CHAPTER I

INTRODUCTION

1. Cryptography

Cryptography is the science of securing data. Cryptography enables us to store sensitive information or transmit it across insecure network so that it cannot be read by anyone except the intended recipient. Some experts argue that cryptography appeared spontaneously sometime after writing was invented. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any un-trusted medium, which includes just about any network, particularly the Internet [1].

Within the context of any application-to-application communication, there are some specific security requirements, including:

- *Authentication:* The process of proving one's identity.
- *Privacy/confidentiality:* Ensuring that no one can read the message except the intended receiver.
- *Integrity:* Assuring the receiver that the received message has not been altered in any way from the original.
- *Non-repudiation:* A mechanism to prove that the sender really sent this message.

Cryptography not only protects data from theft or alteration, but can also be used for user authentication. There are, in general, three types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions. In all cases, the initial unencrypted data is referred to as *plaintext*. It is encrypted into *ciphertext*, which will in turn be decrypted into usable plaintext.

1.1 Secret Key Cryptography

With *secret key cryptography*, a single key is used for both encryption and decryption. Sender uses the key (or some set of rules) to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key (or ruleset) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called *symmetric encryption*.

With this form of cryptography, it is obvious that the key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key.

Secret key cryptography algorithms that are mostly used today include:

- *Data Encryption Standard (DES):* The most common SKC scheme used today, DES was designed by IBM in the 1970s and adopted by the National Institute for Standards and Technology (NIST) in 1977 for commercial and unclassified government applications. DES is a block-cipher employing a 56-bit key that operates on 64-bit blocks. An algorithm takes a fixed-length string of plaintext and transforms it through a series of complicated operations into another ciphertext bitstring of the same length [2].
- *Advanced Encryption Standard (AES):* AES uses an SKC scheme called Rijndael, a block cipher designed by Belgian cryptographers Joan Daemen and Vincent Rijmen. The algorithm can use a variable block length and key length; the latest specification allowed any combination of keys lengths of 128, 192, or 256 bits and blocks of length 128, 192, or 256 bits [2].

1.2 Public-Key Cryptography

Public-key cryptography has been said to be the most significant new development in cryptography in the last 300-400 years. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key [6].

Public Key Cryptography depends upon the existence *one-way functions*, or mathematical functions that are easy to computer whereas their inverse function is relatively difficult to compute. Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the ciphertext. The important point here is that it **does not matter which key is applied first**, but that both keys are required for the process to work. Because a pair of keys is required, this approach is also called *asymmetric cryptography*. In PKC, one of the keys is designated the *public key* and may be advertised as widely as the owner wants. The other key is designated the *private key* and is never revealed to another party.

Public-key cryptography algorithms that are in use today for key exchange or digital signatures include:

- *RSA:* The first, and still most common, PKC implementation, named for the three MIT mathematicians who developed it — Ronald Rivest, Adi Shamir, and Leonard Adleman. RSA today is used in hundreds of software products and can be used for key exchange, digital signatures, or encryption of small blocks of data. RSA uses a variable size encryption block and a variable size key. The key-pair is derived from a very large number, *n*, that is the product of two prime numbers chosen according to special rules; these primes may be 100 or more digits in length each, yielding an *n* with roughly twice as many digits as the prime factors. The public key information includes *n* and a derivative of one of the factors of *n*; an attacker cannot determine the prime factors of *n* (and, therefore, the private key) from this information alone and that is what makes the RSA algorithm so secure.

- *Diffie-Hellman:* After the RSA algorithm was published, Diffie and Hellman came up with their own algorithm. Diffie-Hellman is used for secret-key key exchange only, and not for authentication or digital signatures.
- *Digital Signature Algorithm (DSA):* The algorithm specified in NIST's Digital Signature Standard (DSS), provides digital signature capability for the authentication of messages.
- *Elliptic Curve Cryptography (ECC):* A PKC algorithm based upon elliptic curves. ECC can offer levels of security with small keys comparable to RSA and other PKC methods. It was designed for devices with limited compute power and/or memory, such as smartcards and PDAs.

2. Hash Function

A cryptographic hash function is an algorithm which processes message of an arbitrary length into a fixed length digest or hash code. Hash functions are one of the most important tools in cryptography. It can be used in achieving many security goals including authenticity, digital signatures, digital time stamping etc.

