Cryptographic Hash Functions: A Survey

S. Bakhtiari, R. Safavi-Naini, J. Pieprzyk

Centre for Computer Security Research,
Department of Computer Science,
University of Wollongong, Wollongong,
NSW 2522, Australia

Abstract

This paper gives a survey on cryptographic hash functions. It gives an overview of all types of hash functions and reviews design principals and possible methods of attacks. It also focuses on keyed hash functions and provides the applications, requirements, and constructions of keyed hash functions.

1 Introduction

Hash functions map a large collection of *messages* into a small set of *message digests* and can be used for error detection, by appending the digest to the message during the transmission (the appended digest bits are also called parity bits). The error will be detected if the digest of the received message, in the receiving end, is not equal to the received message digest. This application of hash functions is only for random errors, since an active spoofer may intercept the transmitted message, modify it as he wishes, and resend it appended with the digest recalculated for the modified message.

With the advent of public key cryptography and digital signature schemes, cryptographic hash functions gained much more prominence. Using hash functions, it is possible to produce a fixed length digital signature that depends on the whole message and ensures authenticity of the message. To produce digital signature for a message M, the digest of M, given by $H(M)$, is calculated and then encrypted with the secret key of the sender. Encryption may be either by using a public key or a private key algorithm. Encryption of the digest prevents active intruders from modifying the message and recalculating its checksum accordingly. It effectively divides the universe into two groups: outsiders who do not have access to the key of the encryption algorithm and hence cannot effectively produce a valid checksum, and insiders who do have access to the key and hence can produce valid checksums. We note that in a public key algorithm, the group of insiders consists of only one member (the owner of the private key) and hence the encrypted hash value uniquely identifies the signer. In the case of symmetric key algorithms, both the transmitter and the receiver have access to the secret key and can produce a valid encrypted hash for an arbitrary message and therefore, unique identification based on the encrypted hash is not possible. However, an outsider cannot alter the message or the digest.

In the study of hash functions, *Information Theory* and *Complexity Theory* are two major approaches. The methods based on information theory provide unconditional security — an enemy cannot attack such systems even if he/she has unlimited power. This approach is generally impractical.

In the second approach, some assumptions are made based on the computing power of the enemy or the weaknesses of the existing systems and algorithms, and therefore, the security cannot be proven but estimated by the analysis of the best known attacking algorithms and considering the improvements of the hardware and softwares. In other words, hash functions based on complexity theory are *computationally* secure. In this paper, we concentrate on the second approach.

The organization of the paper is as follows. This section will be completed by providing notations, definitions, and types of hash functions. Section 2 study the design principals of hash functions. Methods of attack on hash functions are introduced in Section 3. Section 4 gives applications, requirements, and constructions of keyed hash functions. Some open problems are given in Section 5.

Notation	*Description*
P	*Plaintext* in an encryption algorithm.
C	*Ciphertext* in an encryption algorithm.
IV	*Initial vector* of a hash function.
M	Arbitrary length input *message* of a hash function.
MD	*Message digest* of a hash function. (Other names for MD are: *digest, hashcode, hashtotal, hashresult, imprint, checksum, compression, compressed encoding, seal, authenticator, authentication tag, fingerprint, test key, condensation, message integrity code,* and so on.)
X	Intermediate value (or chaining variable) in a cryptosystem.
K	Secret *key* in a cryptosystem.
x_i	i^{th} element (or block) of x, where $x \in \{P, C, M, X, K\}$.
A	One communicant (Alice).
B	Another communicant (Bob).
E	The opponent (Eve).
$E(K,P)$	*Encryption* algorithm $E()$ with a key K and a plaintext P.
$D(K,C)$	*Decryption* algorithm $D()$ with a key K and a ciphertext C.
$H(M)$	Keyless *hash* function $H()$ with input message M. (The notation $H(IV, M)$ may be used when the initial vector IV is emphasized.)
$H(K,M)$	Keyed *hash* function $H()$ with secret key K and input message M. (The notation $H(K, IV, M)$ may be used when the initial vector IV is emphasized.)
mod	Modular reduction or modular computation (reminder of a division).
div	Integer part of a division.
and	Bit-wise *and* operation.
or	Bit-wise *or* operation.
not	Bit-wise *not* operation. (Flipping the bits.)
xor	Bit-wise *exclusive-or* operation (addition modulo 2). Sometimes $\oplus$ is used instead.
$\underline{\oplus}$	Special bit-wise *exclusive-or* operation, where the operands may have different lengths. The XOR will be performed bitwise, starting from the right most bits.
$\overline{\oplus}$	The same as above, but starting from the left most bits.
$\|\|$	Concatenation.

Table 1: *Notations.*

1.1 Notations and Preliminaries

Table 1.1 gives all the notations that are used in this paper, where *encryption/decryption algorithm* is an algorithm that is used for encryption/decryption, and the term *cryptosystem* is a general term for all cryptographic algorithms, such as hash functions.

We use the term *hard* in this paper as *computationally infeasible.* The terms *opponent, adversary, cryptanalyst, intruder*, and *enemy* have the same meaning.

1.2 Definitions

Some useful definitions are given in this section.

Definition 1 *A function $H()$ that maps an arbitrary length message M to a fixed length message digest MD is a* One-Way Hash Function (OWHF), *if it satisfies the following properties:*

1. *The description of $H()$ is publicly known and should not require any secret information for its operation.*

2. *Given M, it is easy to compute $H(M)$.*

3. *Given MD in the rang of $H()$, it is hard to find a message M such that $H(M) = MD$, and given M and $H(M)$, it is hard to find a message M' ($\neq M$) such that $H(M') = H(M)$.*

Definition 2 *A function $H()$ that maps an arbitrary length message M to a fixed length message digest MD is a* Collision Free Hash Function (CFHF), *if it satisfies the following properties:*

1. *The description of $H()$ is publicly known and should not require any secret information for its operation.*
2. *Given M, it is easy to compute $H(M)$.*
3. *Given MD in the rang of $H()$, it is hard to find a message M such that $H(M) = MD$, and given M and $H(M)$, it is hard to find a message M' ($\neq M$) such that $H(M') = H(M)$.*
4. *It is hard to find two distinct messages M and M' that hash to the same result ($H(M) = H(M')$).*

Definition 3 *A function $H()$ that maps a fixed length key K and an arbitrary length message M to n-bit message digest MD is a* Secure Keyed Hash Function (SKHF) *or simply a Keyed Hash Function, if it satisfies the following properties:*

1. *The description of $H()$ is publicly known.*
2. *Given K and M, it is easy to compute $H(K, M)$.*
3. *Without knowledge of K, it is hard both to find M when $H(K, M)$ is given, and to find two distinct messages M and M' such that $H(K, M) = H(K, M')$. (These properties are optional when the secret key K is public.)*
4. *Given (possibly many) pairs of $[M_i, MD_i]$ with $MD_i = H(K, M_i)$, it is hard to find the secret key K.*
5. *Without knowledge of K, it is hard to determine $H(K, M)$ for any message M, even when a large set of pairs $[M_i, H(K, M_i)]$, where M_i's are selected by the opponent ($M \neq M_i$, $\forall M_i$), is given.*

Berson *et al.* [10] have proposed the idea of *Collisionful Hash Functions (CHF)*, where some degree of collision is desired rather than avoided, to make the key leakage more difficult. Gong's version of the definition of CHF is as follows [40].

Definition 4 *A function $H()$ that maps a fixed length key K and an arbitrary length message M to a fixed length message digest MD is a* Collisionful Hash Function (CHF)*, if it satisfies the following properties:*

1. *Given K and M, it is easy to compute $MD = H(K, M)$.*
2. *Given K, it is hard to find two distinct messages M and M' such that $H(K, M) = H(K, M')$.*
3. *Given t pairs of $[M_i, MD_i]$, with $MD_i = H(K, M_i)$, $i = 1, \ldots, t$, it is hard to find the secret key K, though it is less hard to find a K' ($\neq K$) with $H(K, M_i) = H(K', M_i)$, for all M_i.*
4. *Without knowledge of K, it is hard to determine $H(K, M)$ for any message M.*

Universal Hash Functions (UHF) were defined by Carter and Wegman [24] in an attempt to provide an input independent average linear time algorithm for storage and retrieval of keys in associated memories.

Definition 5 *A class H of functions from a set A to a set B is called* universal$_2$*, if for all x and y in A, $\delta_H(x, y) \leq |H|/|B|$, where $|H|$ and $|B|$ are the sizes of H and B, respectively, and $\delta_H(x, y)$ denotes the number of functions $h \in H$ with $h(x) = h(y)$.*

In [25], they extended their work and defined *strongly universal$_n$* and *almost strongly universal$_2$*, and showed their application to authentication.

Definition 6 *A class H of hash functions is* strongly universal$_n$ *(SU_n) if given any n distinct elements $a_1, \cdots, a_n$ of A and any n (not necessary distinct) elements $b_1, \cdots, b_n$ of B, then $|H|/(|B|^n)$ functions take a_1 to b_1, a_2 to b_2,* etc. *SU_n hash functions can be used for multiple authentication.*

1.3 Types of Hash Functions

In this section a brief description of different types of hash functions are given. For each type, some existing functions are presented (or cited).

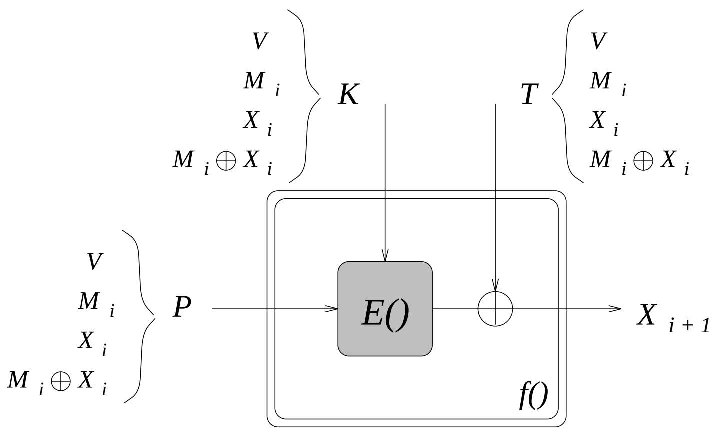

Figure 1: *Round function for a set of hash functions based on block ciphers.*

1.3.1 Hash Functions Based on Block Ciphers

There have been many efforts to base the hash functions on the existing block ciphers and avoid the construction of a hash function from scratch. Also when hardware implementation of a block cipher is available, it is desired to use it both to have a fast algorithm and to decrease the cost of the design.

In the contrast, some block ciphers might not be useful for design of hash functions (they might provide some weak points). The export restriction on encryption algorithms is another problem, in some countries.

Figure 1 illustrates the general construction of a round function $f()$ for the hash functions that are based on block ciphers. $E()$ is a block cipher that takes an input P and a key K to produces an output. The arbitrary length message M is divided into n blocks $M_1, \ldots, M_n$ and each of which is processed in one round. Based on Figure 1, the input P, the key K, and the XORed value T are chosen from the set $S = \{V, M_i, X_i, M_i \oplus X_i\}$, where V is a constant value and X_i is the output of the previous round. This provides $4^3 = 64$ different possibilities for $f()$, where some of the possibilities should be discarded, as M_i should be used at least once in the round function $f()$. Preneel [63] gives detailed analysis of hash functions based on the above $f()$, where most of them are found insecure.

