A study of One-way Hash Functions and Collision Resistance of 24-bit hashes

Jack Webb

April 2022

IV1013 Introduction to Computer Security

Contents

1 Introduction

One-way Hash Functions are designed in such a way that it should be very easy to compute a hash value x from a given message M with the function H such that $H(M) = x$. But from a given hash value x it should be difficult to find a message M such that $x = H(M)$. A common way these hash functions work is by dividing the given message into multiple blocks of the same length (the last block is padded to make it the same length if it is not initially) and applying a cryptographic hash function to each block.
SHA-256 uses the Merkle-Damgård constuction which uses the cryptographic compression function for each block chained together. This function takes two input strings (X, Y) and outputs a hash value 'd' of length 'n'. Here input X is the message block and Y is the hash value d of the previous block in the chain. Since the first block has no previous block an initialization vector v is given instead.
An issue with the Merkle-Damgård construction is that if an attacker finds a collision (a collision being two messages M1 and M2 where $H(M1) = H(M2)$ between two messages M1 and M2 then for any message message P we have $H(M1||P) = H(M2||P)$. Thus the attacker can form other arbitrary collisions.

2 Generating Message Digest and MAC

2.1 Question 1

What can be observed here is that each hash function has completely different hash value and length even though the input is the same.

2.2 Question 2

Hash Function	Hash value (hex)	length (bits)
md5	04c6902f33a308614a31db33e540fb71	128
sha1	b3831c7b35f88ceef700d43b78c9a121409c4e58	160
sha256	d52e8d52be98c8f554b33e3cbfebd962d599ef47d91a124dac11f434e45bcbe4	256

Table 1: Generated hashes from the message jwebb@kth.se

3 Keyed Hash and HMAC

3.1 Question 3

The key does not have to be of a certain size when using HMAC and this is due to how it is implemented:

$$HMAC(K, m) = H((K' \oplus opad) \quad || \quad H((K' \oplus opad) \quad || \quad m))$$

Where K is the key and $K' = H(K)$ if K is larger than the block size; K otherwise. Thus it does not matter what size the key is since if will be truncated if bigger than the block size or 0 padded if smaller [2].

3.2 Question 4

Hash Function	Hash value (hex)
md5	e46b548105852cb64ea8f74de711445f
sha1	6e05f9a8c8cb5e93354ffb782c6b3ff454a7c2ed
sha256	a82bf3d75be3a88a2f1279da3f9398bdbd7c3966f161580f75a7e40d4743b717

Table 2: Generated hashes with HMAC where K = IV1013-key, m = jwebb@kth.se

4 Randomness of One-way Hash (Question 5)

Two messages M and M' where M' had the first bit flipped compared to M, were hashed with both MD5 and SHA-256 generating two hashes H and H' to see how different the output of the hash functions would be with just a minor change and the results are shown in table 3.
For comparison a test of 10^6 pairs of unique (pseudo randomly generated string) messages were hashed and the bits were compared to see how many would be the same. The results show that a message with only one bit flipped does have a vastly more similar output than that of two unique messages as seen in table 4.
This shows that even though two different messages give vastly different outputs the balance of the hash functions are not perfect and thus are not distributed perfectly evenly which becomes evident since as the messages become more similar so does the output.

Hash Function	# same bits in H and H'
md5	67
sha256	139

Table 3: Comparison of hash values H and H' derived from message M with the first bit flipped

Hash Function	# same bits in H and H' average
md5	31
sha256	64

Table 4: Comparison of hash values H and H' derived from two unique messages M and M'

5 Collision Resistance (Question 6)

To get a "modified" SHA-256 hash value of 24 instead of 256 bits I created my own MyMessageDigest class with its own digest function which only outputs the first three bytes of the java.security.MessageDigest.digest() output, given a string as input.
This could then be used to test for collisions by first generating a 24 bit hash from a pre-defined input string such as "IV1013 security" and testing for collisions by generating pseudo random arrays of bytes as input until a matching 24 bit hash is found and counting how many tries it took.

Pseudocode:

```
FUNCTION testForCollision(inputString)
    generate bytes

    hash = digest(inputString)
    randomHash = digest(bytes)

    tries = 1
    WHILE randomHash != hash
        generate bytes
        randomHash = digest(bytes)
        tries++
    ENDWHILE

    return tries, bytes

ENDFUNCTION testForCollision
```

This was then used to test for collisions on the following five messages:

- IV1013 security
- Security is fun
- Yes, indeed
- Secure IV1013
- No way

The result of these tests are shown in table 5. Finding more than 1000 collisions per message was unfeasible and is the reason why that number was chosen even though the results might be statistically insignificant due to the small sample size and even that took several hours for all messages running concurrently on all cores. I was very surprised at first how many tries it took to

Message	Average # tries of 1000 runs
IV1013 security	21173451
Security is fun	$>$ 4294967 (overflow)
Yes, indeed	27590959
Secure IV1013	28789419
No way	18260166
TOTAL	20021792

Table 5: Average number of tries until collision was found for each message

find a collision of an output of just 24 bits since according to a table on Wikipedia about birthday attacks, an output of 32 bits should have a 75% possibility of a random collision after 110,000 tries which is several orders of magnitude lower than my results [1]. Since the output of the full length hash should be close to an even distribution taking only the first 3 bytes of the output should not affect the results this greatly which left me at a loss as to why I got this massively different result.

After some more thought I thought more about if I had misunderstood what I had read about birthday attacks. How can there be a 75% possibility of a collision after only 110,000 tries when a 24 bit hash has 2^{24} possible values. If I then generate 8 million unique hashes I have generated half of all possible hashes and thus should have around a 50% chance of collision. This then matches very closely with my results where it took, on average, 20 million tries. The higher amount of tries than possible hashes can be explained by the program generating a lot of duplicates, 20 million tries would not generate 20 million unique hash values.

References

[1] *Birthday attack*. URL: https://en.wikipedia.org/wiki/Birthday_attack.

[2] *HMAC*. URL: https://en.wikipedia.org/wiki/HMAC.