Depending on whether or not a key is used in designing a hash function, hash function can be divided in to two types:

- Keyed hash function and
- Unkeyed hash function

2.1 Keyed hash functions

As the name indicates, keyed hash functions use a key in generating a hash value. The function will accept two inputs: one a message of arbitrary finite-length, and the other is a fixed-length key. The main idea is that, an adversary without the knowledge of this key should be unable to forge the message. Message Authentication Code (MAC) is a keyed hash function because it uses two different inputs specifically an arbitrary length message and a fixed length key. Besides that, the output is of fixed length.

The map $H : \{0,1\}^* \times \{0,1\}^k \rightarrow \{0,1\}^n$ is said to be a keyed hash function with n -bit output and k -bit key if H is a deterministic function that takes two inputs, the first of an arbitrary length, the second of k -bit length and outputs a binary string of length n -bits. Where k, n are positive integers. $\{0,1\}^n$ & $\{0,1\}^k$ are the set of all binary strings of length n and k respectively and $\{0,1\}^*$ is a set of all finite binary strings [3].

2.2 Unkeyed hash functions

Almost all the hash functions that have been used since the early 1990's for various types of applications in cryptography are unkeyed. The generation of hash function under this mechanism do not need a key. These hash functions can be used for error detection, by appending the digest to the message during the transmission. The error can be detected, if the digest of the received message, at the receiving end is not equal to the received message digest. This is also known as modification detection and hence these functions are also called modification detection codes or manipulation detection codes (MDC). Infact, keyed hash functions can also be used for error detection but the unkeyed hash functions are easier to use for this application because there will not be any problem of secrecy of key used.

The map $H : \{0,1\}^* \rightarrow \{0,1\}^n$ is said to be an unkeyed hash function with n -bit output if H is a deterministic function that takes an arbitrary length message as input and outputs a binary string of length n -bit. The notations n, $\{0,1\}^n$ and $\{0,1\}^*$ are similar to that of in section 2.1.

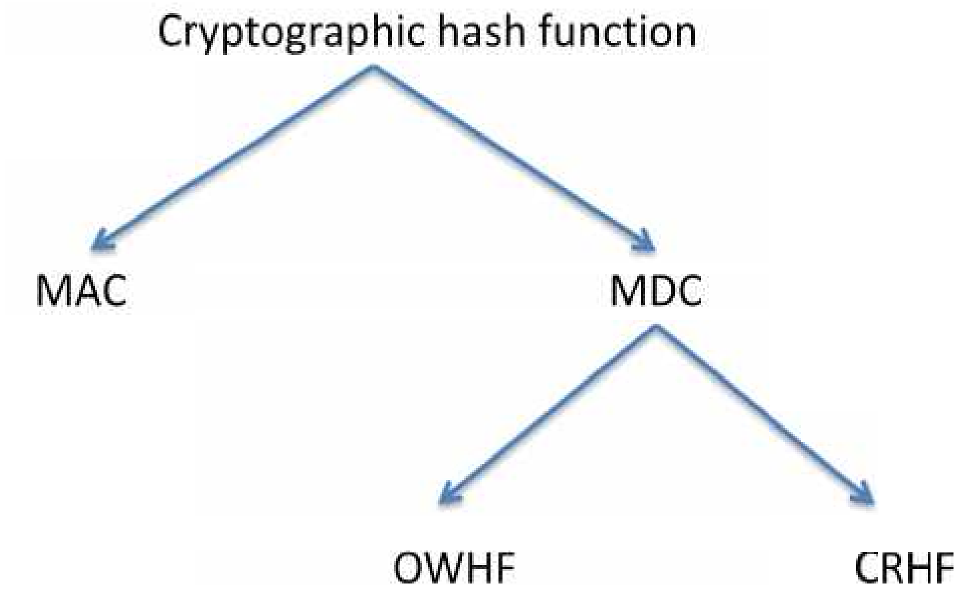


Figure 1: A taxonomy for cryptographic hash functions [9].

2.3 One-way hash function (OWHF)

A one-way hash function is a function h satisfying the following conditions:

- The description of h must be publicly known and should not require any secret information for its operation.
- The argument X can be of arbitrary length and the result h(X) has a fixed length of n bits.
- Given h and X, the computation of h(X) must be "easy".
- The hash function must be one-way in the sense that given a Y in the image of h, it is "hard" to find a message X such that h(X) = Y and given X and h(X) it is "hard" to find a message $X' \neq X$ such that h(X′) = h(X).

