On The Security of Hash Functions

Myo Myo Aung
Department of Engineering Physics, Mandalay Technological University, Mandalay 05052
Email: mmaung.mm@gmail.com

Abstract— Cryptographic hash functions are an important building block for a wide range of applications such as the authentication of information and digital signatures. In this paper we give special emphasis on the design and security of the standard hash functions.

Key word- hash functions, design and security

I. Introduction

Hash function is a fundamental tool in Information Security. In its simplest from a hash function is an algorithm that takes an input of any size and outputs a fixed length "hash code" that is, in some sense, difficult to predict in advance. The basic idea is that, the hash code serves as compact representative image of an input string and can be used as if it is uniquely identifiable with that string. That is, the output of the hash function serves as a digital finger-print for the input and should be the same each time the same message is hashed. We use hash functions to help provide data integrity in Message Authentication Codes (MACs), to produce message digests for use with digital signature schemes and to produce Manipulation Detection Codes (MDCs) in entity authentication and key establishment schemes.

For a hash function to be secured it is required to be one-way and collision resistant. The one-way property can be achieved if it is easy to generate the message digest of a message but, is hard to determine the original message when the digest of it is known. On the other hand, collision resistance can be attained if it is hard to find two different messages, having same message digest as output. Apart from these requirements, the hash function should be accepting a message of any size as input and computation of the message digest must be fast and efficient.

Hash functions have been fairly widely standardized by International Electrotechnical Commission (ISO/IEC), National Institute of Standards and Technology (NIST), and the Internet Engineering Task Force (IETF). In this paper, we give special emphasis on the design and security of Message Digest (MD) family which has standard by IETF and Secure Hash Algorithm (SHA) family of NIST secure hash standard.

The paper is organized as follows. Section 2 describes the design and security of standard hash functions. Section 3 deals with their software performance. Section 4 presents the conclusions.

II. Design and Security of Standard Hash Functions

The most commonly used hash functions are MD5 and SHA-1. These are dedicated or custom-designed hash functions, this is, algorithms that were especially designed for hashing operations. Other examples of custom-designed hash algorithms are MD2, MD4, and MD5 (the MDx-family), SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 (the SHA-family), RIPEMD-160, HAVAL and N-hash.

Dedicated hash algorithms are designed to be very efficient on 32-bit machines, which make them very popular; even through their security is only based on heuristic arguments. None of the desired properties of cryptographic hash functions can actually be proven for them. However recent advances in cryptanalysis have shown that this is not good enough. In fact, all of the hash functions mentioned above apart from the SHA-2 algorithm (i.e. SHA-224, SHA-256, SHA-384, and SHA-512) are currently considered broken. Although not all of the theoretical attacks are practical yet, they are rapidly being improved and put into practice. Trust in dedicated hashing has long been undermined, leaving hardly any cryptographic hash functions that can still be used without concern.

The two most widely used dedicated or custom-designed hash algorithms MD5 SHA-1, how their security is argued and how they have been attacked. This motivates the research of hash functions with provable security, which will be investigated in the following.

A. MD5

This Message Digest Algorithm 5, known as MD5, was designed by Ronald Rivest of MIT in 1991 and is specified in the MD5. It takes a message of arbitrary length as input and produces a 128-bit message digest.

MD5 is used widely in the software world to compute cryptographic checksums and to store passwords. It is part of applications such as GPG (public key encryption), Kerberos (network authentication), TLS (secure client-server connections), SSL (client-server authentication),

Cisco type 5 enable passwords (password storage system) and RADIUS (remote user authentificatiion).

The MD5 Algorithm. Let M be the input message of length *b* bits. *M* is first padded to a multiple of 512 bits and then divided into 512-bit blocks $M_0, ..., M_{n-1}$ each consisting of 16 words. Each block is then processed in 4 rounds; each consisting of 16 operations, using a 4-word buffer denoted *A, B, C, D*. After all blocks have been processed, the buffer contains the message digest. More specifically, the steps in MD5 are:

Padding. A single bit "1" is appended at the end of the message. The "0" bits are appended until the length of the new message is congruent to 448 modulo 512. Finally a 64-bit representation of *b* (the length of the original message) is appended. The resulting message is an exact multiple of 512 bits long.

Initialize buffer. The buffer is initialized to the hex values

$A = 01234567,\ B = 89ABCDEF,$

$C = FEDCBA98,\ D = 76543210$

(with the least significant bit listed first).

Compute constants. A 64-element table is computed from the sine function according to the formula $K_t = \lfloor 2^{32} |\sin(t+1)| \rfloor$ for $t = 0,...,63$, where *t* is in radians.