Unfortunately, there are very few encryption based hash functions that are secure for message authentication. Examples of some encryption (not necessarily block cipher) based hash functions are in [2, 15, 44, 45, 52], and the reader is referred to [63] for the study of the weaknesses of these hash functions.

1.3.2 Hash Functions Based on Modular Arithmetic

The idea of cryptosystems based on modular arithmetic is to reduce the security of a system to the difficulty of solving the problems in the number theory. Two important hard problems in number theory are *factorization* and *discrete logarithm*.

- *Factorization Problem* is the problem of finding two integers p and $q > 1$, such that $pq = N$, and N is a given positive integer.
- *Discrete Logarithm Problem* is the problem of finding $\log_g^s$ (modulo N), where s and g are elements of a Galois field $GF(N)$, and N is a prime number or a power of 2.

The two most important cryptosystems, based on modular arithmetic, are RSA public key cryptosystem [71] and ElGamal cryptosystem [38]. Hash functions that are based on modular arithmetic can have variable digest length, depending on the size of modulus. Examples of the attack on this type of hash functions are: *to find fixed points in the modular expression, to find small numbers that result in (again) small numbers (less that the modulo), and multiplicative attack.*

In [63, Sections 6.3 and 7.3.1], Preneel refers to two keyed hash functions which were designed by F. Cohen for the anti-virus package ASP [28]. They generate a secret checksum for each program and evaluate it every time that the program is being loaded. Preneel has given a short description of these algorithms and has examined some of their weaknesses.

1.3.3 Hash Functions Based on Cellular Automaton

There are some automata based cryptosystems that among them only a few are hash functions. Wolfram [79] has suggested the use of one-dimensional cellular automaton for pseudo-random bit generator. The one-wayness property of such a generator can lead to the design of a hash function.

An example of the automata based hash function is *Cellhash* which was proposed by Daemen *et al.* in [31, 32]. This hash function is hardware oriented and has some disadvantages [63, Section 7.2.7].

1.3.4 Hash Functions Based on Knapsack Problem

Knapsack problem can be defined, in general, as: *Given a set* $U = \{u_i \mid i = 1, \ldots, n\}$ *and an integer valued function* $s : U \rightarrow \mathbb{N}$, *is there a subset* $U' \subseteq U$ *such that* $\sum_{u_i \in U'} s(u_i) = B$, *where* B *is a given integer?* This problem was first used in Merkle-Hellman public key cryptosystem [56]. Although it was broken later, there have been a number of research efforts to find a secure cryptosystem based on knapsack problem. Two examples of the hash functions based on knapsack problem are *hash functions based on additive knapsacks* and *hash functions based on multiplicative knapsacks* [23, 35, 36, 39, 80] (cf. [63]).

1.3.5 Hash Functions Based on Algebraic Matrices

In [41], Harari used Algebraic Matrices to generate keyed one-way functions for authentication. A random $t \times t$ matrix K is used as the key and is multiplied to the $1 \times t$ message matrix M. The message digest is computed as $MD = M^tKM$ (or $MD = K^tMK$), where X^t is transpose matrix of X. There are some weaknesses in the proposed algorithms, such as the possibility of finding collisions, which are pointed out in [63].

1.3.6 Dedicated Hash Functions

There are several hash functions that are especially designed for hashing and are not provably secure. (They are not based on a hard problem such as factorization.) In this section a brief review of these hash functions and the existing attacks (if any) are given.

MD2 [47] is one of the MD-family hash functions that were proposed by RSA Data Security Inc. The algorithm generates a 16-byte message digest for an arbitrary length input message. The author has conjectured that:

1. The difficulty of finding a message with a given message digest is on the order of 2^{128} operations.

2. The difficulty of finding two messages with the same message digest is on the order of 2^{64} operations.

However, careful study of MD2 (cf. Appendix A.1) reveals some weak points. For example, in the process of the last 16-byte block, the last 33 operations ($T = X_k = X_k \oplus S_T$, for $k = 17, \ldots, 48$, and $T = (T + j)$ **mod** 256) can be omitted, because, X_{17} to X_{48} are discarded at the final stage. Preneel [63] states that the most successful approach to find collisions seems to be a differential attack (cf. Section 3.2.5). There is a recent attack by Rogier and Chauvaud [72] which can find collisions based on some weaknesses in MD2.

MD4 [69] is another MD-family hash function which uses a very simple structure on 32-bit machines, and is believed to be a fast hashing algorithm (cf. Appendix A.2). The resulting message digest length is 128 bits and the same claim about the difficulty of finding collisions, which was mentioned for MD2, is conjectured by the designer. Boer and Bosselaers [18] have found an attack on the last two rounds of MD4. They can find collisions for MD4, if the first round of the algorithm is omitted. Vaudenay [78] has also shown how to construct collisions for MD4, when the last round is omitted. His attack also finds two close digests (according to Hamming distance) when the full MD4 is used. For the time being, there is not any attack for the full version of MD4.

Name	*Type*	*Speed (Kbit/s)*
MD2	Dedicated (keyless)	78
FFT-hash I	Dedicated (keyless)	212
N-hash	Dedicated (keyless)	266
Snefru-8	Dedicated (keyless)	270
SHA	Dedicated (keyless)	710
BCA	Dedicated (keyless)	764
RIPEMD	Dedicated (keyless)	1334
MD5	Dedicated (keyless)	1849
MD4	Dedicated (keyless)	2669

Table 2: *A comparison on the performance of several hash functions, on a 16MHz IBM PS/2.*

MD5 [70] is the strengthened version of MD4, where one extra round is added and each round consists of more operations (cf. Appendix A.3). Although MD5 is considered as one of the most promising hash functions, current technologies demand a faster hash function [74].

For the time being, there is not any successful attack on the full MD5. Berson [9] introduced the idea of differential cryptanalysis modulo 2^{32} and applied to MD5. He could find collisions for individual round functions, but the idea could not be extended to the full algorithm. Also, Boer and Bosselaers [19] gave an algorithm to find *pseudo-collisions* for MD5 (cf. Section 2). The attack is hardly considered as a cryptographic attack and the resulting initial vectors always differ in the most significant bits of the four 32-bit words.

As Kaliski and Robshaw have expressed in [46], the attack is not considered as an active attack, because it does not imply a real collision, and the resulting messages are the same. Moreover, the attack does not allow the intruder to arbitrary fix one of the initial vectors.

SHA [59] is another strengthened version of MD4, where, for instance, the digest length is increased from 128 bits to 160 bits and the number of steps per rounds is increased from 16 steps to 20 steps. These changes make SHA slower than both MD4 and MD5. There is not any successful attack on SHA (for the time being).

HAVAL [83] is very similar to MD5 with the following advantages:

1. It uses five nonlinear boolean functions with Strict Avalanche Criterion (SAC) property.
2. It has 15 different versions by choosing the number of passes (3,4, or 5) and the digest length (128, 160, 192, 224, or 256 bits).
3. It is 60% faster than MD5 when 3 passes are required, and as fast as MD5 when full 5 passes are required.

Other Dedicated Hash Functions are *Snefru* [55], *RIPE-MD* [65, 68, 76], *FFT-Hash I and II* [8, 30, 77], *BCA (Binary Condensing Algorithm)*, *MAA (Message Authentication Algorithm)*, and *DSA (Decimal Shift and Add)* that are not included in this paper.

Table 2 gives a comparison of the performance among different hash functions (extracted from [63]).

2 Design Principals for Hash Functions

Study of hash functions clears that most hash functions have similar structures. We suggest the following structure for hash functions:

1. *Choose a good round function $f()$, depending on the required level of the security.* For instance, if a one-way hash function is required, the round function should be so. (Suppose the function maps a t-bit block into an r-bit block, where $128 \leq r \leq t$.)

2. *Use a padding procedure and pad a suffix to the arbitrary length message M such that the padded message length (in bits) becomes a multiple of t.* The following padding procedure is suggested here:

   (a) Store the length of M modulo 2^s in a s-bit variable L and pre-pend it to the message.

   (b) Append a single '1' bit followed by enough '0' bits to the message such that the length of the result (in bits) is $(t - s)$ modulo t. (At least one bit is appended.)

   (c) Append the s-bit variable L to the previous result. This makes the padded message length equal to a multiple of t.

   The padded message will be $(L \parallel M \parallel 100\ldots0 \parallel L)$, where '$\parallel$' denotes concatenation. It is also safer to append a plain checksum of the entire message before going to the next step.

3. *Divide the padded message into n t-bit blocks $M_1, \ldots, M_n$.* This allows to process the message in t-bit blocks, using the round function $f()$.

4. *Choose a good initial vector IV (probably by a random choice), and store it in an r bit buffer X_1.*

5. *Perform the round function $f()$ to the all n blocks, using a chaining method ($X_{i+1} = f(X_i, M_i)$, $i = 1, \ldots, n$).*

6. *The output of the above process is the message digest.* That is, $MD = X_{n+1}$.

It is clear that, to have a weak/strong one-way hash function, the underlying round function should be so. (This is a necessary condition, that is, a hash function might not be one-way even if the underlying round function is so.) The above structure thwarts general attacks, such as birthday attack, but the cryptanalyst has still a chance to attack the hash function based on the weaknesses of use of the underlying round function. In summary, the round function should have the following properties.

It should be secure against possible attacks (see next section) and satisfy the security assumptions. For example, to have a collision free hash function the round function *must* be collision free, because, if the opponent can find two distinct messages M_i and M_i' such that $f(X_i, M_i) = f(X_i, M_i')$, for some chaining variable (intermediate value) X_i, he/she can find a collision for the entire hash function by just replacing M_i with M_i'.

In some cases, the underlying round function $f()$ might have a weakness that allows the intruder to find a pseudo collision. If the opponent can find two messages M_i and M_i' and two chaining variables X_i and X_i' such that $f(X_i, M_i) = f(X_i', M_i')$, a pseudo collision is found. Sometimes pseudo collisions may lead to real collisions. For instance, for the above pseudo collision, if the enemy can find a block M_i'' such that $f(X_i, M_i'') = X_i'$, he/she can replace M_i with $(M_i'' \parallel M_i')$, because $f(X_i, M_i) = f(f(X_i, M_i''), M_i')$. In general, the ability of modifying the chaining variable, by traveling back and forth in the chaining process, should be prevented.

Existence of *fixed points* for the round function is another weakness. The cryptanalyst may find a chaining variable X_i such that $f(X_i, M_i) = X_i$. In this case M_i can be inserted without affecting the result. However, adding the message length to the process provides some protections against this insertion (another fixed point should be found and removed from the message to keep the message length intact).

A hash function must use every bit of the message block many times in the round function. This prevents the cryptanalyst to create undetectable changes in the message, as changing a bit will cause many changes to the result of the round function. This can be done more efficiently when the round function is especially designed for hashing (probably from scratch). For instance, if the round function is an encryption function such as DES, the invertiblity might harm the one-wayness of the hash function.

