

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/283763680>

Cryptographic hash function: An elevated view

Article · July 2010

CITATIONS

14

READS

308

2 authors, including:



[Krishna Asawa](#)

Jaypee Institute of Information Technology

56 PUBLICATIONS 328 CITATIONS

SEE PROFILE

Cryptographic Hash Function: An Elevated View

Harshvardhan Tiwari

Computer Science and Engineering Department, IIIT University, A-10, Sector-62, Noida, India

E-mail: tiwari.harshvardhan@gmail.com

Tel: +91- 0120-2400973; Fax: +91-0120- 2400986

Krishna Asawa

Computer Science and Engineering/I.T. Department, IIIT University, A-10, Sector-62, Noida, India

E-mail: krishna.asawa@iiit.ac.in

Tel: +91- 0120-2400973; Fax: +91-0120- 2400986

Abstract

Information is an important commodity in the world of Electronic communication. To achieve a secure communication between communicating parties, the protection of authenticity and integrity of information is necessary. Cryptographic hash functions play a central role in cryptology. A cryptographic hash function takes an input of arbitrary large size and returns a small fixed size hash value. It satisfies three major cryptographic properties: preimage resistance, second preimage resistance and collision resistance. Due to its cryptographic properties hash function has become an important cryptographic tool which is used to protect information authenticity and integrity. This paper presents a review of cryptographic hash functions. The paper includes numerous definitions of hash functions, different types of hash functions such as block cipher based hash function and dedicated hash function, and various applications of hash functions. It gives special emphasis on dedicated hash functions like MD5, SHA-1 and RIPEMD-160.

Keywords: Cryptographic hash function, Authentication, Data integrity, MD5, SHA-1

1. Introduction

Hash functions were introduced in cryptography to provide message integrity and authentication. A function that compresses an input of arbitrary large length into a fixed small size hash code is known as hash function. The input to a hash function is called as a message or plain text and output is often referred to as message digest, the hash value, hash code, hash result or simply hash. In [3] hash function is defined as: A hash function H is a transformation that takes an input m and returns a fixed size string, which is called the hash value h (that is, $h = H(m)$). The required cryptographic properties of hash function are application dependent, but most important in practice are one-way ness and collision resistance. The first property implies that it should be computationally infeasible to find any input which hashes to a pre-specified output (pre-image resistance), or find any second input which hash the same output as a specified input (second preimage resistance). We recall the other desired characteristics of hash functions that are: it is computationally easy to extract message digest from the message and hard to find the message from the digest. The hash value of an input string is an imprint or digital fingerprint of that input string because it is unique and compact to input string. No two input strings can have the same digest value. A small modification in a message or document results in a

complete different digital fingerprint. Because of their important cryptographic properties hash functions are used in several applications like digital signature scheme, software integrity, e-cash, password protection, pseudo random string generation and in various communication protocols to protect authenticity and integrity of information [1, 7, 9]. In 1976 Diffie and Hellman [14] without specifying hash function directly, they stressed on the need for it as a building block of a digital signature scheme. By digital signature the authentication, integrity and preventing denial of original message can be practiced. It works in following steps: the sender creates a fixed length digital message digest from message and encrypts it with his or her own private key to form digital signature; The digital signature is then appended to the message and sent to the recipient party with the message: While the recipient party calculates the message digest HV1 of original message received from sender by using same hash function and then decrypts the signature(encrypted message digest from sender) by the sender's public key and gains HV2. If HV1 is same as HV2, the recipient knows that the message came from a legitimate party. Initially the message is hashed, and then the hash value as a representative of message, is signed in place of the original message. In this way time and space are saved compared with the case of signing the entire message. The problem of preserving the integrity of a potentially large message is thus reduced to that of a small fixed size hash value [5, 6].

2. Basic Definitions, Properties, Classification and Requirements of Hash Functions

Hash functions have been used in vast variety of cryptographic application and must provide different security properties depending on the security requirements of the application. The well known basic security properties of hash functions are preimage resistance, second preimage resistance and collision resistance. They are explained below:

- Preimage resistance: for any given code h , it is computationally infeasible to find x such that $H(x) = h$.
- Second preimage resistance: for any given input x , it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
- Collision resistance: it is computationally infeasible to find any pair (x, y) such that $H(y) = H(x)$.

Properties preimage resistance, second preimage resistance and collision resistance are also known as one-way, weak collision resistance and strong collision resistance respectively. Table 1 summarizes the level of effort required producing a birthday or square root attack for different types of hash functions, assuming n -bit result [4].

Cryptographic hash function can be traditionally classified as unkeyed hash functions and keyed hash functions. Unkeyed hash functions, also known as modification detection codes (MDCs), use message as a single input whereas keyed hash functions, also known as message authentication codes (MACs), can be viewed as hash functions which take two functionally distinct inputs, a message and a secret key. Unkeyed hash function is further classified into one-way hash function (OWHF), collision resistant hash function (CRHF), universal one way hash function (UOWHF) [2, 8, 11, 13]. The construction of CRHF is hard than OWHF. CRHF usually deals with longer length hash values.

