One Way Hash Function and MAC

2.1 Generating Message Digest and MAC

Question 1. Describe your observations. What differences do you see between the algorithms?

The differences between the three algorithms seem to be the bit lengths of them. MD5 has 128 bits, SHA1 160 bits and SHA256 has 256 bits.

Question 2. Write down the digest generated using the three algorithms.

Md5 – 87cf23d9547fd02f1f476b9c930c73b8
Sha1 – 13cad2b003cb1dc96bb016834f63a45f175a6d41
Sha256 – 54b802b6350f7f87cdcaaf34ffd81bc04ce63ca6375889068c877eac5e41ce0e

2.2 Keyed Hash and HMAC

Question 3. Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why?

When I try to use different key sizes the hash values still get generated each time. The size does not affect the result. However, if the key length is shorter than the message, padding will be used in order to work correctly.

Question 4. Now use the string IV1013-key as the secret key and write down they keyed hashes generated using the three algorithms.

HMAC-MD5 = d841d4d6c7651651c970316cce146bd4
HMAC-SHA1 = 60c1f842a59c93cabe6274f78243f3406b46c470
HMAC-SHA256 = 73de8139e6f95755af7ae5d78c1274085529a9642b5123177157786a694b7467

2.3 The Randomness of One-way Hash

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 → 11100001).
3. Generate the hash value H2 for the modified file.
4. How similar are H1 and H2?

H1 → Md5 – 87cf23d9547fd02f1f476b9c930c73b8
H2 (flipped) → Md5 – c3fd4230129d0c7bde76d30bc0556709

H1 → Sha256 – 54b802b6350f7f87cdcaaf34ffd81bc04ce63ca6375889068c877eac5e41ce0e
H2 (flipped) → Sha256 -
90df6ed66eddd191bb199a5c54c7326836ebf5dac74068db149afb59fec29a1f

Question 5. 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.

Writing a java program that counts the same bits from the two inputfiles (Original.txt and FlippedBit.txt) was helpful when doing this task. The result given from program was:

Using MD5: 73 similar bits
Using SHA-256: 142 similar bits

Discussion

In this lab, the focus has been on hash One way hash functions and Message Authentication Code (MAC). It was interesting to experiment with the different hash algorithms and see the hash values generated in the terminal directly. The first observation I did for the first task was that the hash values generated were different lengths. When testing the key sizes in HMAC, the sizes of the keys did not seem to matter. After some research, I found out that the recommended key size should be the same as the output size. However, it is not a good idea to have key sizes shorter than the output because it will affect the security strength, decreasing it. Keys longer than the output will not improve the security either that much. (https://tools.ietf.org/html/rfc2104)

For task 2.3 I had some difficulties figuring out how to flip the first bit according to the lab instructions. However, I found an online tool that easily did the job. The trickiest part for this task was to write the java program that counted the similar bits. I took the code from the first lab and modified it to work for this task, reading two input files with the bits from the terminal. There were some trouble reading in the files correctly, but I ended up saving them in a byte [] array for easy access, then comparing the files bit by bit in a loop.

Source code for counting similar bits:

```
// This program counts the same bits from two inputfiles

import java.io.*;

public class BitCounter {
    public static void main (String [] args ) {

        File inputOriginal = new File(args[0]);
        File inputFlippedBit = new File(args[1]);

        int length = (int) inputOriginal.length();

        try (
            InputStream inputStream1 = new FileInputStream(inputOriginal);
            InputStream inputStream2 = new FileInputStream(inputFlippedBit);

        ) {

          byte[] inputOr = new byte[length];
          byte[] inputFl = new byte[length];

           int count = 0;
           int bytesRead;

            bytesRead = inputStream2.read(inputFl);

            while ((bytesRead = inputStream1.read(inputOr)) != -1 ) {

                for (int i=0; i < length; i++) {
                        if (inputOr[i] == inputFl[i]) {
                            count++;

                        }
                        System.out.println(count);
                 }

            }
            inputStream1.close();
            inputStream2.close();
            System.exit(0);

        } catch (IOException e) {
            System.out.println("Error reading inputFile!");
            System.exit(1);
        }

    }
```