The first part of the last condition corresponds to the intuitive concept of one-wayness, namely that it is "hard" to find a preimage of a given value in the range. The second part of this condition, namely that finding a second preimage should be hard, is a stronger condition, that is relevant for most applications [9].

2.4 Collision resistant hash function (CRHF)

A collision resistant hash function is a function h satisfying the following conditions:

- The description of h must be publicly known and should not require any secret information for its operation.
- The argument X can be of arbitrary length and the result h(X) has a fixed length of n bits.
- Given h and X, the computation of h(X) must be "easy".
- The hash function must be one-way in the sense that given a Y in the image of h, it is "hard" to find a message X such that h(X) = Y and given X and h(X) it is "hard" to find a message $X' \neq X$ such that $h(X') = h(X)$.
- The hash function must be collision resistant: this means that it is "hard" to find two distinct messages that hash to the same result.

2.5 Message Authentication Code (MAC)

Message Authentication Codes have been used for a long time in the banking community and are thus older than the open research in cryptology that started in the mid seventies. However, MAC's with good cryptographic properties were only introduced after the start of open cryptologic research.

A MAC is a function satisfying the following conditions:

- The description of h must be publicly known and the only secret information lies in the key.
- The argument X can be of arbitrary length and the result h(K,X) has a fixed length of n bits.
- Given h, X and K, the computation of h(K,X) must be "easy".
- Given h and X, it is "hard" to determine h(K,X) with a probability of success "significantly higher" than 1/2n. Even when a large set of pairs {Xi, h(K, Xi)} is known, where the Xi have been selected by the

opponent, it is “hard” to determine the key K or to compute h(K, X′) for any X′ ≠ Xi. This last attack is called an adaptive chosen text attack.

3. Hash functions based on block ciphers

Hash functions based on block ciphers, are usually slower when compared to that of the dedicated hash functions. But, in few cases they are useful and easier because single implementation of block cipher can be used for a block cipher as well as a hash function. 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[3].

4. Dedicated hash functions:

Hash functions that are specially designed for the purpose of hashing a plaintext are known as *dedicated hash functions*. These hash functions are not based on hard problems such as factorization and discrete logarithms.

Different types of Dedicated hash functions are:

- MD2 : Designed for systems with limited memory, such as smart cards.
- MD4 : Developed by Rivest, similar to MD2 but designed specifically for fast processing in software.
- MD5 : Also developed by Rivest after potential weaknesses were reported in MD4; this scheme is similar to MD4 but is slower because more manipulation is made to the original data.
- Secure Hash Algorithm (SHA)-0
- SHA-1
- SHA-2
- HALAVL (*HAsh of VAriable Length)*
- RIPEMD

The basic requirement for a cryptographic hash function is that the hash value does not reveal any information about the message itself, and moreover that it is hard to find other messages that produce the same hash value. If only a single bit of the message is changed, it is expected that the new hash value is dramatically different from the original one. A hash function is required to have the following features:

- **Preimage resistant:** A hash function hash is said to be preimage resistant if it is hard to invert, where "hard to invert" means that given a hash value h, it is computationally infeasible to find some input x such that hash(x) = h.
- **Second preimage resistant**. If, given a message x, it is computationally infeasible to find a message y different from x such that hash(x) = hash(y), then hash is said to be second preimage resistant.
- **Collision-resistant.** A hash function hash is said to be collision-resistant if it is computationally infeasible to find two distinct messages x and y such that hash(x) = hash(y).

Hash functions have wide and important role in cryptography. They produce hash values, which concisely represent longer messages or documents from which they were computed. The main role of cryptographic hash functions is in the provision of message integrity checks and digital signatures [15].

4.1 MD5

In the past few years, there have been significant research advances in the analysis of hash functions. The MD4 message-digest algorithm, designed by Ronald Rivest is the first hash function in the MD4 family and was designed to work fast on 32-bit machines [3]. Just a year after its publication in 1990, an attack on the last two out of three rounds (i.e. last 32 out of 48 steps) has been presented [8]. In consequence of these attacks, R. Rivest proposed in 1991 a strengthened version of MD4, namely MD5.