Choosing an appropriate round function, the process of the whole hash function should be analyzed. Any irregularities in the distribution of the output can lead to an attack. Differential attack (cf. Section 3.2.5) is an instance attack that can use such irregularities.

In keyed hash functions, since a secret key contributes to the process, the designer can base the hash function on either the security or the efficiency — he/she can make the hash function secure against the outsiders (who do not know the key) or make it secure for all people (including the insiders who know the key). The first approach allows the designer to construct an efficient algorithm, by a well studied use of the secret key in the process. The security is increased if the secret key contributes in the process of at least the first and the last rounds, because, the opponent will not have access to the components of the algorithm which is protected by a secret key.

If the designer intends to design a keyed hash function that is secure when the key is known, the design procedure becomes more complex. The problem of having a keyed hash function secure against the insiders is left open by Preneel [63]. However, in most cases, such a property will not be useful. For instance, if for a given key K, the keyed hash function $H()$ is collision free, it means that it is not possible to find two distinct messages M and M' such that $H(M) = H(M')$. This prevents a sender to send a pair $[M, H(M)]$ to a receiver and claim that $[M', H(M')]$ is sent. However, a key holder could send $[M, H(M)]$ and claim that $[M', H(M')]$ is sent, since he/she has access to the secret key and can calculate $H(M')$ for any M'. This problem can be solved by using techniques such as digital signature which certainly reduce the efficiency.

At present, hash functions that are based on a hard problem, such as factorization, are not efficient, and dedicated hash function are being used in practice.

The methods of hashing, described in Section 1.3, are *Serial Methods* [62] or *Chaining Methods* [63]. In *Parallel Method* [62] or *Tree Approach* [63], the hashing process can be sped up, using many processors. A simple parallel construction is as described below.

$$X_i^1 = f(M_{2i-1}, M_{2i}), \qquad i = 1, \ldots, 2^{q-1},$$

$$X_i^j = f(X_{2i-1}^{j-1}, X_{2i}^{j-1}), \qquad i = 1, \ldots, 2^{q-j}, \;\; j = 2, \ldots, k-1,$$

$$MD = f(X_1^{k-1}, X_2^{k-1}),$$

where $k = 2^q$, $t = 2r$, q is an integer, and $f()$, X_i^j, M_i, and MD have the same meanings as before. The message digest is $O(\log k)$ instead of $O(k)$. Similar to the chaining method, the underlying function $f()$ must be secure [36, 64].

To design a hash function, one can also use nested form of existing hash functions. For instance, if the hash functions $H_1()$ and $H_2()$, and the round function $f()$ exist, one can define the following hash functions (M and X are message and chaining variable, respectively):

$$H(M) = f(H_1(M) \,||\, H_2(M)),$$

$$H(M) = H_1(M) \,||\, H_2(M)).$$

It is also possible to form a round function $g()$ from an existing round function $f()$ and a hash function $H()$:

$$g(X, M_i) = f(H(X), H(M_i)).$$

The new hash function can be described as,

$$H'(M) = g(g(\cdots g(IV, M_1) \cdots, M_{n-1}), M_n),$$

where IV is the initial vector. More examples are given in Section 4.3.1.

3 Methods of Attack on Hash Functions

In this section an overview of the known methods of attack on hash functions is presented. The attacks are divided into two major groups. The ones that depend on the weaknesses of the underlying algorithm and the ones that do not depend on the algorithm.

A successful attack on a hash function means to find a way to falsify a claimed security property of the hash function. For example, if a hash function is claimed to be one-way, a successful attack is to find at least one case that a message can be constructed for a given digest.

In keyed hash functions, since a secret key contributes to the hashing process, the methods of attack on the secret key should be included. If the cryptanalyst can find a method to extract the secret key, the system is entirely compromised (during the key life time). There might also be some weaknesses in the algorithm that allow the intruder to bypass the secret key. Existence of weak keys is one example of this case, where the algorithm results in the same digest for several keys (cf. either for a fixed message or all messages). The intruder can then guess the key, since the key domain is shrunk by some factors. Therefore, design of the keyed hash functions needs careful consideration.

In this section, we also study high level attacks which can be applied to the protocols that use hash functions.

3.1 General Attacks (independent of the algorithm)

Assuming that a hash function uniformly distributes the set of messages to the set of possible digests, there are some general methods available to an attacker. These methods do not assume knowledge of the algorithm and only depend on the message digest length, and the key length in keyed hash functions. Examples of these attacks are *Birthday Attack, Exhaustive Key Search, Random Attack,* and *Pseudo Attack.*

3.1.1 Birthday Attack

This attack is originated from *Birthday Paradox* which is the probability of finding at lease two people with the same birthday among 23 people. To describe the attack, we assume that the message digest length is r bits, which provides 2^r possibilities for the message digest. If two pools from the digest space, one containing x_1 samples and the other containing x_2 samples, are generated by an adversary, the probability of finding a match between the two pools is approximated by,

$$P \approx 1 - e^{-\frac{x_1 x_2}{2^r}},$$

where x_2 is a big integer [61]. In particular, when $x_1 = x_2 = 2^{\frac{r}{2}}$ and $x_1, x_2 = O(\sqrt{n})$, the probability of the match equals 63%. That is,

$$P \approx 1 - \frac{1}{e} \approx 0.63.$$

The important advantage of birthday attack is that, if the opponent slightly increases the size of the samples (x_1 and x_2), the above probability will significantly increase. In practice, the opponent wishes to substitute a bogus messages with a genuine message. Assuming that x_2 genuine digests are available, the probability of a match equals $1 - e^{-\frac{x_2}{2^r}}$.

In keyed hash functions (assuming that the key is kept secret against the adversary), the adversary cannot calculate the digest corresponding to the bogus message, and therefore, cannot find a match. However, this technique can be used to find collisions for genuine pairs of [message , digest].

Preneel [63] gives a complete description of this attack and suggests that a 160-bit message digest will be secure against this attack for at least 20 years. Pieprzyk and Sadeghiyan [62] recommend a 128-bit message digest to achieve the security against this attack.

3.1.2 Exhaustive Key Search

In keyed hash functions, a secret key is used in the hashing process to make the algorithm secure. If the adversary has access to at least one pair [message , digest], the key can be found by examining the key space elements against the [message , digest] pair(s). Since the map from message space to digest space is not one-to-one, more than one key could be found. However, it might be possible to determine the key uniquely, if a large number of pairs is given.

As stated in [63], the expected number of trials to find the secret key is upper bounded by,

$$(1 - \frac{1}{2^r}) \sum_{i=1}^{m} \frac{i}{2^{r(i-1)}} \; < \; \frac{1}{1 - 2^{-r}},$$

where r is the message digest length and m is the number of [message , digest] pairs. The total number of trials to identify the key is upper bounded by,

$$m + \frac{2^k - 1}{1 - 2^{-r}},$$

where k is the key length (in bits). The number of resulting keys (including the real key) is expected to be $1 + \frac{2^k - 1}{2^{mr}}$ (cf. [63]). It means that, about $\frac{k}{r}$ pairs of [message , digest] are needed to uniquely determine the secret key (since, $(1 + \frac{2^k-1}{2^{mr}}) \approx 2^{k-mr}$).

3.1.3 Random Attack

Another attack on hash functions, that is independent of the algorithm, is *Random Attack*. In this attack, the adversary chooses a random message (or part of a message) and hopes that its message digest is equal to a genuine one.

The success probability of this attack for a hash function which holds the required random behavior is $\frac{1}{2^r}$, where r is the message digest length. It is suggested to have at least 64 bits for the message digest to thwart this attack.

3.1.4 Pseudo Attack

We have called this attack *Pseudo Attack*, since the cryptanalyst tries to find a pseudo key $\widehat{K}$ with $H(K, M) = H(\widehat{K}, M)$, where $H()$ is a keyed hash function, K is the real key, and M is a message. (This is similar to finding more than one key in Exhaustive Key Search.) This may allow the enemy (Eve) to identify herself as a legitimate user who has access to the secret key.

A pseudo key $\widehat{K}$ for some given [message , digest] pairs does not necessarily generate a correct message digest for another message. In other words, suppose a key K is used to generate t pairs $[M_1, MD_1]$, $[M_2, MD_2]$, ..., $[M_t, MD_t]$, where $MD_i = H(K, M_i)$, $i = 1, \ldots, t$. Now if the intruder can find a pseudo key $\widehat{K}$ with $MD_i = H(\widehat{K}, M_i)$, $i = 1, 2, \ldots, t$, it does not necessarily imply that for an $M'(\neq M_i,\ \ i = 1, 2, \ldots, t)$, $H(K, M') = H(\widehat{K}, M')$. This attack is also discussed in [6].

3.2 Special Attacks (that depend on the algorithm)

In this section, the methods of attack which are based on the weaknesses of the round function or, in general, the hash function, are briefly described. However, these attacks would not be successful on keyed hash functions, because a secret key protects the components of the hash function against the outsiders. The attacks that are discussed in this section are *Meet in the Middle Attack, Correcting Block Attack, Fixed Point Attack, Attack on the Underlying Encryption Algorithm, Differential Cryptanalysis,* and *Linear Cryptanalysis.*

3.2.1 Meet in the Middle Attack

This attack is a variation of birthday attack and is applicable to the hash functions that use a round function $f()$, invertible to the chaining variable X or the message block M_i. It allows the opponent to construct a message which corresponds to a certain message digest. To apply this attack, the adversary should generate x_1 samples for the first and x_2 samples for the last part of a bogus message. He/she goes forwards from initial value and goes backwards from the hash value, and the probability that the two intermediate values are same, is given by,

$$P \approx 1 - e^{-\frac{x_1 x_2}{2^r}},$$

where r is the length of initial vector, intermediate values, and message digest (cf. [63, Appendix B]). Now if a meeting point is found, the concatenation of the message parts forms a bogus message that results in the given hash value.

3.2.2 Correcting Block Attack

In this attack, the opponent uses a pre-existing [message , digest] pair, and tries to change one or more message blocks such that the resulting digest remains intact. One round of MD5 is vulnerable to this attack, where the cryptanalyst takes a message block M_i (16 words), fixes 11 message words, modifies one word, and calculates the remaining 4 words to form another message block M_i' which maps to the same digest.

A simple solution is to provide redundancy to the message blocks in such a way that the message digest heavily depend on every bit of the message. Full MD5 is an instance, where each 32-bit word of the message block contributes in all four rounds of the algorithm. Disadvantage of this solution is the decrease in the speed of hashing.

3.2.3 Fixed Point Attack

In this attack, the intruder looks for a *fixed point* for the round function $f()$. A fixed point is a chaining variable (intermediate value) X_i which satisfies $f(X_i, M_i) = X_i$. In other words, existence of the message M_i does not change the final result. Therefore, M_i can be inserted to the message, whenever the chaining variable equals X_i. Again, there are very simple methods to thwart this attack. Similar to the previous attack, the designer can make redundancy to the input blocks and/or simply add the message length to the input message, before performing the last rounds. Note that, in the latter case, if there are plenty of fixed points, the enemy may insert M_i with $f(X_i, M_i) = X_i$ to the i^{th} round and remove M_j with $f(X_j, M_j) = X_j$ from the j^{th} round. In this case the message length remains intact.