Table 1: Strength of different hash functions

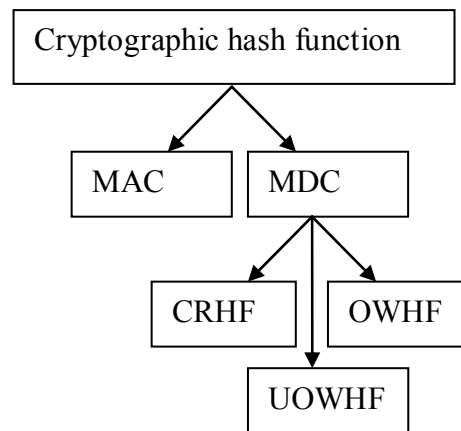
Type of hash function	Strength of hash function
One-way	2^n
Weak collision resistance	2^n
Strong collision resistance	$2^{n/2}$

2.1. Unkeyed Hash Function

An unkeyed hash function is a function $h: \{0,1\}^* \rightarrow \{0,1\}^n$, for a fixed positive integer n which has, as a minimum, the following two properties:

- Compression: h maps an input x of arbitrary finite bit length, to an output $h(x)$ of fixed bit length n .
- Ease of computation: given h and an input x , $h(x)$ is easy to compute.

Figure 1: Classification of cryptographic hash function



2.1.1. One-way Hash Function (OWHF)

One-way hash function is a hash function with properties preimage resistance and second preimage resistance. For these, finding an input which hashes to a prespecified hash value is difficult.

2.1.2. Collision Resistant Hash Function (CRHF)

A collision resistant hash function is a hash function with properties second preimage resistance and collision resistance. For these, finding any two inputs having the same hash value is difficult.

2.1.3. Universal One-way Hash Function (UOWHF)

In a universal one-way hash function, for randomly chosen input x , key k and the function h_k , it is hard to find $y \neq x$ such that $h_k(x) = h_k(y)$

2.2. Keyed Hash Function (MAC)

A keyed hash function is a function $h_k: \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^n$ for fixed positive integer n and k , if it satisfies following two properties:

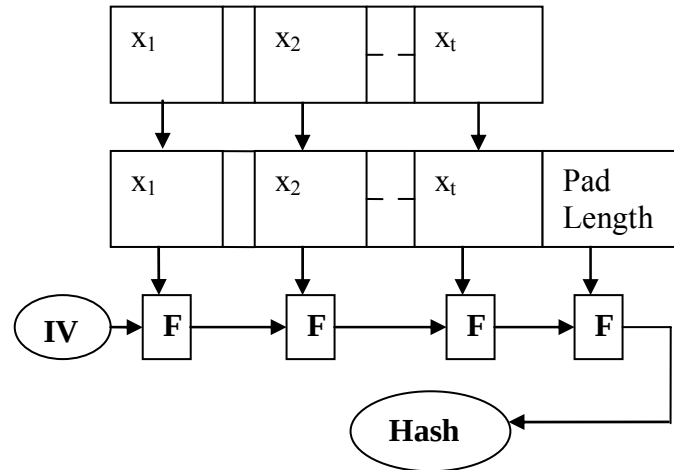
- Compression: h_k maps an input x of arbitrary finite bit length, to an output $h_k(x)$ of fixed bit length n
- Ease of computation: for a known function h_k , given a value k and an input x , $h_k(x)$ is easy to compute. The result is called MAC value.
- Computation-resistance: given zero or more text-MAC pairs $(x_i, h_k(x_i))$, it is computationally infeasible to compute any text-MAC pairs $(x, h_k(x))$ for any new input $x \neq x_i$.

Almost all hash functions are iterative processes which hash inputs of arbitrary length by processing successive fixed size blocks of the input. The input X of arbitrary finite length is divided into fixed length t -bit blocks, x_1 through x_t . This number of fixed length blocks must be multiple of the block length for attaining the overall bit length, it typically involves appending extra bits (padding). Hash function can be described as the following: $H_0 = IV$, $H_i = F(x_i, H_{i-1})$, $i=1, 2, \dots, t$; $h(x)=H_t$ where

IV is stood for initial value and the result of hash function F is called the hash round function. Such a recursive construction known as Merkle-Damgård hash construction designed by Ralph Merkle and Ivan Damgård independently in 1989[10,46]

Apart from the classification of keyed and unkeyed hash functions, they can be classified into other ways such as hash function based on block cipher, hash function based on modular arithmetic and dedicated hash functions. We are giving a brief review of these hash functions.

Figure 2: Merkle-Damgård Construction



2.3. Hash Function Based on Modular Arithmetic

Number theory problems are used to design these hash functions. Security of such hash function is directly proportional to the hardness of these problems. The two most important cryptosystems, based on modular arithmetic are RSA public key cryptosystem and ElGamal cryptosystem. Hash functions that are based on modular arithmetic can have variable digest length, depending on the size of modulus. Example of this approach is MASH-1 (Modular Arithmetic Secure Hash algorithm-1). The purpose of employing modular arithmetic is to save on implementation costs. A cryptographic hash function can use modular arithmetic as the basis of its compression function. This allows the reuse of existing implementation of modular arithmetic. An advantage of these schemes is that it is easy to scale the security level by choosing a modulus of appropriate length. A significant disadvantage is that hash functions based on modular arithmetic are very slow, even when compared to block cipher based construction.