Auxiliary functions. Four auxiliary functions, which each take as input three 32-bit words and produce as output one 32-bit word, are defined as

$f_t(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)\ \ for\ t = 0,...,\ 15,$

$f_t(B,C,D) = (B \wedge D) \vee (C \vee \neg D)\ \ for\ t = 16,...,\ 31$

$f_t(B,C,D) = B \oplus C \oplus D\ \ for\ t = 32,...,\ 47,$

$f_t(B,C,D) = C \oplus (B \vee \neg D)\ \ for\ t = 48,...,\ 63$

Word order. Defined the following vector that determines in which order the words of a block will be processed in each round:

$Round 1 : (j_0,...,j_{15}) = (0,1,...,15)$

$Round\ 2 : (j_{16},...,j_{31})$

$= (1,6,11,0,5,10,15,4,9,14,3,8,13,2,7,12)$

$Round\ 3 : (j_{32},...,j_{47})$

$= (5,8,11,14,1,4,7,10,13,0,3,6,9,12,15,2)$

$Round\ 4 : (j_{48},...,j_{63})$

$= (0,7,14,5,12,3,10,1,8,15,6,13,4,11,2,9)$

Shift amounts. Define the following shift amounts

$Round\ 1 : (s_0,...,\ s_{15})$

= (7, 12, 17, 22,7, 12, 17, 22,7, 12, 17, 22,7, 12, 17, 22)

$Round\ 2 : (s_{16},...,\ s_{31})$

= (5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20)

$Round\ 3 : (s_{32},...,s_{47})$

= (4, 11, ,16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23)

$Round\ 4 : (s_{48},...,s_{63})$

= (6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21)

Process message in 16-word blocks.

/* Process each 16-word block.*/

$for\ i = 0,\ ....,\ n-1\ \ do$

(a) Divide M_i into words $W_0, ..., W_{15}$ where W_0 is the left-most word.

(b) Save A as $\overline{A}$, B as $\overline{B}$, C as $\overline{C}$ and D as $\overline{D}$: $\overline{A} = A,\ \overline{B} = B,\ \overline{C} = C,\ \overline{D} = D$

(c) for t = 0, …, 63 do

$X = B + (f_t(B,C,D)W_{jt} + K_t <<< S_t)$,

$A = D, D = C, C = B, B = X$

(d) Then increment each of the four registers by the value it had before this block was started:

$A = \overline{A} + A,\ \ B = \overline{B} + B,$,

$C = \overline{C} + C,\ \ D = \overline{D} + D$

end /*of loop on *i** /

Output. The message digest is *A, B, C, D.*

One MD5 operation at step can be described in the following diagram:

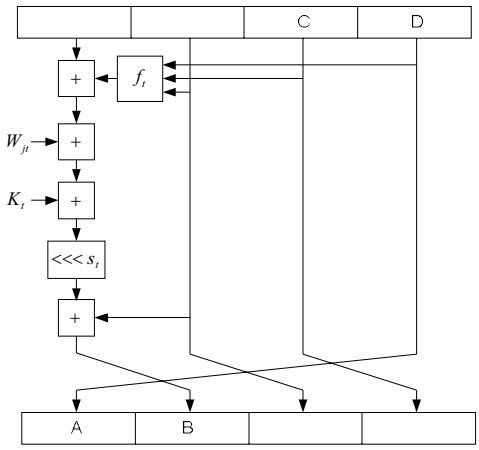

Fig 1: Step operation for MD5

B. Security of MD5

MD5 is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest, that is , to be collision resistant and preimage resistant. In addition, MD5 was made to be fast on 32-bit machines and to operate without large substitution tables; hence it can be coded compactly.

While the second and third attributes can be easily verified and are definitely true, the security of MD5 is based on a number of heuristic arguments and no proofs of security exist. Heuristic arguments include that the auxiliary functions are non-invertible, non-linear and asymmetric, if bit in B, C and D are independent and unbiased, then each bit of $f(B,C,D)$ will be independent and unbiased, each step adds in the result of the previous step, each step has a unique additive constant, input words are accessed in a different order in each round, and shift amounts in different rounds are distinct.

All of these attributes are said to increase the avalanche effect, meaning that if an input is changed slightly (for example, changing a single input bit), then the output changes significantly (for example, half the output bit flip).

While its speed and the fact that the algorithm is fairly simple and publicly available have made MD5 very popular, it seems rather alarming that the algorithm is used in many cryptographic applications to this day, considering its security is not supported by any proof at all.

C. Attacks on MD5

MD5 was designed in 1991 to replace an earlier hash function MD4 in which flaws had been found. However, it was soon discovered that MD5 also has its problems. Starting in 1993 the use of MD5 was more and more questioned by several successful collision attacks, and recent results have completely destroyed confidence in the algorithm.

In 1993, Boer and Bosselaers were able to find a so-called pseudo-collision for the compression function of MD5, that is, two different initialization vectors that produce a collision when the MD5 compression function is applied to the same message. Although this is an attack that has no practical significance, it exposed the first weakness in MD5.