Most of the dedicated hash functions are based on the basis construction of Merkel-Damgard. The compression function of MD5 operates on 512-bit blocks further subdivided into sixteen 32-bit sub blocks. The size of the internal state (i. e. the chaining variable) and its output are both 128 bits. One important parameter of the compression function is the number of rounds—the number of sequential updates of the internal state. The compression function of MD5 has 64 rounds each using a 32-bit message subblock to update the internal state via a non-linear mix of boolean and arithmetic operations. Every 32-bit sub block is used four times by the compression function. MD5 allocates 64 bits in the last block to encode the message's length and it pads the message so that its length is congruent to 448 modulo 512. The padding procedure expands the message by at least one bit, so the largest message fitting into one block is 447 bits.

4.2 SHA-1

The Secure Hash Algorithm (SHA) was developed by the National Security Agency (NSA) and published in 1993 by the National Institute of Standard and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS PUB 180) [23]. SHA is based on and shares the same building blocks as the MD4 algorithm. In 1994, NIST announced that a technical flaw in SHA was found. And that this flaw makes the algorithm less secure than originally believed. No further details were given to the public, only that a small modification was made to the algorithm which was now known as SHA-1. It uses the same padding algorithm, as in MD5, breaking the message into 512-bit blocks and encoding the length as a 64-bit number. The size of its internal state and its output length are 160 bits, which is substantially longer than MD5's 128 bits. Although its round functions are simpler and less varied than those of MD5, there are more of them—80 instead of 64. SHA-1 uses a more complex procedure for deriving 32-bit sub blocks from the 512-bit message. If one bit of the message is flipped, more than a half of the sub blocks get changed (as opposed to just four in the case of MD5) [10].

5. Motivation

Historically the first constructions of hash functions were based on strong block ciphers and many efforts have since been done for their design and proof of security. However since this design approach does not necessarily result in fast hash functions in practice and often their hashing speed is much slower than underlying block ciphers, many “dedicated hash functions” suitable for software implementation on modern processors have been proposed and are now widely used in real world applications.

Therefore, performance analysis of hash functions in real environments is recognized as an important research topic.

6. Problem Statement

The most important characteristic of a cryptographic hash function are its security properties. For many applications, however, the speed of a hash function is of almost the same importance.

CHAPTER II

BACKGROUND AND LITERATURE REVIEW

1. Cryptography

The origin of the word cryptology lies in ancient Greek. The word cryptology is made up of two components: "kryptos", which means hidden and "logos" which means word. Cryptology is as old as writing itself, and has been used for thousands of years to safeguard military and diplomatic communications. For example, the famous Roman emperor Julius Caesar used a cipher to protect the messages to his troops. Within the field of cryptology one can see two separate divisions: cryptography and cryptanalysis. The cryptographer seeks methods to ensure the safety and security of conversations while the cryptanalyst tries to undo the former's work by breaking his systems [7].

It is well known that the concealment of information or protection of privacy is as old as writing itself. Human mind found many ways to conceal information: steganography, i.e., the hiding of the mere existence of a message, codes, where words or combinations of words are replaced by fixed symbols, and cryptology or ciphers, where information is transformed to render it useless for the opponent [1].

In the past few years, there have been significant research advances in the analysis of hash functions. The advent of electronic computers and telecommunication networks created the need for a widespread commercial encryption algorithm. In this respect, the publication in 1977 of the Data Encryption Standard (DES) by the U.S. National Bureau of Standards was without any doubt an important milestone. The DES was designed by IBM in cooperation with the National Security Agency (NSA). It later became an ANSI banking standard [6]. Soon the need for specific measures to protect the authenticity of the information became obvious, since authenticity does not come for free together with secrecy protection. The first idea to solve this problem was to add a simple form of redundancy to the plaintext before encryption, namely the sum modulo 2 of all plaintext blocks [13]. This showed to be insufficient, and techniques to construct redundancy which is a complex function of the complete message were proposed. It is not surprising that the first constructions were based on the DES [9].

1.1 Privacy protection with symmetric cryptology

Cryptology has been used for thousands of years to protect communications of kings, soldiers, and diplomats. Until recently, the protection of communications was almost a synonym for the protection of the secrecy of the information, which is achieved by encryption. In the encryption operation, the sender transforms the message to be sent, which is called the plaintext, into the ciphertext. The encryption algorithm uses as parameter a secret key; the algorithm itself is public. The receiver can use the decryption algorithm and the same secret key to transform the ciphertext back into the plaintext. The main concept of encryption is to replace the secrecy of a large amount of data by the secrecy of a short secret key which can be communicated via a secure channel (see Figure 2). Because the key for encryption and decryption are equal, this approach is called symmetric cryptography.