2.4. Hash Function Based on Block Cipher

There have been many efforts to construct hash functions from the existing block ciphers. The main motivation to construct a hash function based on a block cipher is the minimization of design and implementation effort. The advantage of this approach is that the trust in the security of block ciphers can be transformed on to the hash functions. Hash functions developed using block ciphers are either keyed or MDCs. Hash functions based on block ciphers are usually slower when compared to that of the dedicated hash functions. Davies-Meyer, Miyaguchi-Preneel, Matyas-Meyer-Oseas, MDC-2 and MDC-4 are some methods to generate a compression function of a hash function from a block cipher.

2.4.1. Davies-Meyer (DM) Scheme

The DM-scheme was proposed independently by Davies and by Meyer. This scheme can be used with any block cipher. The message block M_i , that is hashed in each step of this scheme has length l equal to the key length k of the block cipher, i.e., $l = k$. The block cipher E takes a block of the message M_i as a

key and H_{i-1} the previous hash value as a plain text to be encrypted. The output of the cipher text is then XORed with the previous hash value H_{i-1} to produce the next hash value H_i .

$$H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1}$$

Figure 3: DM scheme

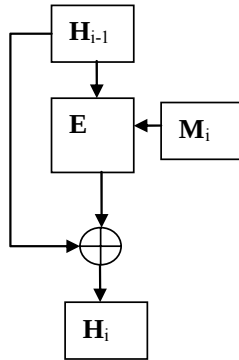


Figure 4: MMO scheme

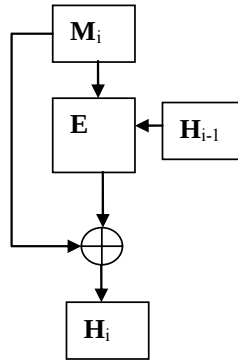
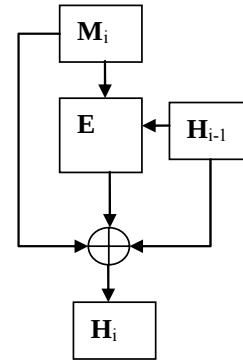


Figure 5: Miyaguchi-Preneel scheme



2.4.2. Matyas-Meyer-Oseas (MMO) Scheme

In this construction current message block is encrypted. In encryption previous hash value H_{i-1} is used as a key. Then this encrypted message block is XORed with current message block M_i to produce hash value H_i . This scheme constructs the compression function as follows:

$$H_i = E_{H_{i-1}}(M_i) \oplus M_i$$

2.4.3. Miyaguchi-Preneel Scheme

This scheme is an extended version of MMO scheme. The only difference is that, in this scheme the previous hash value H_{i-1} is also XORed with the cipher text along with the message block M_i .

$$H_i = E_{H_{i-1}}(M_i) \oplus M_i \oplus H_{i-1}$$

2.4.4. MDC-2 and MDC-4 Scheme

MDC-2 and MDC-4 are manipulation detected codes requiring 2 and 4, respectively, block cipher operations per block of hash input. MDC-2 scheme was originally defined for use with the DES block cipher; however it can be instantiated with any block cipher. The MDC-2 compression function contains two parallel block cipher encryptions and can be seen as a two-way parallel extension of the MMO scheme. MDC-4 employ a combination of four iteration of Matyas-Meyer-Oseas method to generate a double length hash [2, 11, 12].

2.5. Dedicated Hash Function

Dedicated hash functions are specially designed from the scratch for the purpose of hashing a plain text with optimized performance and without being constrained to reusing existing system components such as block ciphers and modular arithmetic. These hash functions are not based on hard problems such as factorization and discrete logarithms. The most popular method of designing compression functions of dedicated hash functions is a serial successive iteration of a small step function. MD2[32], MD-4[17], MD-5[18], SHA-1[25], SHA-2[15], TIGER[41], HAVAL[40], RIPEMD[19] and RIPEMD-160[20] are some examples of dedicated hash functions. Almost all the dedicated hash functions are based on the basic construction of Merkle-Damgård. In next section we provide an overview of some popular dedicated hash functions.

3. Analysis of dedicated Hash Functions

3.1. MD4 and MD5

Ronald Rivest introduced the hash function MD4 (Message Digest 4) in 1990. MD4 was a novel design, oriented towards software implementation on 32 bit architectures. Several hashing algorithms such as MD5, SHA-0, SHA-1, HAVAL, were derived from MD4 hash function called MDx class hash functions.

MD4 hash function is a message digest algorithm which compresses any arbitrary bit-length message into a 128-bit hash value. The input message is processed by 512 bit block. The message is first padded with a single 1 bit followed by a variable number of 0's, so that the size of the message is congruent to 448 modulo 512 and then the last 64 bits are filled with the size of the original message modulo 2^{64} to make the total length of the message divisible by 512. MD4 uses little-endian convention to append 64 bit length. The message is divided into 512 bit blocks. These 512 bit blocks are processed by the compression function of MD4. Then message is split into sixteen 32-bit words denoted by W_t for $t = 0, 1, \dots, 15$, then expanded to provide one word for each step of the compression function. Message expansion in MD4 is simple, it just reuses W_t . There are four chaining variables in MD4. Each chaining variable is a 32 bit register. Collectively they form a four word buffer (A, B, C, D) which is used to compute the message digest. Each run of compression function takes 128 bit four word buffer (A, B, C, D) and 512 bit message block as input and updates the value of four word buffer, to be used as input for the next run of the compression function. Each run of a compression function consists of three rounds and 48 sequential steps (each round consists of 16 steps), where each step is used to update the value of one of the four registers.