Dobbertin announced a collision of the MD5 compression function in 1996. While this was not an attack on the full version of MD5, it worried cryptographers enough to recommend switching to a replacement, such as SHA-1, WHIRLPOOL, or RIPEMD-160.

Also, a hash of 128 bits is small enough to allow birthday attacks. Cooke and his company launched a distributed search project in 2004 with the aim of finding collisions for MD5 by a brute force search using Pollard's rho method. The project was a abandoned a few months later, when it was announced that collisions had actually been found by analytical for many, and it is said that Wang and her team received a standing ovation when they reported that they had found collisions for the full MD5 at the CRYPTO conference in August 2004.

D. SHA-1

SHA-1 is the most commonly used member of the SHA family. It was published by the National Security Agency (NSA) in 1995 as a US government standard and to replace the SHA-0 algorithm from 1993, in which a flaw had been found. SHA-1 takes an input message of at most 2^{64}-1 bits and produces a message digest of length 160 bits.

Since MD5 become untrustworthy, SHA-1 has become the most commonly used hash function. It is employed in security applications and protocols such as Open PGP (encryption of data), S/MIME (public key encryption and signing of e-mail), IPSec (encryption and / or authentification of IP packets) and SHH (secure remote login). The copy prevention of Microsoft's Xbox game console also relies on SHA-1.

The SHA-1 Algorithm. SHA-1 is often considered a successor of MD5 because its design is very similar. Padding is performed in the same way, then a message M of length b bit is split into 16-word blocks $M_0, ..., M_{n-1}$ and each block is processed in 4 rounds, consisting of 20 operations each, and using a 5-word buffer A, B, C, D, E. After all blocks have been processed, the buffer contains the message digest. More specifically, the steps in SHA-1 are:

Padding. M is considered as a bit string and a single bit "1" is a appended at the end of the message. Then "0" bits are appended until the length of the new message is congruent to 448 modulo 512. Finally a 64-bit representation of b is appended, resulting in a message which is an exact multiple of 512 bits long.

Initialize buffer. The buffer is initialized to the values

$A = 67452301,\ B = EFCDAB89,$

$C = 98BADCFE,\ D = 10325476,\ E = C3D2E1F0$

Constants. The following constants are used (in hex):

$K_t = 5A827999\ \ for\ t = 0, ..., 19$

$K_t = 6ED9EBA1\ \ for\ t = 20, ,..., 39$

$K_t = 8F1BBCD\ \ for\ t = 40, ..., 59$

$K_t = CA62C1D6\ \ for\ t = 60, ..., 79$

Auxiliary functions. A sequence of logical functions is used, each operating on three words and producing one word as output. They are defined as follows:

$f(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)\ \ for\ \ t = 0, ..., 19,$

$f(B,C,D) = B \oplus C \oplus D\ \ for\ \ t = 20, ..., 39$

$f_t(B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$

$for\ \ t = 40,...,59$

$f(B,C,D) = BCD for\ t = 60,...,79$

Process message in block.

/* Process each 16-word block.*/

$for\ i = 0, ..., n-1\ \ do$

(a) Divide M_i into words $W_0, ..., W_{15}$ where W_0 is the left-most word

(b) For $t = 16, ..., 79$

let $W = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<<1$

(c) Save A as $\overline{A}$, B as $\overline{B}$, C as $\overline{C}$, D as $\overline{D}$ and E as $\overline{E}$: $\overline{A} = A,\ \overline{B} = B,\ \overline{C} = C,\ \overline{D} = D,\ \overline{E} = E$

(d) for t = 0, …, 79 do

(e) $X = (A <<< 5) + f(B,C,D) + E + W + K$

$E = D,\ D = C,\ B <<< 30,\ B = A,\ A = X$

(f) Then increment each of the four registers by the value it had before this block was started:

$A = \overline{A} + A,\ B = \overline{B} + B,\ C = \overline{C} + C,$

$D = \overline{D} + D,\ E = \overline{E} + E$

end /* of loop on i *

Output. The message digest is *A, B, C, D, E.*

One SHA-1 operation at step can be described by the following diagram:

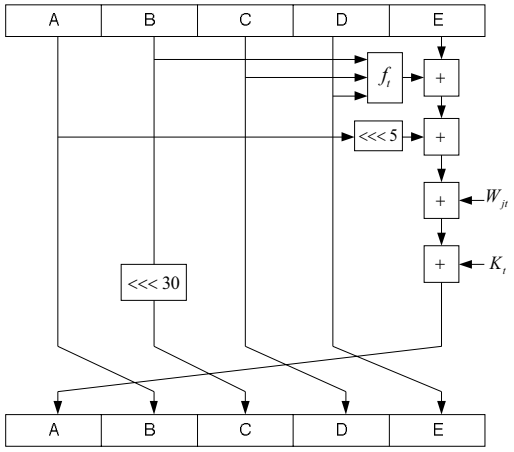

Fig. 2: Step operation for SHA-1

1) Security of SHA-1

The authors of the SHA-1 claim that it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different message which produce the same message digest for SHA-1. As with MD5, however, no proofs of security exist and all there is to support this statement are heuristic arguments like those mentioned for MD5 in Section 2.1.1.

