1 Hash Functions

by Lukas Wagenlehner, ID: 00089814

1.1 Introduction

This article shall explain hash functions and on which basic construction they have been developed. The padding method used for the algorithms is also explained. The article is mainly based on the book ”Cryptography Made Simple” by Nigel P. Smart [1].

1.2 Merkl-Diagård Construction

This chapter explains the Merkl-Diagård construction. All hash functions in this paper use this construction.
In the construction, the message X is first divided into blocks of equal size using a padding method. Then a compression function is applied. The function has internal variables which are assigned constant values at the start. Now every single block is given to the function one at a time. After a block has been processed, the internal variables have changed and are sent to the output. In the next iteration, the internal variable values are no longer assigned constants, but are assigned with the values previously sent to the output. When the last block is processed, the internal variables are collected and give the hash value $H(X)$.

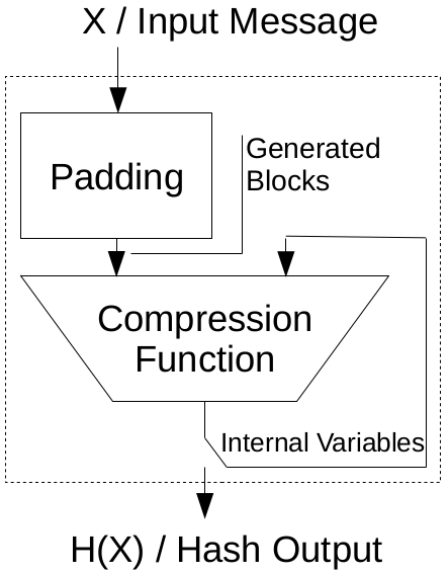

Figure 1: Diagram of the Merkl-Diagård construction [3, min 25].

1.3 MD-4 Family

The following listed functions can be distinguished by output length of the hash value. Furthermore they are different in how many rounds and steps are executed in the compression function. These are not all differences. Which operations are processed internally is not always the same, mostly they are combinations of xor, and, or, roundshift and shift. Some of the listed functions like MD-4 and SHA-1 are not safe anymore. This list should give an overview of some existing hash algorithms, but only SHA-1 is explained in more detail in this paper.

Algorithm	output size	rounds	steps
MD-4	128	3	16
MD-5	128	4	16
SHA-1	**160**	**4**	**20**
RIPEMD-160	160	5	16
SHA-256	256	64	1
SHA-512	512	80	1

1.4 Padding

The padding algorithm shown here generates a block size of 512 bit, which is a prerequisite for the algorithm presented here. First, if the message is not already stored in a bit array, it has to be translated, e.g. using the ASCII code. The length of the bit array is determined and stored. Now a logical bit with the value 1 is appended to the end of the data. As long as possible always 512 bits are collected and stored as 1 block. If this is no longer possible, the last block is filled with up to 448 bits. If the last block cannot be filled with 448 data bits, it is filled with logical zeros. In case the 448 bits are not enough for the last data bits, the block is filled up to 512 bits , if nesscasary with zeros. Than a new block with 448 zeros is created. Finally, the initially stored size is converted to binary and appended to the last block as a 64 bit number. Now the message is divided into 512 bit blocks and has its length binary decoded at the end [4, min 8-11].

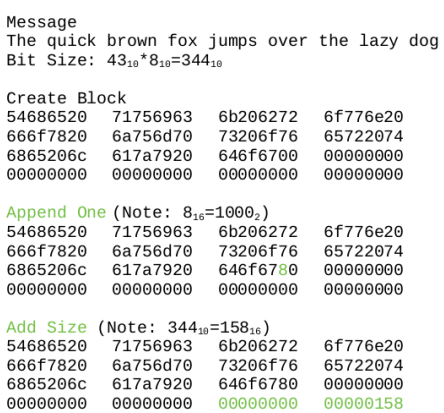

Figure 2: Example padding.

1.5 SHA-1

First of all, SHA stands for Secure Hash Algorithm. All variables in the SHA-1 algorithm have the size of 32 bit to allow a fast implementation on 32 bit processors [2, page 1]. The calculated hash value has a length of 160 bit for any input length. The following is a step-by-step description of what is necessary for a real code implementation of SHA-1. In the figures 2-4 you can see an example of the algorithm executing.

1. Define some variables and funktions.
   First define 5 internal state variables. These result in the hash value at the end.

   $$H = \{H_1, H_2, H_3, H_4, H_5\}$$

   At the very beginning, H is always initialized as follows.

   $$H_1 = \texttt{0x67452301}$$
   $$H_2 = \texttt{0xEFCDAB89}$$
   $$H_3 = \texttt{0x98BADCFE}$$
   $$H_4 = \texttt{0x10325476}$$
   $$H_5 = \texttt{0xC3D2E1F0}$$

   Next define 4 round constants. Which are necessary for extra securety.

   $$y_1 = \texttt{0x5A827999}$$
   $$y_2 = \texttt{0x6ED9EBA1}$$
   $$y_3 = \texttt{0x8F1BBCDC}$$
   $$y_4 = \texttt{0xCA62C1D6}$$

   Last define 3 bitwise functions on three 32-bit variables.

   $$f(u, v, w) = (u \wedge v) \vee ((\neg u) \wedge w)$$
   $$g(u, v, w) = (u \wedge v) \vee (u \wedge w) \vee (u \wedge w)$$
   $$h(u, v, w) = u \oplus v \oplus w$$