Table 2: Boolean functions of MD4

Function name	Steps	Boolean function
F1	$0 \leq t \leq 15$	$(BAC) \vee (\neg BAD)$
F2	$16 \leq t \leq 31$	$(BAC) \vee (BAD) \vee (CAD)$
F3	$32 \leq t \leq 47$	$B \oplus C \oplus D$

Each round uses a different nonlinear auxiliary Boolean function. Each Boolean function takes as input three 32 bit words and produces as output one 32 bit word. The Boolean functions used in three rounds of the compression function are shown in Table2. The step operation of MD4 is of the following form:

$$(A, B, C, D) = ((A + F_i(B, C, D) + W_t + K_i) \ll r, B, C, D), \text{ for } 0 \leq t \leq 47$$

where K_i is an additive constant, $\ll r$ denotes left shift by r bits, $\neg, \wedge, \vee, \oplus$ are respectively logical bit wise Complement, AND, OR and XOR operations (same notations are used in rest of the paper). K_i and F_i , for $1 \leq i \leq 3$, depends on round. After execution of all 48 steps, the compression function uses a feed-forward operation which adds the initial values of the registers to their final values. Boer and Bosselaers described an attack against the last two rounds on MD4. Merkle described an attack against the first two rounds but the work was not published. Vaudenay described another attack against the first two rounds of MD4. In 1996, H.Dobbertin gave a collision attack on MD4 which finds a collision with probability 2^{-22} . H.Dobbertin also showed that the first two rounds of MD4 are not one-way [33, 34, 35]. More recently, Wang et. al. found a very efficient collision attack on MD4, which was improved by Sasaki et. al. [39], due to all these attacks MD4 is no longer used as a collision resistant hash function. In 1992 hash function MD5 was designed by Ronald Rivest as a strengthen version of MD4. Working of MD5 is almost similar to MD4 but some changes have been made to MD4. One extra round is added in MD5. MD5 also compresses arbitrary bit-length input into a 128-bit hash value. Compression function of MD5 consists of 64 sequential steps and 64 different additive constants, one for each step. A new Boolean function, $(\oplus BV - D)$, has been introduced in fourth round. The step operation of MD5 is of the following form:

$$(A, B, C, D) = ((A + F_i(B, C, D) + W_t + K_t + B) \ll r, B, C, D), \text{ for } 1 \leq I \leq 4 \text{ and } 0 \leq t \leq 63$$

Table 3: Boolean functions of MD5

Function name	Steps	Boolean function
F1	$0 \leq t \leq 15$	$(B \wedge C) \vee (\neg B \wedge D)$
F2	$16 \leq t \leq 31$	$(B \wedge D) \vee (C \wedge \neg D)$
F3	$32 \leq t \leq 47$	$B \oplus C \oplus D$
F4	$48 \leq t \leq 63$	$C \oplus (B \vee \neg D)$

In 1993 Boer and Bosselaers found pseudo-collision for MD5. In 1996 H.Dobbertain published an attack that found a collision in MD5 [36,37]. At Crypto'2004, a team of researchers from Shandong University in Jinan China, led by Xiaoyun Wang announced collision in MD5 as well as collisions in other hash functions such as MD4, RIPEMD and HAVAL-128[16]. These results have been improved by Klima and Naito et. al [38,39].

3.2. SHAx Family

The Secure Hash Algorithm (SHA) was developed by National Institute of Standards and Technology (NIST) along with National Security Agency (NSA) and published as a federal information processing standard (FIPS 180) in 1993. This version is often referred to as SHA-0. It was withdrawn by NSA shortly after publication. The NSA suggested minimal changes to the standard because of security issues. The NSA did not disclose any further explanations. A revised version was issued as FIPS 180-1 in 1995 and is generally referred to as SHA-1. The actual standards document is entitled Secure Hash Standard. SHA-1 differs from SHA-0 only by a single bitwise rotation in the message schedule of its compression function. SHA-0 and SHA-1 both produce a 160 bit message digest from a message with maximum size of 2^{64} bits. Here is the description of SHA-1: The input message is processed by 512 bit block. Padding and parsing processes are similar to MD4 and MD5 i.e. the message is padded to make its length congruent to 448 modulo 512 and after appending a 1 bit and 0 bits appropriately, the length is appended as a 64-bit integer. It uses big-endian notation to append 64 bit length to the message. SHA-1 uses five 32-bit chaining variables. This five word buffer (A, B, C, D, E) is used to store intermediate and final result. In SHA-1, the initial values of variables A through D have the same value as they had in MD5. Then padded message is parsed into 512 bit blocks. Then message is split into sixteen 32 bit words. These sixteen words are then expanded to eighty 32 bit words by using following equation:

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \ll 1 \text{ for } 16 \leq t \leq 79$$