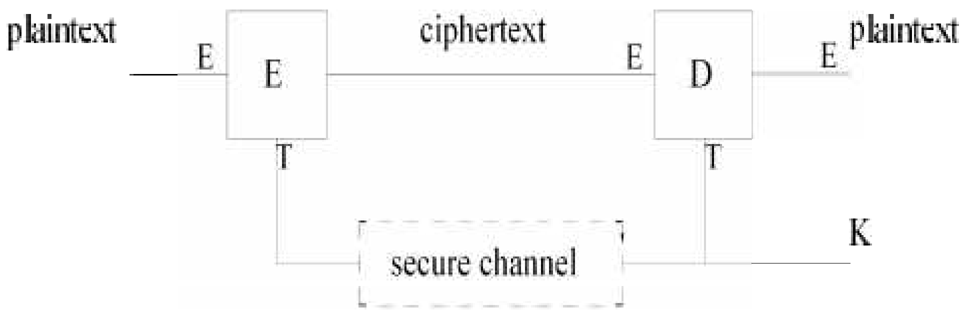


Figure 2: Model of symmetric encryption system [7]

1.2 Authentication with symmetric cryptology

In the military world it was known for some time that modern telecommunication channels like radio require additional protection of the authenticity. One of the techniques applied was to append a secret key to the plaintext before encryption [10]. The protection then relies on the error propagating properties of the encryption algorithm and on the fact that the secret key for authentication is used only once.

In the banking environment, there is a strong requirement for protecting the authenticity of transactions. Before the advent of modern cryptology, this was achieved as follows: the sender computes a function of the transaction totals and a secret key; the result, which was called the test key, is appended to the transaction. This allows the receiver of the message, who is also privy to the secret key, to verify the authenticity of the transaction.

New techniques were proposed to produce redundancy under the form of a short string which is a complex function of the complete message. A function that compresses its input was already in use in computer science to allocate as uniformly as possible storage for the records of a file. It was called a hash function, and its result was called a hashcode. If a hash function has to be useful for cryptographic applications, it has to satisfy some additional conditions. Informally, one has to impose that the hash function is one-way (hard to invert) and that it is hard to find two colliding inputs, i.e., two inputs with the same output. If the information is to be linked with an originator, a secret key has to be involved in the hashing process (this assumes a coupling between the person and his key), or a separate integrity channel has to be provided. Hence two basic methods can be identified.

- The first approach is analogous to the approach of a symmetric cipher, where the secrecy of large data quantities is based on the secrecy and authenticity of a short key. In this case the authentication of the information will also rely on the secrecy and authenticity of a key. To achieve this goal, the information is compressed with a hash function, and the hashcode is appended to the information. The basic idea of the protection of the integrity is to add redundancy to the information. The presence of this redundancy allows the receiver to make the distinction between authentic information and bogus information. In order to guarantee the origin of the data, a secret key that can be associated to the origin has to intervene in the process. The secret key can be involved in the compression

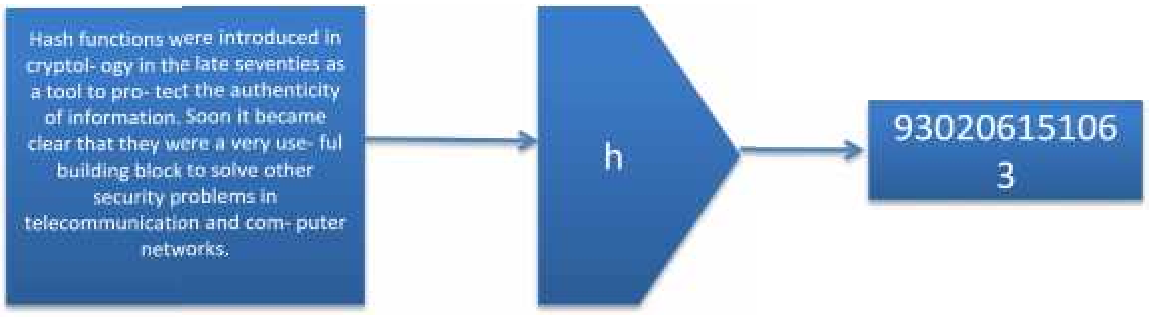

Hash functions were introduced in cryptol- ogy in the late seventies as a tool to pro- tect the authenticity of information. Soon it became clear that they were a very use- ful building block to solve other security problems in telecommunication and com- puter networks.
h
93020615106
3