3.2.4 Attack on the Underlying Encryption Algorithm

In Section 1.3, the hash functions based on encryption algorithms were briefly discussed. Because the encryption algorithms are designed for encryption and decryption, they might have some weaknesses when they are used as the round function of a hashing algorithm.

The major weaknesses of the encryption algorithms are *Key Collisions, Complementation Property, Weak Keys,* and *Fixed Points* [63]. Details on these weaknesses are beyond the scope of this paper, and the reader is referred to [1, 16, 21, 20, 42, 48, 49, 50, 51, 58, 60, 66, 67] for more information.

3.2.5 Differential Cryptanalysis

The idea of this attack was first given by Biham and Shamir in [14]. In Differential Cryptanalysis, the correlation between the difference in input and output is studied. In other words, the intruder searches for a particular difference in input that cause a specific difference in output. This attack is applied to almost all cryptosystems, including most dedicated hash functions. In the case of hash functions, the difference in output should be zero to result in collisions. Examples of this attacks are in [9, 11, 14, 15, 16, 17, 51].

3.2.6 Linear Cryptanalysis

Linear Cryptanalysis was proposed by Matsui [54] in early 1993. Although it is inspired by Differential Cryptanalysis, better results are gained compared with Differential Cryptanalysis (specially on block ciphers such as DES) [12, 13, 26, 43, 54].

For the time being, there is no proposed attack on the hash functions, based on Linear Cryptanalysis. However, the hash functions based on encryption algorithms are expected to be the most vulnerable hash functions against this attack. An implementation of Linear Cryptanalysis on DES can be found in [4, 5].

3.3 High Level Attacks

This section gives an overview of some high level methods of attack on hash functions, when they are used in a protocol or for a non-hashing purpose.

The first method is *Replay Attack* or *Restore Attack*. The components of the hash function (or scheme) are not changed in this attack, but they are reused. The opponent may store transmitted information and retransmit it at a later time. On the other hand, an adversary may delete the contents of a transmission. Therefore, one party will not receive the message that another party assumes he/she has sent to that party. In banking transmissions, in particular, such an attack can let the intruder to earn millions of dollars.

Although these attacks are very serious, there are very simple methods to thwart them. To avoid replay attack, the transmission date and time can be attached to the message. This attached part is called *time stamps*. The time resolution should be higher than the resolution of the message transmission. For example, if a message can be received at least 2 second after the previous one, the time stamp should include even the seconds of the transmission. However, there are some difficulties to synchronize the clock pulses and also to remove the delays in a communication channel.

Another high level attack is *Padding Attack*. Suppose a keyed hash function is constructed from an unkeyed one by just using the initial vector as the secret key. If Alice sends $[M, MD]$ as a pair of [message , digest] to Bob, the adversary (Eve) can easily pad M with an arbitrary message M' and evaluate the message digest corresponding to $(M \parallel M')$. Then the pair $[M, MD]$ would be substituted by $[(M \parallel M'), MD']$. Padding attack can be easily thwarted by pre-pending the message length to the message or by using some fixed suffixes that are not appeared within the message. More examples of this attack are in [29, 33, 34]. Some methods of avoiding this attack are also suggested in [6].

4 Keyed Hash Functions

Keyed hash functions (cf. Definition 3) can be used for message authentication. In this section, applications, security requirements, and constructions of keyed hash functions are studied.

4.1 Applications of Keyed Hash Function

Several applications of keyed hash functions are discussed in this section, and advantages and disadvantages of each application is reviewed. These applications are mostly authentication schemes that use symmetric keys to make the algorithm secure.

4.1.1 Message Authentication between Two Parties

Although *electronic signature* schemes ([73]) can be used for message authentication, they are usually slow and inefficient. They also need key distribution schemes for public keys and therefore the problem remains with the authenticity of the public key and so on. When two parties need to communicate, it is easier to use a symmetric key rather than public and secret keys. We also note that keyed hash functions are preferred to be used in authentication schemes (compared with the schemes based on encryption algorithms), since hashing and encryption are combined and, hence, higher speed can be achieved.

Having a secure keyed hash function (examples are in Section 4.3), a message authentication between two communicants can be provided by sending the pair $[M, MD]$ from one communicant to the other, where MD is the message digest of the message M, using the keyed hash function and a secret key shared between the two communicants.

4.1.2 Message Authentication among Multiple Parties

In some applications, such as *Electronic Mail (Email)*, a message can be broadcasted to people to reduce the communication bandwidth. (The message distribution is done whenever the destinations are different.) When a user A (Alice) wants to send a message M to other users $B_1, \ldots, B_n$, she can, for example, send $[M, H(K_{AB_1}, M), \ldots, H(K_{AB_n}, M)]$ or $[M, E(K_{AB_1}, (K \parallel H(K, M))), \ldots, E(K_{AB_n}, (K \parallel H(K, M)))]$, where $H()$ is a keyed hash function, $E()$ is an encryption algorithm, K_{AB_i} is the secret key shared between A and B_i $(i = 1, \ldots, n)$, and K is a fixed length session key chosen by A.

In the first case, B_i $(1 \leq i \leq n)$ can evaluate the authenticity of the message by recomputing $H(K_{AB_i}, M)$. Disadvantage of this technique is that, A should compute $H()$, n times for n users. However, it is straight forward and uses only a keyed hash function.

In the second case, B_i $(1 \leq i \leq n)$ can decrypt $E(K_{AB_i}, (K \parallel H(K, M)))$, using the secret key K_{AB_i}, and find the session key K. The evaluation of the authenticity can be done now, by recomputing $H(K, M)$. Note that, $(K \parallel H(K, M))$ has a fixed length and will be computed once for all users. This technique is more efficient when the message is very long. Otherwise, the existence of the encryption algorithm $E()$ makes this technique less efficient.

4.1.3 Password Checking

In many systems, to let a user log into the system, user's login-name and password is asked. In the earlier schemes, password was taken as a key to encrypt the login-name, using an encryption algorithm. The ciphertext was kept together with the login-name by the system and was decrypted whenever the password is being validated by the system. In the latter schemes, it was suggested to verify the authenticity by recomputing and not decrypting the ciphertext.

This technique does not need decryption and can be implemented by use of a secure one-way algorithm. Keyed hash functions are examples of secure one-way algorithms and can be used to compute the encrypted part of the password table. Since keyed hash functions are, in general, faster than encryption algorithms, validation of the password entry will be done faster.

4.1.4 Software Protection

Keyed hash functions can be used to protect a software against modifications (and viruses). The protection can be provided by computing the digest of the file (software) using a keyed hash function and saving in a safe place.

In the case that some access protected memory or disk is reserved for the system and users cannot interrupt the system, any change in the file can be realized by the system, if the secret key is known by the system.

If no space is reserved for the system, the protection should be provided and verified by the owner of the file (or an authorized user).

4.1.5 Encryption Algorithms

Hash functions can be used to construct encryption algorithms. In keyed hash functions, since a secret key contributes to the process of hashing, design of encryption algorithms becomes even simpler. An example clarifies the idea.

Suppose $H()$ is a secure keyed hash function which produces r-bit message digests. To encrypt a plaintext $P = (P_1 \mid\mid P_2)$, where P_1 and P_2 are r-bit blocks, the following steps are followed:

1. $T = H(K_1, P_1) \oplus P_2$,
2. $C_1 = H(K_2, T) \oplus P_1$,
3. $C_2 = H(K_3, C_1) \oplus T$,

where $K = (K_1 \mid\mid K_2 \mid\mid K_3)$ is the secret key and C_1 and C_2 are each r bits. The cipher text is $C = (C_1 \mid\mid C_2)$ and can be decrypted by a similar method:

1. $T = H(K_3, C_1) \oplus C_2$,
2. $P_1 = H(K_2, T) \oplus C_1$,
3. $P_2 = H(K_1, P_1) \oplus T$.

A similar scheme was originally proposed in [53] and was briefly examined in [3]. Their scheme uses a one-way function and adds a secret key as a part of the input message.

4.2 Requirements for Keyed Hash Functions

We consider the following properties for keyed hash functions.

- *Security Requirements:*
  1. It should satisfy the requirements of Definition 3.
  2. It should use a secret key of at least 128 bits to thwart exhaustive key search.
  3. It should produce a message digest with at least 128 bits to thwart birthday attacks.
  4. It should uniformly distribute the message digest in the message digest space. This thwarts statistical attacks.
- *Design Heuristics:*
  1. It should use the secret key many times in the hashing process — especially at the beginning and the end.
  2. It should use every bit of the message several times in the hashing process (redundancy of the message).
  3. The underlying round function should be analyzed very carefully to thwart the attacks that are based on the weaknesses of the round function, such as fixed point attack. It needs special care when the design is based on an existing algorithm, such as an encryption algorithm or a hash function. (Sometimes, it is preferred to construct a keyed hash function from scratch to avoid these problems.)

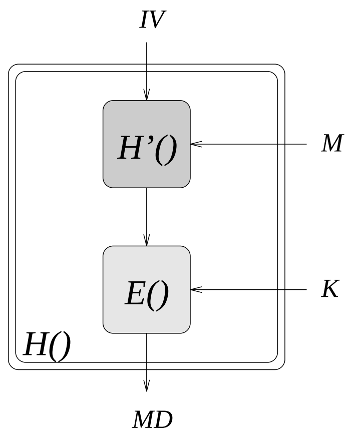

Figure 2: *Hash-then-encrypt construction for keyed hash function $H()$, where $H'()$ is a hash function, $E()$ is an encryption algorithm, IV is the initial vector, M is the message, K is the key, and MD is the message digest.*

- *Operational Requirements:*
  1. The secret key must remain secret among the trusted parties. Usage of a symmetric key cannot guarantee the security of the algorithm against attacks from the trusted users.
  2. If the keyed hash function is used in a protocol, the security of the algorithm should be checked in the new environment. In some cases, other security parameters should be included in the higher level structures. Use of time stamps and serial numbers are examples of such parameters. Use of different keys for different purposes (eg. authentication and/or secrecy) is another example.

Other general requirements for keyed hash functions are *Complementation Freedom, Addition Freedom, Multiplication Freedom, Correlation Freedom.* The reader is referred to [3] for more details about these general requirements.

4.3 Construction of Keyed Hash Functions

In Section 1.3, different types of hash functions were discussed and examples of the existing hash functions for each type were noted. Keyed hash functions can be easily constructed from encryption algorithms, as they already have a secret key. However, an encryption algorithm might not satisfy the security requirements of keyed hash functions. In this section, construction of keyed hash functions from existing hash functions, block ciphers, and also from the scratch are examined.

4.3.1 Construction from Existing Hash Functions

In this section, we show how a pre-existing hash function can be used to construct a keyed hash function. This implies that, some security requirements of the designed keyed hash function relies on the security of the underlying hash function. However, other security requirements for keyed hash functions must be examined separately. We will consider *Hash-then-Encrypt Construction, Nested Hash Function Construction,* and *Construction Using a Key as Part of Message or Initial Vector.*

Hash-then-Encrypt Construction This method is the simplest way of constructing a keyed hash function. The collision freeness depends on a collision free hash function $H'()$ and the protection against active spoofer depends on an encryption algorithm $E()$. The keyed hash function is defined as,

$$H(K, M) = E(K, H'(M)),$$

where K is the secret key and M is the message. Figure 2 illustrates this construction.