Processing logic of these blocks consists of four rounds of 20 steps each. Each step makes the use of a different 32 bit input word W_t . There are four distinct additive constants are used in SHA-1, one for each round. Each round uses a different logical Boolean function. 'IF THEN ELSE' is used by first round, 'XOR' by the second and fourth round and 'MAJORITY' by the third round. These functions are used by the compression function of SHA-1, defined in Table 4. Each round takes as input the current 512 bit block and the 160 bit buffer value and updates the contents of the buffer. The process for each step function can be formally represented as:

$$(A, B, C, D, E) = ((W_t + A \ll 5 + F_i(B, C, D) + E + K_i), A, (B \ll 30), C, D), \text{ for } 1 \leq i \leq 4 \text{ and } 0 \leq t \leq 79.$$

Table 4: Boolean functions of SHA-1

Function name	Steps	Boolean function
F1	$0 \leq t \leq 19$	$(B \wedge C) \vee (\neg B \wedge D)$
F2	$20 \leq t \leq 39$	$B \oplus C \oplus D$
F3	$40 \leq t \leq 59$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
F4	$60 \leq t \leq 79$	$B \oplus C \oplus D$

After the compression function is completed, the results are added to the chaining variables, which compose the message digest at the end. The first result of cryptanalysis of SHA-0 was presented

at Crypto'98 [26]. The authors state that a collision can be found with complexity 2^{61} . This was a differential attack and faster than generic birthday paradox attack. In 2004, Biham and Chen found two near-collisions of the full compression function of SHA-0 [27]. They showed that in SHA-0 near collisions are easy to find than full collisions. The hashes differ by only 18 bits; 142 bits out of 160 bits are equal. In August 2004, a collision for the full SHA-0 algorithm was announced by Joux, Carribault, Lemuet and Jalby. This was done by using a generalization of the Chabaud and Joux attack [28]. The calculation has a complexity of 2^{51} . In February 2005, an attack by Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu was announced which could find collisions in SHA-0 in 2^{39} operations [21]. Some of the methods used for the SHA-0 collisions can also be applied to SHA-1 collision search. In early 2005, Rijmen and Oswald published an attack on a reduced version of SHA-1, 53 out of 80 rounds, which finds collisions with a complexity of fewer than 2^{80} operations [29]. After different cryptanalysts found several attacks on reduced versions of SHA-1, Xiaoyun Wang and her colleagues presented finding collisions with less than 2^{69} hash operations. Soon, they improved their attack to a complexity of 2^{63} [22, 23]. In [24] K. Matusiewicz and J. Pieprzyk presented attack on SHA-1. In [30, 31] Christophe De Cannière et. al. found various attacks on SHA-0 and SHA-1. A modification to the standard SHA-1 hash function's message expansion proposed by Jutla and Patthak [48], in such a way that the minimum distance between the similar words is greater compared with SHA-0 and SHA-1. In August, 2002 NIST has published three new hash functions, SHA-256, SHA-384 and SHA-512. The numeric portion of the name of hash function indicates the size of hash value generated by a hash function. These new hash functions in the SHA family together known as SHA-2. In February 2004, another hash function SHA-224 was added to the SHA-2 family. SHA-384 is a truncated version of SHA-512 with different initialization vectors. In the same way SHA-224 is derived. SHA-224 is a truncated version of SHA-256. The structures of SHA-256 and SHA-512 are almost identical. Analysis of message schedule by H. Gilbert and H. Hanschuh, in [45] determines limits on the probability of collision for SHA-2 and found no weaknesses.

3.3. RIPEMD

The RIPEMD hash function was designed in the framework of the European Race Integrity Primitives Evaluation (RIPE) project. The design of RIPEMD is based on MD4; its compression function consists essentially of two parallel versions of the MD4 compression function. It generates 128 bit message digest. Dobbertin found a collision attack on two rounds of RIPEMD. Later two strengthened versions of RIPEMD are released, RIPEMD-128 and RIPEMD-160. RIPEMD-128 also produces 128 bit message digest as its predecessor. Both RIPEMD-128 and RIPEMD-160 are extended to RIPEMD-256 and RIPEMD-320 respectively. A short description of RIPEMD-160 hash function is as follows: This is a 160 bit message digest algorithm developed by Hans Dobbertin, Antoon Bosselaers and Bart Preneel, and first published in 1996. The algorithm takes as input a message of arbitrary length and produces as output a 160 bit message digest. Like MD4 and MD5, RIPEMD-160 also uses little-endian convention. RIPEMD-160 performs 512-bit blocks processing and uses 5 32-bit chaining variables (160 bit buffer) to hold intermediate and final results. These chaining variables, A, B, C, D, E are initialized to the same values as SHA-1. Boolean functions of RIPEMD-160 are shown in Table 5.

Table 5: Boolean functions of RIPEMD-160

Function name	Steps	Boolean function
F1	$0 \leq t \leq 15$	$B \oplus C \oplus D$
F2	$16 \leq t \leq 31$	$(B \wedge C) \vee (\neg B \wedge D)$
F3	$32 \leq t \leq 47$	$(B \vee \neg C) \oplus D$
F4	$48 \leq t \leq 63$	$(B \wedge D) \vee (C \wedge \neg D)$
F5	$64 \leq t \leq 79$	$B \oplus (C \vee \neg D)$

Compression function computes the new 160 bit buffer from the old buffer and next 16 word block. The compression function consists of five parallel rounds, each containing 16 steps. The total number of steps thus is 160. First two copies are made from old chaining variables (five left and right registers of 32 bits). Both halves are processed independently. Each round takes as input the current 512 bit block and two copies of 160 bit working variables (left and right line). Each round also makes use of a distinct additive constant. There are ten additive constants (one of which is zero). Each step updates one of the registers. At the end of the compression function new 160 bit buffer is computed by adding to each word of old 160 bit buffer one register from left half and one from the right half. In [22] all the detailing of RIPEMD-160 is given.