2. Perform the padding procedure described in figure 2.

3. Take the 1st/next 512 bit block and put it in $X_j \; with \; 0 \leq j < 16$ where each X_j has the size 32 bit.

4. Load H, for the first run use the inital values defined above. Else use the previously calculated H values. Note: $s_r - 1$ means that you need to use the values of the previous round.

   $$(A, B, C, D, E) \leftarrow s_r - 1 = (H_1, H_2, H_3, H_4, H_5)$$

5. Expand X so that at the end applies $X_j \; with \; 0 \leq j < 80$.

   $$for \; j = 16 \; to \; 79 \; do$$
   $$X_j \leftarrow ((X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) \lll 1)$$

6. Perform Round 1 to 4. Each round has 20 steps.
   Round 1:

$$
\begin{aligned}
&for\ j = 0\ to\ 19\ do\\
&\quad t \leftarrow (A \lll 5) + f(B, C, D) + E + X_j + y_1)\\
&\quad (A, B, C, D, E) \leftarrow (t, A, B \lll 30, C, D)
\end{aligned}
$$

   Round 2:

$$
\begin{aligned}
&for\ j = 20\ to\ 39\ do\\
&\quad t \leftarrow (A \lll 5) + h(B, C, D) + E + X_j + y_2)\\
&\quad (A, B, C, D, E) \leftarrow (t, A, B \lll 30, C, D)
\end{aligned}
$$

   Round 3:

$$
\begin{aligned}
&for\ j = 40\ to\ 59\ do\\
&\quad t \leftarrow (A \lll 5) + g(B, C, D) + E + X_j + y_3)\\
&\quad (A, B, C, D, E) \leftarrow (t, A, B \lll 30, C, D)
\end{aligned}
$$

   Round 4:

$$
\begin{aligned}
&for\ j = 60\ to\ 79\ do\\
&\quad t \leftarrow (A \lll 5) + h(B, C, D) + E + X_j + y_4)\\
&\quad (A, B, C, D, E) \leftarrow (t, A, B \lll 30, C, D)
\end{aligned}
$$

7. Store A, B, C, D, E in H.

$$(H_1, H_2, H_3, H_4, H_5) \leftarrow s_r = (H_1 + A, H_2 + B, H_3 + C, H_4 + D, H_5 + E)$$

8. If there is one more block, continue with step 3. Otherwise append all H values of the sequence one after the other and get the final hash value.

```
Block after step 2:
54686520  71756963  6b206272  6f776e20
666f7820  6a756d70  73206f76  65722074
6865206c  617a7920  646f6780  00000000
00000000  00000000  00000000  00000158

Step 3: Assign X
54686520  71756963  6b206272  6f776e20  666f7820
6a756d70  73206f76  65722074  6865206c  617a7920
646f6780  00000000  00000000  00000000  00000000
00000158  00000000  00000000  00000000  00000000
...

Step 4: Load H and A-E
H1 H2 H3 H4 H5
67452301  efcdab89  98badcfe  10325476  c3d2e1f0
A B C D E
67452301  efcdab89  98badcfe  10325476  c3d2e1f0
```

Figure 3: Assign X, initialize H and $A - E$.

```
Step 5: Expand X
54686520  71756963  6b206272  6f776e20  666f7820
6a756d70  73206f76  65722074  6865206c  617a7920
646f6780  00000000  00000000  00000000  00000000
00000158  ae5a4e7c  fef0fcc6  d240f914  56b09a59
...

Step 6: (Note: A-E change every round)
Round 1
A B C D E
7f67b89a  b423cc8e  9da5be04  feb7d73b  80f4d745
Round 2
A B C D E
7d92efdf  182630f2  9bfa5df7  58012101  14536b71
Round 3
A B C D E
f9533bfe  0075e58a  d749ca0d  752d290d  75f11cb0
Round 4
A B C D E
c88fbec5  8a5f7d73  54c9c1e3  ab4492c3  57c10922

Step 7: Store A-E in H
H1 H2 H3 H4 H5
2fd4e1c6  7a2d28fc  ed849ee1  bb76e739  1b93eb12

Step 8: Hash
2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
```

Figure 4: Expand X, process rounds and show hash value.

1.6 Conclusion

I find the topic interesting, because it is present everywhere today, e.g. password storing and document signing. In order to create the above examples I made my own implementation of the algorithm. I learned a lot during the implementation. Therefore I think it would be cool to have a practical part in the class besides presentation and paper writing. I mainly decided for a more detailed introduction of SHA-1, because the explanation in the book of MD-4 didn't make sense to me first. Online I simply found better results when searching for SHA-1. The code from my SHA-1 implementation can be downloaded on my github account [LukasW352435/hash].

References

[1] *Cryptography Made Simple*. Nigel P. Smart, 2016.

[2] Ross Anderson1 and Eli Biham. Tiger: A fast new hash function, n.d. [Online; accessed 2-July-2019]. URL: https://pdfs.semanticscholar.org/ef53/86db757b342fc94a62ed3bc9608ec302e8f6.pdf.

[3] Kiran Kuppa. Sha-1 hash function, 2013. [Online; accessed 2-July-2019]. URL: https://www.youtube.com/watch?v=5q8q4PhN0cw.

[4] Sam Wheeler. How does sha-1 work - intro to cryptographic hash functions and sha-1, 2017. [Online; accessed 2-July-2019]. URL: https://www.youtube.com/watch?v=kmHojGMUn0Q&list=RDQMlqNibgkjFIU&index=4.