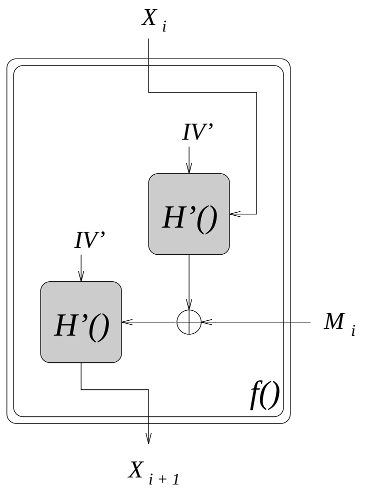

Figure 3: *Nested hash function construction for the round function $f()$ in keyed hash function $H()$, where $H'()$ is a hash function (with IV' as its initial vector), M_i is the i^{th} message block, and X_i is the output of the previous round. Note that, $X_1 = IV$ is the initial vector of the keyed hash function $H()$, which is set to the key K.*

Note that, the length of $H'(M)$ should be equal to the required length of the input for $E()$. This scheme is proposed in the CCITT X.509 standard as a standard scheme ([22]). We note that, the first property of Definition 3 slightly suffers from the slow speed of the encryption algorithm.

Nested Hash Function Construction There are many ways to construct a keyed hash function from nested hash functions. In this section, three schemes are examined and their advantages and disadvantages are explained.

1. The first scheme is as follows. Suppose $H'()$ is a one-way hash function with r-bit message digest. The round function is defined as:

$$f(X_i, M_i) = H'(H'(X_i) \oplus M_i),$$

where X_i is the chaining variable (output of the last round) and M_i is the i^{th} message block (r bits). The chaining variable X_{i+1} will be set to $f(X_i, M_i)$ at the end of each round, where $X_1 = IV$. The keyed hash function uses the above round function with the same design principals that was discussed earlier. It can be summarized as:

$$H(K, M) = f(f(\cdots f(f(K, M_1), M_2), \ldots, M_{n-1}), M_n),$$

where K is the key ($K = X_1 = IV$) and M is the message, which is divided into n r-bit blocks $(M_1, \ldots, M_n)$. Figure 3 illustrates the round function $f()$.

It can be easily tested that $H()$ satisfies all requirements of Definition 3 [10]. Advantage of this scheme is its dependency on only one one-way hash function. Disadvantage of this scheme is the overhead in the calculation of $H'()$ (twice for each message block M_i, in every round). The plain use of this scheme suffers from padding attack — an arbitrary message M' can be appended to M to generate a genuine pair $[(M \parallel M'), H(K, (M \parallel M'))]$. However, there are some easy methods to thwart this attack (cf. [6]).

2. The second scheme is similar to the previous one, and the round function $f()$ is defined as:

$$f(X_i, M_i) = H'(H'(X_i) \parallel M_i).$$

Figure 3 can be used to illustrate this method. The only difference is that, the '$\oplus$' operation should be changed to the concatenation operation. Similar to the first scheme, the keyed hash function can be summarized as:

$$H(K, M) = f(f(\cdots f(f(K, M_1), M_2), \ldots, M_{n-1}), M_n).$$

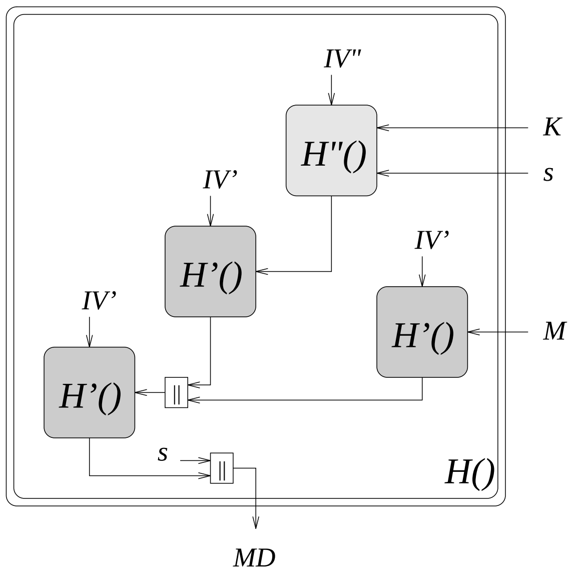

Figure 4: *Nested hash function construction for the modified keyed hash function $H()$, where $H''()$ is a keyed hash function (with IV'' as its initial vector), $H'()$ is a hash function (with IV' as its initial vector), M is the message, K is the key, s is a suitable salt value, and '$||$' denotes concatenation.*

In this scheme, the message blocks, M_i's, can be of any fixed size, because, it is concatenated to $H'(X_i)$ rather than being XORed. This change in the round function adds one advantage: *it is hard to find K, M, K', and M' with $K \neq K'$ and $M \neq M'$ such that $f(K, M) = f(K', M')$.* That is true because, the XOR operation is replaced by concatenation. However, it is not clear that the fourth property of Definition 3 is satisfied. This scheme still has the overhead problem (twice use of $H'()$) in the process of the round function and also suffers from padding attack.

3. In the previous method, it was not clear that the fourth property of Definition 3 is satisfied. The third scheme is the combination of the previous two methods and satisfies the fourth property. Using the same assumptions that were mentioned earlier, the keyed hash function with additional property can be defined as:

$$H(K, M) = s \mid\mid H'(H'(H''(K, s)) \mid\mid H'(M)),$$

where $H''()$ is a keyed hash function and s is a suitable *salt* value. (Berson *et al.* [10] have used a salt value to make dictionary attacks more difficult.) Figure 4 illustrates this modified keyed hash function.

For example, if $H''()$ is the keyed hash function that was constructed in the first scheme, this method can be described as:

$$H(K, M) = s \mid\mid H'(H'(f(K, s)) \mid\mid H'(M)),$$

with $f(K, s) = H'(H'(K) \oplus s)$, or simply:

$$H(K, M) = s \mid\mid H'(H'(H'(H'(K) \oplus s)) \mid\mid H'(M)).$$

Note that, the message is processed only in the last use of $H'()$. Therefore, the scheme can be summarized as:

$$H(K, M) = s \mid\mid H'(K' \mid\mid H'(M)),$$

where K' is the secret part and depends on s.

Construction Using a Key as Part of Message or Initial Vector In this section, the construction of keyed hash functions from a pre-existing hash function, by having a key as parts of message or initial vector, is studied, and therefore, the speed of the constructed keyed hash function remains almost the same as that of the hash function.

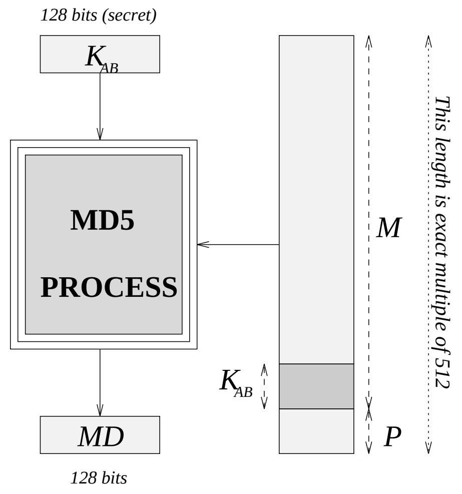

Figure 5: *Construction of a keyed hash function using MD5, where K_{AB} is the secret key, M is the message, P is the padding bits for MD5, and MD is the message digest. In this construction, the secret key (128 bits) is used as both the initial vector and a suffix to the message.*

In the following, several schemes are given, where $H'()$ is a collision free (and pseudo-collision free) hash function, $K = (K_1 \parallel K_2)$ is the secret key, '$\parallel$' denotes concatenation, and the notations '$\overline{\oplus}$' and '$\underline{\oplus}$' are especial XOR operations (cf. Section 1.1).

1. $H(K, M) = H'(IV, (K \parallel M))$, where IV is a fixed initial vector.
2. $H(K, M) = H'(IV, (M \parallel K))$, where IV is a fixed initial vector.
3. $H(K, M) = H'(IV, (K_1 \parallel M \parallel K_2))$, where IV is a fixed initial vector.
4. $H(K, M) = H'(IV, (K \parallel M \parallel K))$, where IV is a fixed initial vector.
5. $H(K, M) = H'(IV, (K \overline{\oplus} M))$, where IV is a fixed initial vector.
6. $H(K, M) = H'(IV, (M \underline{\oplus} K))$, where IV is a fixed initial vector.
7. $H(K, M) = H'(IV, (K_1 \overline{\oplus} M \underline{\oplus} K_2))$, where IV is a fixed initial vector.
8. $H(K, M) = H'(IV, (K \overline{\oplus} M \underline{\oplus} K))$, where IV is a fixed initial vector.
9. $H(K, M) = H'(IV, M)$, where $IV = K$.
10. $H(K, M) = H'(IV, (M \underline{\oplus} K))$, where $IV = K$.
11. $H(K, M) = H'(IV, (M \underline{\oplus} K_2))$, where $IV = K_1$.

The first three methods were suggested by Tsudik in [75], where K has 512-bit length. Other methods are improved versions of the first three ones that are studied in [6], and use only 128-bit keys. Methods 1, 5, and 9 suffer from padding attack. However, very simple solutions are suggested in [6]. The tenth method is considered as the best method. This method is safe against the known attacks and at the same time is efficient. Figure 5 is an example of this method, where MD5 is used as the underlying hash function (it is supposed that MD5 is collision free). The figure shows that the only operation added to the normal process of MD5 is the XOR operation on 128 bits, and hence, it is fast. On the other hand, the message is surrounded by the secret key and this thwarts any possible attack.