3.4. Other Hash Functions

There are other hash functions are also like HAVAL [40], TIGER [41], WHIRLPOOL [42], FORK256 [44], HAIFA [43], MD-192[47]. There are three variants of HAVAL: HAVAL3, HAVAL4 and HAVAL5 differ only by the number of rounds. Hash function TIGER produces 192 bit hash value. Other two truncated versions of TIGER are: TIGER-128 and TIGER-160. WHIRLPOOL gives 512 bit message digest. FORK-256 uses four parallel branches to generate 256 bit hash value. FORK-256 is faster than SHA-256. HAIFA also function supports variable hash sizes developed by Eli Biham and Orr Dunkelman. MD-192 is based on the structure of SHA-1. It produces 192 bit hash value.

4. Applications of Dedicated Hash Functions

Hash functions are versatile and powerful primitive of cryptography because it plays an important role in building security applications related to certification, secure communication, authentication and data integrity. Authentication mechanism help establish proof of identities. The authentication process ensures that the origin of an electronic message or document is correctly identified. Hash functions are used in various authentication protocols like Kerberos, IEEE 802.1 X-EAP, and APOP. Integrity deals with privacy and accuracy of information. When the contents of a message are changed after the sender sends it, but before it reaches the intended recipient, the integrity of the message is lost. Certification helps in establishing identity of user and guarantees the validity of signers, documents and issuers. Electronic commerce involves transactions on the internet. Security of these transactions is extremely crucial for the success of the electronic commerce. Hash functions are used in different authentication processes and various integrity checks of numerous email security protocols like Pretty Good Privacy (PGP) and Secure MIME (S/MIME) to ensure a secure communication. IPsec uses hash functions for authentication and data integrity. Internet Key Exchange (IKE) also uses hash function. The signature algorithm DSA and RSA use SHA-1 and MD-5 in some of their variants. Pseudorandom number generators (PRNGs) can be built using hash functions. This PRNG can have infinite period.

- **Digital signatures:** Digital signature has great significance in Web-commerce. Digital signature is an electronic signature that can be used to authenticate the identity of the sender of the message or the signer of a document. Digital signature provides signer authentication and document authorization. It indicates who signed a document, message or record and makes difficult for another person to produce the same without authorization. It also makes impracticable for unauthorized party to falsify or alter either the signed matter or signature. Digital signature scheme is a public key signature scheme uses private key of the sender to create the signature and public key which is ordinarily widely known and is used by relying party to verify digital signature. Hash function is used in both creating and verifying digital signature. Hash function creates a digital representation in the form of the hash value of a fixed standard length which is usually much smaller than the message but substantially unique to it. Any change to the message invariably produces a different hash result when the same hash function is used. By sending a message along with its hash, it is possible to guarantee a message's integrity, that is, the recipient can make sure the message was not altered

(intentionally or by chance) during the communication, the two fingerprints will not match. Hash functions creating digital signatures to operate on smaller and predictable amounts of data, while still providing robust evidentiary correlation to the original message content, thereby efficiently providing assurance that there has been no modification of the message since it was digitally signed. Digital signatures are independent of hash functions it's just more efficient to sign a hash of the message rather than the message itself. Electronic signatures also feature a non-repudiation function, that is, they make it possible to ensure the sender really sent the message.

- **MAC:** A Message Authentication Code (MAC) is designed specially for applications where data integrity is required. The MAC involves cryptographic processing in which both communicating parties share a symmetric secret key K which is not known to anyone else. Sender calculates the MAC by first calculating message digest of the message or document and then applying secret key K to the message digest. Sender then sends original message or document along with calculated MAC to the receiver. Receiver independently computes a MAC over the message and compares the computed MAC to the received MAC. If the two match then receiver concludes that message has not been altered during transit and if it does not match then rejects the message, realizing that the message was changed during transit. MACs differ from digital signatures as MAC values are both generated and verified using the same secret key. This implies that the sender and receiver of a message must agree on the same key before initiating communications, as is the case with symmetric encryption. For the same reason, MACs do not provide the property of non-repudiation offered by signatures. Any user who can verify a MAC is also capable of generating MACs for other messages. In this situation user who sent the message later on can refuse that he had sent the message. MAC also faces key exchange problem.
- **Kerberos:** Kerberos is a widely used authentication protocol. Kerberos allocates the job of authenticating users to a central server and the job of allowing users access to various systems to a different server. Kerberos uses the concept of tickets to authenticate the user. In user authentication phase authentication server generates the secret key. This secret key is nothing but a hash code of a user password calculated with the help of a hash function.
- **One time password (OTP):** One time passwords are the form of authentication. One time password is a password that is valid only for a single login session or transactions. The one time password systems based on hash function works by starting with an initial seed then generating passwords as many times as needed to avoid replay attack.
- **PGP:** PGP provides e-mail encryption and authentication. PGP uses a hash function to ensure the integrity of e-mail message. PGP combines some of the best features of both conventional and public key cryptography. PGP is a hybrid cryptosystem that includes digital signature, compression, encryption and digital enveloping. Digital signature process in PGP includes the creation of message digest of email using a hash function which is encrypted by sender's private key. Then PGP transmits the signature and the plaintext together. Upon receipt of the message, the recipient uses PGP to recompute the digest, thus verifying the signature. As long as a secure hash function is used, there is no way to take someone's signature from one document and attach it to another, or to alter a signed message in any way. The slightest change in a signed document will cause the digital signature verification process to fail.
- **SSL/TLS:** Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols are widely used to ensure secure communication over an untrusted network. A client and server first engage in the handshake protocol to establish shared keys that are subsequently used to encrypt and authenticate the data transfer. To ensure that the obtained keys are as secure as possible, SSL and TLS deploy hash function for key derivation and the authentication step in the handshake protocol.

