
}
```

Q5: Describe your observations. Count how many bits are the same between H1 and H2 for MD5 and SHA256.

Generating the hash values H_1, H_2 give

Algorithm	common bits
MD5	66
SHA256	128

Table 2: Common bits for email.txt and modified-email.txt

Despite the only difference of the two files being 1 bit, the results after hashing are very different. Translating the hexadecimal hash value to binary reveals that only about half of the bits are common.