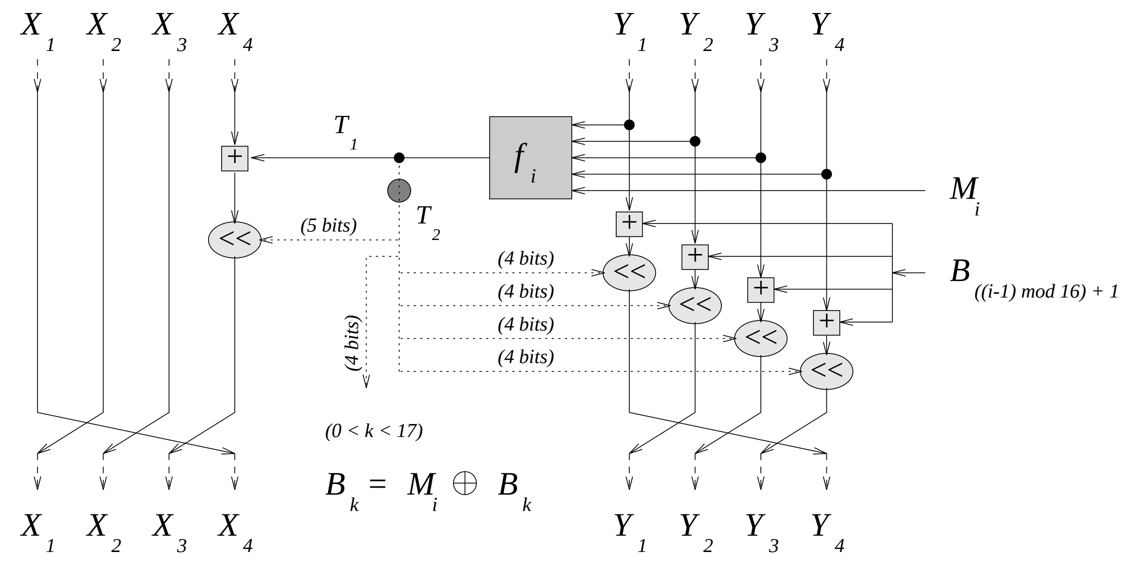

Figure 6: *One round of KHF.*

4.3.2 Construction from Block Ciphers

Since a block cipher uses a secret key to provide the secrecy, it can be used as the round function of a keyed hash function. The problem rises when the block cipher does not satisfy the security requirements of the keyed hash function. Also, most of the known block ciphers work on 64-bit plaintexts and ciphertexts and can provide 64-bit hash values. This is not desired for the hash functions (in Section 3, it was suggested to have about 128 bits as the digest length). A complete study on hash functions based on block ciphers are in [63, 81].

4.3.3 Construction from Scratch

In the previous two sections, constructions of keyed hash functions from hash functions and block ciphers were examined. Existence of a secret key makes the analysis of an algorithm more difficult for the outsiders. Therefore, instead of using a pre-existing hash function a new algorithm with the best performance can be designed. The authors have designed a new keyed hash function (KHF), which is still under the examination. It uses the following functions as the underlying round functions.

$$f_i(A,B,C,D,E) = \begin{cases} (A \textbf{ and } E) \textbf{ xor } (B \textbf{ and } C) \textbf{ xor } ((B \textbf{ xor } C) \textbf{ and } D), & i = 0 \textbf{ mod } 5 \\ A \textbf{ xor } (B \textbf{ and } (A \textbf{ xor } D)) \textbf{ xor } (((A \textbf{ and } D) \textbf{ xor } C) \textbf{ and } E), & i = 1 \textbf{ mod } 5 \\ A \textbf{ xor } (C \textbf{ and } D \textbf{ and } E) \textbf{ xor } ((A \textbf{ and } C) \textbf{ or } (B \textbf{ and } D)), & i = 2 \textbf{ mod } 5 \\ B \textbf{ xor } ((D \textbf{ and } E) \textbf{ or } (A \textbf{ and } C)), & i = 3 \textbf{ mod } 5 \\ D \textbf{ xor } E \textbf{ xor } (((D \textbf{ and } E) \textbf{ xor } A) \textbf{ and not}(B \textbf{ and } C)), & i = 4 \textbf{ mod } 5 \end{cases}$$

They are functions of five variables and have the following important properties [27].

- They are 0-1 balanced.
- They satisfy the Strict Avalanche Criterion (SAC).
- They are highly non-linear.
- They are pairwise linearly non-equivalent.
- They are far from each other, in terms of the Hamming distance.
- They can be shown by short Algebraic Normal Forms (ANF).

In KHF, four possibilities for the 128-bit buffers K_X, K_Y, and K_Z are suggested.

1. K_X, K_Y, and K_Z are all secret keys with $K_X = K_Y = K_Z$ (total key bits = 128).
2. K_X is a fixed (public) initial vector and K_Y and K_Z are secret keys with $K_Y = K_Z$ (total key bits = 128).
3. K_X is a fixed (public) initial vector and K_Y and K_Z are different secret keys (total key bits = 256).
4. K_X, K_Y, and K_Z are all secret keys with $K_X = K_Z$ (total key bits = 256).

Figure 6 illustrates one round of KHF.

5 Open Problems

In the study of hash functions, there are many open problems and difficulties. In this section, a set of typical open problems are given:

1. Is it possible to construct an efficient (fast) and provably secure keyed hash function (or hash function, in general)? In other words, is there an optimizes keyed hash function?
2. Is it possible to construct an efficient (fast) keyed hash function that can provide public and secret key (not based on public key cryptosystems)? In other words, Is it possible to construct a digital signature scheme, using only keyed hash functions? (We call it *Public Keyed Hash Function.*)
3. Is it possible to construct an efficient and secure encryption algorithm based on keyed hash functions?

References

[1] Accredited Standards Committee X3. *American National Standard X3.92: Data Encryption Algorithm (DEA)*, 1981.

[2] Accredited Standards Committee X9. *American National Standard X9.9: Financial Institution Message Authentication*, 1982.

[3] R. Anderson. The Classification of Hash Functions. In *Proceedings of the IMA Conference in Cryptography and Coding*, 1993.

[4] S. Bakhtiari and R. Safavi-Naini. Application of PVM to Linear Cryptanalysis of DES. Technical Report 94-23, Department of Computer Science, University of Wollongong, October 1994.

[5] S. Bakhtiari and R. Safavi-Naini. Application of PVM to Linear Cryptanalysis of DES. In *Proceedings of the Australian Parallel Computing and Transputers (PCAT)*, pages 278–279, November 1994.

[6] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Practical and Secure Message Authentication. In *Series of Annual Workshop on Selected Areas in Cryptography (SAC)*, pages 55–68, May 1995.

[7] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Practical Message Authentication Schemes. Technical Report 95-04, Department of Computer Science, University of Wollongong, May 1995.

[8] T. Baritaud, H. Gilbert, and M. Girault. FFT Hashing is not Collision-Free. In *Advances in Cryptology — Eurocrypt '92*, pages 35–44, 1993.

[9] T. A. Berson. Differential Cryptanalysis Mod 2^{32} with Applications to MD5. In *Advances in Cryptology, Proceedings of EUROCRYPT '92*, pages 71–80, 1992.

[10] T. A. Berson, L. Gong, and T. M. A. Lomas. Secure, Keyed, and Collisionful Hash Functions. Technical Report (included in) SRI-CSL-94-08, SRI International Laboratory, Menlo Park, California, December 1993. Last revised version (September 2, 1994).

[11] E. Biham. On the Applicability of Differential Cryptanalysis to Hash Functions. In *E.I.S.S Workshop on Cryptographic Hash Functions*, pages 25–27, March 1992.

[12] E. Biham. Cryptanalysis of Multiple Mode of Operation. In *Advances in Cryptology, Proceedings of ASIACRYPT '94*, November 1994.

[13] E. Biham. On Matsui's Linear Cryptanalysis. In *Advances in Cryptology, Proceedings of EUROCRYPT '94*, pages 349–361, 1994.

[14] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[15] E. Biham and A. Shamir. Differential cryptanalysis of FEAL and N-Hash. In *Advances in Cryptology — Eurocrypt '91*, pages 1–16, 1991.

[16] E. Biham and A. Shamir. Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer. In *Advances in Cryptology, Proceedings of CRYPTO '91*, pages 156–171, 1992.

[17] E. Biham and A. Shamir. Differential Cryptanalysis of the Full 16-Round DES. In *Advances in Cryptology, Proceedings of CRYPTO '92*, pages 487–496, 1992.

[18] B. Boer and A. Bosselaers. An Attack on the Last Two Rounds of MD4. In *Advances in Cryptology, Proceedings of CRYPTO '91*, pages 194–203, 1991.

[19] B. Boer and A. Bosselaers. Collisions for the Compression Function of MD5. In *Advances in Cryptology, Proceedings of EUROCRYPT '93*, pages 293–304, 1994.

[20] L. Brown, M. Kwan, J. Pieprzyk, and J. Seberry. Improving Resistance to Differential Cryptanalysis of LOKI. In *Advances in Cryptology, Proceedings of ASIACRYPT '91*, pages 3–50, 1991.

[21] L. Brown, J. Pieprzyl, and J. Seberry. LOKI: A Cryptographic Primitive for Authentication and Secrecy Applications. In *Advances in Cryptology — Auscrypt '90*, pages 229–236, 1990.

[22] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. Technical Report 39, Digital Systems Research Center, Palo Alto, California, February 1989.

[23] P. Camion and J. Patarin. The Knapsck Hash Function Proposed at Crypto '89 Can be Broken. In *Advances in Cryptology, Proceedings of EUROCRYPT '91*, pages 39–53, 1991.

[24] J. L. Carter and M. N. Wegman. Universal Class of Hash Functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[25] J. L. Carter and M. N. Wegman. New Hash Functions and Their Use in Authentication and Set Equality. *Journal of Computer and System Sciences*, 22:265–279, 1981.

[26] F. Chabaud and S. Vaudenay. Links Between Differential and Linear Cryptanalysis. In *Advances in Cryptology, Proceedings of EUROCRYPT '94*, pages 363–374, 1994.

[27] C. Charnes and J. Pieprzyk. Linear Nonequivalence versus Nonlinearity. In *Advances in Cryptology, Proceedings of AUSCRYPT '92*, pages 156–164, December 1992.

[28] F. Cohen. The ASP Integrity Toolkit, Ver 3.5. ASP Press, Pittsburgh (PA), 1991.

[29] D. Coppersmith. Analysis of ISO/CCITT Document X.509 Annex D. Internal Memo, IBM T.J. Watson Center, June 11, 1989.

[30] J. Daemen, A. Bosselaers, R. Govaerts, and J. Vendewalle. Collisions for Schnorr's Hash Function FFT-Hash Presented at CRYPTO '91. In *Advances in Cryptology, Proceedings of ASIACRYPT '91*, pages 477–480, 1991.

[31] J. Daemen, R. Govaerts, and J. Vandewalle. A Framework for the Design of One-Way Hash Functions Including Cryptanalysis of Damgård's One-Way Function Based on Cellular Autpmaton. In *Advances in Cryptology, Proceedings of ASIACRYPT '91*, pages 82–98, 1991.

[32] J. Daemen, R. Govaerts, and J. Vendewalle. A Hardware Design Model for Cryptographic Algorithms. In *esorics92*, pages 419–434, 1992.

[33] I. B. Damgård. Collision Free Hash Functions and Public Key Signature Schemes. In *Advances in Cryptology, Proceedings of EUROCRYPT '87*, pages 203–216, 1987.

[34] I. B. Damgård. *The Application of Claw Free Functions in Cryptography.* PhD thesis, Aarhus University, Mathematics Institute, 1988.

[35] I. B. Damgård. A Design Principle for Hash Functions. In *Advances in Cryptology, Proceedings of CRYPTO '89*, pages 416–427, October 1989.

[36] I. B. Damgård. A Design Principle for Hash Functions. In *Advances in Cryptology, Proceedings of CRYPTO '89*, pages 416–427, 1990.

[37] Y. Desmedt. Unconditionally Secure Authentication Schemes and Practical and Theoretical Consequences. In *Advances in Cryptology, Proceedings of CRYPTO '85*, pages 42–55, 1986.

[38] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.