5. Conclusion

In this paper we have given an extensive overview of cryptographic hash functions. In the past few years various cryptanalysis results have shown that a variety of cryptographic hash functions based on design principle of MD4 are vulnerable to the collision attack. Among these hash functions MD5 and SHA-1 are widely deployed in various cryptographic applications. Although weaknesses have been found in these algorithms but they are still in use today. This may pose a serious security problem. MD-5 and SHA-1 should be replaced and should not be used further for applications. RIPEMD-160 and hash functions of SHA-2 family are better alternatives for these broken hash functions. They are secure with respect to all known attacks including brute force search. NIST announced that they planned to phase out the use of SHA-1 by 2010 in favour of the SHA-2 variants. Although the use of RIPEMD-160 and SHA-2 might still provide enough security for most application today, the cryptographic community must put considerable efforts into the search for better design criteria for the long term security of the hash functions.

References

- [1] S.Bakhtiari, R. Safavi-Naini and J. Pieprzyk, 1995. "Cryptographic hash functions: A Survey", Technical Report 95-09, Department of Computer Science, University of Wollongong.
- [2] A.J. Menezes, P.C. Van Oorschot, S.A. Vanstone Handbook of Applied Cryptography, CRCpress, 1996.
- [3] RSA Laboratories frequently asked questions about today's cryptography, version 4.1.2000. Available: <http://www.rsasecurity.com>.
- [4] P. Rogaway and T. Shrimpton, 2004. "Cryptographic hash-function basics: Definitions, implications and separations for preimage resistance, second-preimage resistance, and collision resistance", FSE 2004.
- [5] D.R. Stinson, 1994. "Universal hashing and authentication codes." Designs, Codes and Cryptography, 4, pp. 369–380.
- [6] D.R. Stinson, 2006. "Some observations on the theory of cryptographic hash functions" Designs, Codes and Cryptography, 38(2), pp. 259–277.
- [7] Ilya Mironov, 2005. "Hash functions: Theory, attacks, and applications", J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp. 68–73.
- [8] Nigel Smart, Cryptography: An Introduction. McGraw-Hill, Third edition, 2003. Available: http://www.cs.bris.ac.uk/~nigel/Crypto_Book/
- [9] I. Damgård, 1987. "Collision free hash functions and public key signature schemes", in: Proc. of Eurocrypt-87, in LNCS, vol. 304, pp. 203-216.
- [10] I.B. Damgård, 1989 "A design principle for hash functions". In Gilles Brassard, editor, Advances in Cryptology: CRYPTO 89, volume 435 of Lecture Notes in Computer Science, pp. 416-427.
- [11] B. Preenel, 1994. "Cryptographic hash functions", Transactions on Telecommunications, VOL5, pp. 431-448.
- [12] Bart Preneel, 1993. "Analysis and Design of Cryptographic Hash Functions", Dissertation, Katholieke Universiteit Leuven.
- [13] William Stallings. Cryptography and Network Security: Principles and Practice. Third edition, Prentice Hall. 2003.
- [14] W. Diffie and M.E. Hellman, 1976. "New directions in cryptography", IEEE Transaction on Information Theory. vIT-22 i6, pp. 644-654.
- [15] NIST, 2002, "Secure Hash Standards", FIPS PUB 180-2.
- [16] X. Wang, X. D. Feng, X. Lai and H. Yu, 2004. "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD", rump session, CRYPTO 2004.
- [17] R.L. Rivest, 1992. "The MD4 Message Digest Algorithm", RFC 1320.
- [18] R.L. Rivest, 1992. "The MD5 Message Digest Algorithm", RFC 1321.
- [19] RIPEMD, Research and Development in Advanced Communication Technologies in Europe, RIPE Integrity Primitives: Final Report of RACE Integrity Primitives Evaluation (R1040), RACE, June 1992.
- [20] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel, 1996. "RIPEMD-160— A Strengthened Version of RIPEMD", Lecture Notes on Computer Science, Volume 1039, Fast Software Encryption 1996, pp. 71–82.
- [21] X. Wang, H. Yu and Y.L. Yin, 2005. "Efficient Collision Search Attacks on SHA-0", CRYPTO 2005.
- [22] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, 2005. "Finding Collisions in the Full SHA-1, Lecture Notes in Computer Science, Volume 3621, Advances in Cryptology – CRYPTO 2005 Proceedings, pp. 17–36.
- [23] Xiaoyun Wang, Andrew Yao, and Frances Yao, 2005. "New Collision Search for SHA-1, Presented at rump session of CRYPTO 2005.
- [24] K. Matusiewicz and J. Pieprzyk, 2006. "Finding good differential patterns for attacks on SHA-1", Lecture Notes in Computer Science, Volume 3969, pp. 164-177.