SHA-1 is also very fast on 32-bit machines and can be coded quite compactly, and it is thus used very widely. In fact, as flaws were found in MD5, cryptographers recommended replacing MD5 by SHA-1, which was done in many applications. Since SHA-1 has been broken as well (meaning that collisions can be produced with less computational complexity than that of a brute force attack). NIST now plans to replace SHA-1 by members of the SHA-2 family (SHA-224, SHA-256, SHA-384, SHA-512 named after their digest lengths), for which no attacks have been reported, by 2010.

2) Attacks on SHA-1

The members of the SHA-family were designed as successors of MD4, just as MD5 was, but they lasted a bit longer. SHA-1 is very similar to its predecessor SHA-0, and so the first reason to doubt the security of SHA-1 was the announcement that SHA-0 had been broken by Chabaud and Joux at CRYPTO'98. The next milestone in the cryptanalysis of SHA-0 was when Wang and her team announced their collision attack in 2004, which also works for SHA-0. That was when cryptographers first started to recommend finding alternatives to SHA-1, especially in the design of new cryptosystems. Also as a result of that, NIST announced it would phase out the use of SHA-1 by 2010 and replace it by SHA-2 variants.

The first successful attack on SHA-1 itself was performed by Rijmen and Oswald in early 2005. They were able to break a reduced version of SHA-1: 53 out of 80 rounds. Only a month later a break of the full version of SHA-1 was announced by Wang, Yin and Yu. This was another famous day for Wang and her team, who based their attack on several different methods used in earlier attacks on SHA-0 and MD5. This attack required 2^{69} operations, but was soon improved to take only 2^{63}. Such collision attacks generally work by starting off with two messages and continually modifying them throughout the attack. That means that the structure of the colliding messages is determined by the attack, and they will almost certainly turn out to be complete gibberish. Although this is of theoretical importance, it is hard to turn it into a practical attack.

Recheberger and de Canniere announced the first collision attack on SHA-1 where the attacker can influence the colliding messages. According to Rechebger, the new attack allows up to 25% of the amount to be freely selected can be further increased by optimizing the attack. This is now a quite practical attack itself, considering that HTML documents, for example, may have complete nonsense after the tag that will never be printed. So it is now possible to produce two seemingly identical html documents with the same SHA-1 hash. This leaves SHA-1 no better off than MD5.

Just as with MD5, (second) preimage attacks on SHA-1 have not been accomplished, but the collision attacks have reached a level that causes serious concern and makes urgent a quick replacement of the algorithm.

III. Performance of Hash Functions

In order to compare the performance of software implementations of hash functions, an overview has been compiled in Table 1. All timings were performed on a 90MHz Pentium processor. The implementations were written by A. Bosselaers [3]. Most of them use additional memory to improve the speed. The C-code was compiled with a 32-bit compiler in protected mode. Some algorithms like Snefru and SHA would perform relatively better on a RISC processor, where the complete internal state can be stored in the registers. On this type of processor, SHA is only about 15% slower than MD5.

TABLE 1: PROCESSING SPEED (IN MBIT/S) FOR A 90MHZ PENTIUM PROCESSOR, FOR BOTH ASSEMBLY IMPLEMENTATION AND PORTABLE C IMPLEMENTATION.

algorithm	MD4	MD5	SHA-1	RIPEMD	RIPEMD-128	RIPEMD-160
C		82	21	44		16
Assember	191	137		56	78	46

IV. Conclusion

In this paper we discuss about the security features which are essential to construct a cryptographic hash function. We also present the design of cryptographic hash functions which are both secure and efficient in software and hardware implementation. An important note is that the increased security of hash functions is achieved at the expense of lower performance.

References

[1] B.Preneel, Analysis and Design of Cryptographic hash functions.
Katholieke University Leuven, Beligum,
[2] Alexander W.Dent and Chris J.Mitchell.
User's Guide to cryptography and standards.
www.artchouse.com.
[3] S.Bakhtiari,R.Safavi-Naini,J.Pieprzyk, Cryptographic Hash Functions:
A Survey Paper, University of Wollongong, Australia.
[4] Maile Massierer, Provably Secure Cryptographic Hash Functions, University of New South Wales.
[5] Murali Krishna Reddy Danda, Design and Analysis of Hash functions.