[39] P. Godlewski and P. Camion. Manipulation and Errors, Detections and Localization. In *Advances in Cryptology, Proceedings of EUROCRYPT '88*, pages 97–10, 1988.

[40] L. Gong. A New Construction of Collisionful Keyed One-Way Hash Functions. Technical Report (included in) SRI-CSL-TR-94-14, SRI International Laboratory, Menlo Park, California, September 1994.

[41] S. Harari. Non-Linear, Non-Commutative Functions for Data Integrity. In *Advances in Cryptology, Proceedings of EUROCRYPT '84*, pages 25–32, 1985.

[42] M. E. Hellman, R. Merkle, R. Schroeppel, L. Washington, W. Diffie, S. Pohlig, and P. Schweitzer. Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard. Technical Report SEL 76–042, Stanford University, 1976.

[43] H. M. Heys and S. E. Tavares. The Design of Substitution-Permutation Networks Resistant to Diffrential and Linear Cryptanalysis. *ACM Conference on Computer and Communication Security*, November 1994.

[44] ISO/IEC. *Information Technology - Data Cryptograpgy Techniques - Modes of Operation for a 64-bit Block Cipher Algorithm*, 1987. IS 8372, ISO/IEC.

[45] ISO/IEC. *Information Technology - Data Cryptograpgy Techniques - Data Integrity Mechanism Using a Cryptographic Check Function Employing a Block Cipher Algorithm*, 1989. IS 9797, ISO/IEC.

[46] B. S. Kaliski Jr. and M. J. B. Robshaw. On "Pseudocollisions" in the MD5 Message-Digest Algorithm, April 1993. RSA Laboratories, 100 Marine Parkway, Redwood City, CA 94065.

[47] B. S. Kaliski. The MD2 Message-Digest Algorithm. RFC 1319, April 1992. Network Working Group, RSA Laboratories.

[48] M. Kwan and J. Pieprzyk. A General Purpose Technique dor Locating Key Scheduling Weaknesses in DES-Like Cryptosystems. In *Advances in Cryptology, Proceedings of ASIACRYPT '91*, pages 237–346, 1991.

[49] X. Lai. *On the Design and Security of Block Ciphers.* Konstanz, Hartung-Gorre, Germany, 1992.

[50] X. Lai and J. Massey. A Proposal for a New Block Encryption Standard. In *Advances in Cryptology, Proceedings of EUROCRYPT '90*, pages 389–404, 1990.

[51] X. Lai and J. Massey. Markov Ciphers and Differential Cryptanalysis. In *Advances in Cryptology, Proceedings of EUROCRYPT '91*, pages 17–38, 1991.

[52] X. Lai, R. A. Rueppel, and J. Woollven. A Fast Cryptographic Checksum Algorithm Based on Stream Ciphers. In *Advances in Cryptology — Auscrypt '92*, pages 339–348, 1993.

[53] M. Luby and C. Rackoff. How to Construct Pseudorandom Permutations and Pseudorandom Functions. *SIAM J. Computing*, 17(2):373–386, April 1988.

[54] M. Matsui. Linear Cryptanalysis of DES Cipher (I), 1993. Version 1.03, Computer and Information Systems Laboratory, Mitsubishi Electric Corporation 5-1-1, Ofuna, Kamakura, Kanagawa, 247, Japan.

[55] R. Merkle. A Fast Software One-Way Hash Function. *Journal of Cryptology*, 3(1):43–58, 1990.

[56] R. Merkle and M. Hellman. Hiding Information and Signatures in Trapdoor Knapsacks. *IEEE Trans. Inform. Theory*, IT-24:525–530, September 1978.

[57] S. Miyaguchi, K. Ohta, and M. Iwata. 128-Bit Hash Function (N-Hash). In *SECURICOM '90*, pages 123–137, 1990.

[58] J. H. Moore and G. J. Simmons. Cycle Structures of the DES with Weak and Semi-Weak Keys. In *Advances in Cryptology, Proceedings of CRYPTO '86*, pages 9–32, 1987.

[59] National Institute of Standards and Technology (NIST). *FIPS Publication 180: Secure Hash Standard (SHS)*, May 11, 1993.

[60] National Bureau of Standards. Announcing the Data Encryption Standard. Technical Report FIPS Publication 46, National Bureau of Standards, January 1977.

[61] K. Ohta and K. Koyama. Meet-in-the-Middle Attack on Digital Signature Schemes. In *Abstract of AUSCRYPT '90*, pages 110–121, 1990.

[62] J. Pieprzyk and B. Sadeghiyan. *Design of Hashing Algorithms*. Springer-Verlag, 1993.

[63] B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke University Leuven, January 1993.

[64] B. Preneel, A. Bosselaers, R. Govaerts, and J. Vandewalle. A Chosen Text Attack on the Modified Cryptographic Checksum Algorithm of Cohen and Huang. In *Advances in Cryptology, Proceedings of CRYPTO '89*, pages 154–163, 1990.

[65] B. Preneel, D. Chaum, W. Fumy, C. J. A. Jansen, P. Landrock, and G. Roelofsen. Race Integrity Primitives Evaluation (RIPE): A Status Report. In *Advances in Cryptology, Proceedings of EUROCRYPT '91*, pages 547–551, 1991.

[66] J.-J. Quisquater and J.-P. Delescaille. How Easy is Collision Search? Application to DES. In *Advances in Cryptology, Proceedings of EUROCRYPT '89*, pages 429–435, 1989.

[67] J.-J. Quisquater and J.-P. Delescaille. How Easy is Collision Search. New results and applications to DES. In *Advances in Cryptology, Proceedings of CRYPTO '89*, pages 408–415, 1990.

[68] RIPE Consortium. *Final Report of RACE 1040*. Centrum voor Wiskunde en Informatica, April 1993. Report CS-R9324.

[69] R. L. Rivest. The MD4 Message-Digest Algorithm. RFC 1320, April 1992. Network Working Group, MIT Laboratory for Computer Science and RSA Data Security, Inc.

[70] R. L. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, April 1992. Network Working Group, MIT Laboratory for Computer Science and RSA Data Security, Inc.

[71] R. L. Rivest, A. Shamir, and L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[72] N. Rogier and P. Chauvaud. The Compression Function of MD2 is not Collision Free. In *Series of Annual Workshop on Selected Areas in Cryptography (SAC)*, pages 69–83, May 1995.

[73] J. Seberry and J. Pieprzyk. *Cryptography: An Introduction to Computer Security*. Prentice Hall, 1989.

[74] J. D. Touch. Performance Analysis of MD5. In *ACM Special Interest Group in Communication (SIGCOMM)*, 1995. To apper.

[75] G. Tsudik. Message Authentication with One-Way Hash Functions. *IEEE INFOCOM*, May 1992.

[76] J. Vandewalle, D. Chaum, W. Fumy, C. Jansen, P. Landrock, and G. Roelofsen. A European call for cryptographic algorithms: RIPE; Race Integrity Primitives Evaluation. In *Advances in Cryptology, Proceedings of EUROCRYPT '89*, pages 267–272, 1989.

[77] S. Vaudenay. FFT-Hash-II is not Yet Collision-Free. In *Advances in Cryptology, Proceedings of CRYPTO '92*, pages 587–593, 1992.

[78] S. Vaudenay. On the need for Multipermutations: Cryptanalysis of MD4 and SAFER. In *Proceedings of the Leuven Workshop on Cryptographic Algorithms*, pages 195–206, 1994.

[79] S. Wolfram. Random Sequence Generation by Cellular Automata. *Advances in Applied Mathematics*, 7:123–169, 1986.

[80] G. Zémor. Hash Functions and Graphs with Large Girths. In *Advances in Cryptology, Proceedings of EUROCRYPT '91*, pages 508–511, 1991.

[81] Y. Zheng. *Principles for Designing Secure Block Ciphers and One-Way Hash Functions.* PhD thesis, Electrical and Computer Engineering, Yokohama National University, December 1990.

[82] Y. Zheng, T. Hardjono, and J. Pieprzyk. The Sibling Intractable Function Family (SIFF): Notion, Construction and Applications. *IEICE Trans. Fundamentals*, E76-A(1), January 1993.

[83] Y. Zheng, J. Pieprzyk, and J. Seberry. HAVAL - A One-Way Hashing Algorithm with Variable Length of Output. In *Advances in Cryptology, Proceedings of AUSCRYPT '92*, pages 83–104, December 1992.

A Description of some Dedicated Hash Functions

In this appendix, a brief description of some dedicated hash functions are given.

A.1 MD2

MD2 (Message Digest 2) can be described as follows [47].

1. *Appending the Padding Bytes:* 'x' bytes of value 'x' are appended to the arbitrary length message M so that the length (in bytes) of the padded message becomes congruent to zero, modulo 16. Let $M_{i,j}$ denotes the j^{th} byte of the i^{th} block of the padded message, where $1 \leq j \leq 16$, $1 \leq i \leq n$, and n is the number of 16-byte blocks of the padded message.

2. *Appending the Checksum:* The following algorithm calculates a 16-byte checksum $(C_1, \ldots, C_{16})$ for the message. It uses a 256-byte string as a random permutation, where S_i denotes the i^{th} element (byte) of that string.

   - Do $C_i = 0$, for $i = 1, \ldots, 16$.
   - Set $L = 0$.
   - Do $L = C_j = S_{L \oplus M_{i,j}}$, for $j = 1, \ldots, 16$, $i = 1, \ldots, n$.

   The 16 bytes $C_1, \ldots, C_{16}$ are the checksum and should be appended to the padded message. (The message length is now $(n+1) \times 16$ bytes.)

3. *Initialization:* A 48-byte buffer $IV = (X_1, \ldots, X_{48})$, which is used to keep the message digest, is initialized to zero.

4. *Processing the Message in 16-byte Blocks:*
   Do the followings, for $i = 1, \ldots, n+1$.

- Do $X_{j+16} = M_{i,j}$, $X_{j+32} = M_{i,j} \oplus X_j$, for $j = 1, \ldots, 16$.
- Set $T = 0$.
- Do the followings, for $j = 1, \ldots, 18$.
  - Do $T = X_k = X_k \oplus S_T$, for $k = 1, \ldots, 48$.
  - Set $T = (T + j) \textbf{ mod } 256$.

5. *Message Digest:* The message digest is the 16-byte output string $MD = (X_1, \ldots, X_{16})$.