- [25] NIST, "Secure Hash Standard", 1995. FIPS PUB 180-1.
- [26] Florent Chabaud, Antoine Joux, 1998. "Differential collisions in SHA-0," *Advances in Cryptology-CRYPTO'98*.
- [27] Eli Biham and Rafi Chen, 2004. "Near-Collisions of SHA-0", *Lecture Notes in Computer Science*, Volume 3152, *Advances in Cryptology – Crypto 2004 Proceedings*, pp. 290–305.
- [28] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, William Jalby, 2005. "Collision in SHA-0 and Reduced SHA-1", *Advances in Cryptology-EUROCRYPT 2005*.
- [29] Vincent Rijmen and Elisabeth Oswald, 2005. "Update on SHA-1". In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005*, San Francisco, CA, USA, volume 3376 of LNCS, pp. 58–71.
- [30] Christophe De Cannière, Florian Mendel, and Christian Rechberger, 2007. "Collisions for 70-Step SHA-1, On the Full Cost of Collision Search", In *Selected Areas in Cryptography*, pp. 56–73.
- [31] Christophe De Cannière and Christian Rechberger, 2008. "Preimages for Reduced SHA-0 and SHA-1", In *CRYPTO 2008*, pp. 179–202.
- [32] R.L. Rivest, 1992. "The MD2 Message-Digest Algorithm", RFC 1319.
- [33] Bert den Boer and Antoon Bosselaers, 1991. "An Attack on the Last Two Rounds of MD4", *Lecture Notes in Computer Science*, Volume 576, *Advances in Cryptology – Crypto '1991 Proceedings*, pp. 194–203.
- [34] Hans Dobbertin, 1996. "Cryptanalysis of MD4", *Lecture Notes in Computer Science*, Volume 1039, *FSE 1996*, pp. 53–69, February 1996.
- [35] Hans Dobbertin, 1997. "The First Two Rounds of MD4 are Not One-Way", *Lecture Notes in Computer Science*, Volume 1372, *FSE 1998*, pp. 284–292.
- [36] Bert Den Boer and Antoon Bosselaers, 1994. "Collisions for the Compression Function of MD5", *Advances in Cryptology, Proceedings Eurocrypt '93*, Springer-Verlag LNCS 765, pp. 293–304.
- [37] Hans Dobbertin, 1996. "Cryptanalysis of MD5", *Rump Session, EUROCRYPT 1996*.
- [38] Vlastimil Klima, 2006. "Tunnels in Hash Functions: MD5 Collisions Within a Minute.", *Cryptology ePrint Archive*, Report 2006/105, 2006. Available: <http://eprint.iacr.org/>.
- [39] Yusuke Naito, Yu Sasaki, Noboru Kunihiro, and Kazuo Ohta, 2005. "Improved Collision Attack on MD4", *Cryptology ePrint Archive*, Report 2005/151, May 2005. <http://eprint.iacr.org/2005/151.pdf>
- [40] Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry, 1993. "HAVAL – A One-Way Hashing Algorithm with Variable Length of Output", *Lecture Notes in Computer Science*, Volume 718, *Advances in Cryptology – Auscrypt '92*, pp. 83–104.
- [41] R.J. Anderson, E. Biham, 1996. "TIGER: A Fast New Hash Function", *FSE, LNCS*, vol. 1039, pp. 89–97.
- [42] Paulo S.L.M. Barreto and Vincent Rijmen, 2000. "The Whirlpool Hash Function", *First open NESSIE Workshop*.
- [43] Eli Biham and Orr Dunkelman, 2006. "A framework for iterative hash functions-HAIFA", *NIST Second Hash Functions Work Shop*, Santa Barbara.
- [44] D. Hong, S. Jaechul, S. Hong, S. Lee and D. Moon, 2005. "A new dedicated 256-bit hash function: FORK-256". *First NIST Workshop on Hash Functions*.
- [45] H. Gilbert and H. Hanschuh, SAC 2003, "Security Analysis of SHA-256 and sisters, *Selected Areas in Cryptography*", Ottawa, Canada, *Lecture Notes in Computer Science*, vol. 3006, M. Matsui and R. Zuccheratopp (Eds), Springer, 2004, pp. 175–193.
- [46] R. Merkle, 1989. "One way hash functions and DES. In: Brassard, CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer, Heidelberg.

- [47] H. Tiwari and K. Asawa, 2010, “A Secure Hash Function MD-192 with Modified Message Expansion”, IJCSIS, Vol. 7, No. 2, pp. 108-111.
- [48] C. S. Jutla and A. C. Patthak, 2005. “A simple and provable good code for SHA message expansion”. In IACR ePrint archive 2005/247.