Figure 3: A hash function[9]

process; the hash function is then called a Message Authentication Code or MAC. A MAC is recommended if authentication without secrecy is required. If the hash function uses no secret key, it is called a Manipulation Detection Code or MDC; in this case it is necessary to encrypt the hashcode and/or the information with a secret key. In addition, the encryption algorithm must have a strong error propagation: the ciphertext must depend on all previous plaintext bits in a complex way. Additive stream ciphers can definitely not be used for this purpose.

The second approach consists of basing the authenticity (both integrity and origin authentication) of the information on the authenticity of a Manipulation Detection Code or MDC. A typical example for this approach is an accountant who will send the payment instructions of his company over an insecure computer network to the bank. He computes an MDC on the file, and communicates the MDC over the telephone to the bank manager. The bank manager computes the MDC on the received message and verifies whether it has been modified. The authenticity of the telephone channel is offered here by voice identification.

2. Applications of hash function in cryptography

Hash functions are used in many situations to support various cryptographic protocols. Their most common applications are creation and verification of digital signature (a means to verify the authenticity of an electronic document), MAC, File integrity verification, Password table etc.

2.1. Digital Signature

One of the widespread applications of cryptographic hash functions is digital signatures. We all know what a hand-written signature is and we certainly understand its purpose. It is a way to prove that a paper document is signed by us and not by someone else. The digital signature is the electronic equivalent to the hand-written signature with regard to its purpose. More precisely, a digital signature is a sort electronic "stamp" or digital "fingerprint" placed on a document that is unique to the signer of the document and to the signed document. One major difference between a digital and a hand-written signature is that for every different document, the digital signature is different even if the signer and the private/public key pair are the same. We also note that a digital signature scheme provides both data integrity protection and data origin authentication. Instead of using the signature algorithm to sign the original data directly, a cryptographic hash function is used to compute the digest of the message first and then, rather than the data, the digest is signed. During verification phase, the digest of data to be verified is computed and is used in the signature verification algorithm. The big advantage of this approach is its increased efficiency of signing long messages. Signature algorithms are much slower than hash functions and signing long messages directly would take a very long time. By computing cryptographic digest of the data first it is possible to avoid this costly computation [14].

2.2. Data Integrity

Data authentication and data integrity are two issues which cannot be separated. There should be a source for any data which has been altered and if a source cannot be determined, then the question of alteration cannot be settled. Integrity mechanisms thus provide data authentication and vice versa. Hence data authentication should also be considered along with data integrity. Data integrity is the property whereby data has not been altered in an unauthorized manner since the time it was created, transmitted, or stored by an authorized source.

Hash functions are widely used to verify file integrity. Indeed, it certifies that the document has not been modified somewhere between the moment it was sent and the moment it was received. Files can become corrupted for a variety of reasons including faulty storage media, errors in transmission, write errors during copying or moving, and software bugs. File integrity verification ensures that a file has not been corrupted by comparing the file's hash value to a previously calculated value. Before sending the data its digest is computed by the means of a cryptographic hash function. The digest is then sent over a secure channel to the recipient who after receiving the original digest computes the hash of the received data and compares both hash values. If they are different, the information has been modified somehow on its way over the insecure channel. On the other hand, if the digests are identical, with overwhelming probability the message has not been altered, provided that the cryptographic hash function is secure.

2.2. Password tables

A common method of client authentication is to require the client to present a password previously registered with the server. Storing passwords of all users on the server poses an obvious security risk. Fortunately, the server need not know the passwords—it may store their hashes and use the information to match it with the hashes of alleged passwords. With this scheme in place, the adversary succeeds in breaking into the system if he is able to construct any string that has the same hash as any of the original passwords. This should be difficult even if the adversary has access to the password file.

3. Merkle-Damgard Construction

Most of the Dedicated hash functions are based on the basis construction of Merkle-Damagard. Named after its two inventors, the American Ralph C. Merkle and the Danish Ivan Damgard, the Merkle-Damgard structure defines a generic step by step procedure for deriving a fixed-length output value from a variable-length input value.