A.2 MD4

MD4 (Message Digest 4) is designed for 32-bit operation machines. In the process of MD4, the following notations and functions are used:

$$f_i(X,Y,Z) = \begin{cases} (X \textbf{ and } Y) \textbf{ or } (\textbf{not}(X) \textbf{ and } Z), & i = 1, \ldots, 16. \\ (X \textbf{ and } Y) \textbf{ or } (X \textbf{ and } Z) \textbf{ or } (Y \textbf{ and } Z), & i = 17, \ldots, 32. \\ X \textbf{ xor } Y \textbf{ xor } Z, & i = 33, \ldots, 48. \end{cases}$$

$$t_i \overset{(in\ hex.)}{=\!=\!=\!=\!=\!=} \begin{cases} \texttt{00000000}, & i = 1, \ldots, 16. \\ \texttt{5a827999}, & i = 17, \ldots, 32. \\ \texttt{6ed9eba1}, & i = 33, \ldots, 48. \end{cases}$$

$$r_i = \begin{cases} i, & i = 1, \ldots, 16. \\ 4 \times ((i-13) \textbf{ mod } 4) + ((i-13) \textbf{ div } 4), & i = 17, \ldots, 32. \\ ((2 \times i - 66) \textbf{ mod } 15) + 1, & i = 33, 35, 38, 40, 41, 43, 46. \\ ((2 \times i - 66) \textbf{ mod } 15) + 7, & i = 34, 36, 42, 44. \\ (((2 \times i - 66) \textbf{ mod } 15) + 7) \textbf{ mod } 12, & i = 37, 39, 45, 47. \\ 16 & i = 48. \end{cases}$$

$$s_i = \begin{cases} \texttt{3}, & i = 1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45. \\ \texttt{5}, & i = 18, 22, 26, 30. \\ \texttt{7}, & i = 2, 6, 10, 14. \\ \texttt{9}, & i = 19, 23, 27, 31, 34, 38, 42, 46. \\ \texttt{11}, & i = 3, 7, 11, 15, 35, 39, 43, 47. \\ \texttt{13}, & i = 20, 24, 28, 32. \\ \texttt{15}, & i = 36, 40, 44, 48. \\ \texttt{19}, & i = 4, 8, 12, 16. \end{cases}$$

$$add(X, Y, \ldots) = (X + Y + \cdots) \textbf{ mod } 2^{32}.$$

$$rol(X, s) = \text{ Rotate left } X \text{ by } s \text{ bits.}$$

Now MD4 can be summarized as:

1. *Appending the Padding Bits and the Length:* The arbitrary length message M is padded by a single '1' bit followed by enough '0' bits so that the length of the padded message (in bits) is congruent to 448, modulo 512. Length of M (only the least significant 64 bits of the length) is then appended so that the final length becomes a multiple of 512 (in bits). Let $M_{i,j}$ denote the j^{th} word of the i^{th} block of the padded message, where each word has 32 bits, each block has 16 words ($1 \le j \le 16$), and n is the number of 512-bit blocks in the padded message ($1 \le i \le n$).

2. *Initialization:* A 4-word buffer $IV = (X_1, X_2, X_3, X_4)$, which is used to keep the 128-bit message digest, is initialized (in hexadecimal) to:

$$X_1 = \texttt{67452301}, \quad X_2 = \texttt{efcdab89}, \quad X_3 = \texttt{98badcfe}, \quad X_4 = \texttt{10325476}.$$

3. *Processing the Message in 16-word (512-bit) Blocks:*
   - Set $A = X_1$, $B = X_2$, $C = X_3$, $D = X_4$.

- Do the followings, for $i = 1, \ldots, n$.
  - Do $T = add(A, f_j(B,C,D), M_{i,r_j}, t_j)$, $A = D$, $D = C$, $C = B$, $B = rol(T, s_j)$, for $j = 1, \ldots, 48$.
  - Set $X_1 = add(X_1, A)$, $X_2 = add(X_2, B)$, $X_3 = add(X_3, C)$, $X_4 = add(X_4, D)$.

4. *Message Digest:* The produced message digest is the 4-word (128-bit) output $MD = (X_1, X_2, X_3, X_4)$.

A.3 MD5

MD5 (Message Digest 5) is the strengthened version of MD4. It has one more round and uses more operations in each round. The basic notations and functions are defined first.

$$f_i(X,Y,Z) = \begin{cases} (X \textbf{ and } Y) \textbf{ or } (\textbf{not}(X) \textbf{ and } Z), & i = 1, \ldots, 16. \\ (X \textbf{ and } Z) \textbf{ or } (\textbf{not}(Z) \textbf{ and } Y), & i = 17, \ldots, 32. \\ X \textbf{ xor } Y \textbf{ xor } Z, & i = 33, \ldots, 48. \\ (\textbf{not}(Z) \textbf{ or } X) \textbf{ xor } Y, & i = 49, \ldots, 64. \end{cases}$$

$$t_i = \sin(i) \textbf{ mod } 2^{32}, \quad i = 1, \ldots, 64.$$

$$r_i = \begin{cases} i, & i = 1, \ldots, 16. \\ (1 + 5 \times (j - 17)) \textbf{ mod } 16, & i = 17, \ldots, 32. \\ (5 + 3 \times (j - 33)) \textbf{ mod } 16, & i = 33, \ldots, 48. \\ 7 \times (j - 49) \textbf{ mod } 16, & i = 49, \ldots, 64. \end{cases}$$

$$s_i = \begin{cases} \texttt{4}, & i = 33, 37, 41, 45. \\ \texttt{5}, & i = 17, 21, 25, 29. \\ \texttt{6}, & i = 49, 53, 57, 61. \\ \texttt{7}, & i = 1, 5, 9, 13. \\ \texttt{9}, & i = 18, 22, 26, 30. \\ \texttt{10}, & i = 50, 54, 58, 62. \\ \texttt{11}, & i = 34, 38, 42, 46. \\ \texttt{12}, & i = 2, 6, 10, 14. \\ \texttt{14}, & i = 19, 23, 27, 31. \\ \texttt{15}, & i = 51, 55, 59, 63. \\ \texttt{16}, & i = 35, 39, 43, 47. \\ \texttt{17}, & i = 3, 7, 11, 15. \\ \texttt{20}, & i = 20, 24, 28, 32. \\ \texttt{21}, & i = 52, 56, 60, 64. \\ \texttt{22}, & i = 4, 8, 12, 16. \\ \texttt{23}, & i = 36, 40, 44, 48. \end{cases}$$

$$add(X, Y, \ldots) = (X + Y + \cdots) \textbf{ mod } 2^{32}.$$

$$rol(X, s) = \text{ Rotate left } X \text{ by } s \text{ bits.}$$

MD5 can be summarized as:

1. *Appending the Padding Bits and the Length:* The arbitrary length message M is padded by a single '1' bit followed by enough '0' bits so that the length of the padded message (in bits) is congruent to 448, modulo 512. Length of M (only the least significant 64 bits of the length) is then appended so that the final length becomes a multiple of 512 (in bits). Let $M_{i,j}$ denote the j^{th} word of the i^{th} block of the padded message, where each word has 32 bits, each block has 16 words ($1 \leq j \leq 16$), and n is the number of 512-bit blocks in the padded message ($1 \leq i \leq n$).

2. *Initialization:* A 4-word buffer $IV = (X_1, X_2, X_3, X_4)$, which is used to keep the 128-bit message digest, is initialized (in hexadecimal) to:

$$X_1 = \texttt{67452301}, \quad X_2 = \texttt{efcdab89}, \quad X_3 = \texttt{98badcfe}, \quad X_4 = \texttt{10325476}.$$

3. *Processing the Message in 16-word (512-bit) Blocks:*
   - Set $A = X_1,\ B = X_2,\ C = X_3,\ D = X_4$.
   - Do the followings, for $i = 1, \ldots, n$.
     - Do $T = add(A, f_j(B,C,D), M_{i,r_j}, t_j),\ A = D,\ D = C,$ $C = B,\ B = add(B, rol(T, s_j)),$ for $j = 1, \ldots, 64$.
     - Set $X_1 = add(X_1, A),\ X_2 = add(X_2, B),\ X_3 = add(X_3, C),\ X_4 = add(X_4, D)$.
4. *Message Digest:* The computed message digest is the 4-word (128-bit) output $MD = (X_1, X_2, X_3, X_4)$.

A.4 SHA

SHA (Secure Hash Algorithm) is another strengthened version of MD4, where the buffer size is increased from 128 bits to 160 bits and the number of steps per rounds is increased from 16 steps to 20 steps. There are some other small changes that will be mentioned in the following summary. The basic notations and functions for SHA are:

$$f_i(X,Y,Z) = \begin{cases} (X \textbf{ and } Y) \textbf{ or } (\textbf{not}(X) \textbf{ and } Z), & i = 1, \ldots, 20. \\ X \textbf{ xor } Y \textbf{ xor } Z, & i = 21, \ldots, 40. \\ (X \textbf{ and } Y) \textbf{ or } (X \textbf{ and } Z) \textbf{ or } (Y \textbf{ and } Z), & i = 41, \ldots, 60. \\ X \textbf{ xor } Y \textbf{ xor } Z, & i = 61, \ldots, 80. \end{cases}$$

$$t_i \overset{(in\ hex.)}{=\!=\!=\!=\!=\!=} \begin{cases} \texttt{5a827999}, & i = 1, \ldots, 20. \\ \texttt{6ed9eba1}, & i = 21, \ldots, 40. \\ \texttt{8f1bbcdc}, & i = 41, \ldots, 60. \\ \texttt{ca62c1d6}, & i = 61, \ldots, 80. \end{cases}$$

$$m_j(i) = \begin{cases} M_{i,j}, & j = 1, \ldots, 16. \\ M_{i,(j-3)} \oplus M_{i,(j-8)} \oplus M_{i,(j-14)} \oplus M_{i,(j-16)}, & j = 17, \ldots, 80. \end{cases}$$

$$add(X, Y, \ldots) = (X + Y + \cdots) \textbf{ mod } 2^{32}.$$

$$rol(X, s) = \text{ Rotate left } X \text{ by } s \text{ bits.}$$

Now SHA process can be summarized as:

1. *Appending the Padding Bits and the Length:* The arbitrary length message M is padded by a single '1' bit followed by enough '0' bits so that the length of the padded message (in bits) is congruent to 448, modulo 512. Length of M (only the least significant 64 bits of the length) is then appended so that the final length becomes a multiple of 512 (in bits). Let $M_{i,j}$ denote the j^{th} word of the i^{th} block of the padded message, where each word has 32 bits, each block has 16 words $(1 \le j \le 16)$, and n is the number of 512-bit blocks in the padded message $(1 \le i \le n)$.
2. *Initialization:* A 5-word buffer $IV = (X_1, X_2, X_3, X_4, X_5)$, which is used to keep the 160-bit message digest, is initialized (in hexadecimal) to:

   $$X_1 = \texttt{67452301},\ X_2 = \texttt{efcdab89},\ X_3 = \texttt{98badcfe},\ X_4 = \texttt{10325476},\ X_5 = \texttt{c3d2e1f0}.$$
3. *Processing the Message in 20-word (640-bit) Blocks:*
   - Set $A = X_1,\ B = X_2,\ C = X_3,\ D = X_4,\ E = X_5$.
   - Do the followings, for $i = 1, \ldots, n$.
     - Do $M_{i,j} = m_j(i),\ T = add(rol(A, 5), f_j(B,C,D), E, M_{i,j}, t_j),\ E = D,$ $D = C,\ C = rol(B, 30),\ B = A,\ A = T,$ for $j = 1, \ldots, 80$.
     - Set $X_1 = add(X_1, A),\ X_2 = add(X_2, B),\ X_3 = add(X_3, C),$ $X_4 = add(X_4, D),\ X_5 = add(X_5, E)$.
4. *Message Digest:* The produced message digest is the 5-word (160-bit) output $MD = (X_1, X_2, X_3, X_4, X_5)$.