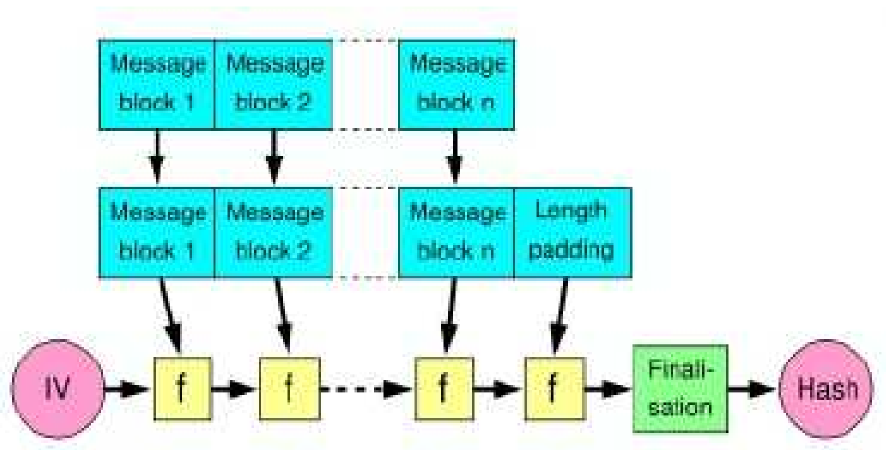


Figure 4: Merkle-Damgard structure [10]

The main building blocks of the Merkle-Damgard structure are:

- **IV**: Initialization Vector or Initial Value is a fixed value used as the chaining variable for the very first iteration.
- **f**: the compression function or one-way hash function which is either specially designed for hashing or based on a block cipher. The compression function generally takes an input of fixed length and produces an output of fixed length.
- **Finalisation**: an output transformation function which usually reduces further the length of the output value of the last iteration.

- **Hash**: the message digest or the hash result.

As we can see from the figure 4, the entire message to be hashed is first divided into n blocks of equal length. The actual length of the message blocks depends on the requirements set by the compression function f. The message is then padded, always, such that its length is a multiple of some specific number. The padding is done by adding after the last bit of the last message block a single 1-bit followed by the necessary number of 0-bits. The length padding which consists of appending a k-bit representation the length in bits of the original message (that is, the message before any padding has been applied) takes place in such a way that the padding length bits are added as the last bits of the padded message block prior to being processed by the compression function. Every block is processed by the compression function in the same iterative manner. The compression function always takes two inputs in each step or iteration, a message block and a chaining variable. In the first iteration, the chaining variable is the IV or Initialization Vector. It is given, together with the first block of message, as inputs to the compression function. The output of the compression function f in the first iteration is the chaining variable in the second iteration. The output of the compression function f in the ith iteration is the chaining variable in the (i + 1)th iteration and so on until we reach the last iteration.

In the last iteration, the output of the compression function is used as an input to a finalization function which reduces further the length of the final output value from the compression function (however, in some cases the finalization function is not present and the output value of the compression function f in the last iteration is used as the final hash result).

In general, for a message M consisting of t blocks M_0, M_1, ...,M_{t-1}, the computation of the message digest can be defined as follows:

H_0 =IV,

For $0 \leq i < t$

H_{i+1} =f(Hi,Mi)

H(M) = Finalisation(Ht)

In a pseudo code, the same computation can be defined as follows:

```
computeDigest(M ) {
        $M_0 \cdots_{t-1}$ = divBlock(M)
        $H_0$ =IV
        for(i = 0; i < t; i + +) {
                $H_{i+1}$ =f(Hi,Mi)
                }
        H(M) = Finalisation(Ht)
         returnH(M)
        }
```

Here the function computeDigest() takes the message M as input and returns as output the hash result H (M). The inner function divBlock() breaks up the message M into t blocks of equal length and returns an array consisting of the t message blocks.

4. Types of attacks on hash functions

A hash function is said to be broken if an attacker is able to show that the design of the hash function violates at least one of the claimed security property. For example, if a hash function is claimed to be collision resistant, a successful attack is to find at least one collision such that two different messages have the same message digest [5].

6.1. Birthday attack

The idea behind this attack originates from Birthday paradox. The birthday paradox states that given a group of 23 randomly chosen people the probability, of at least two people having the same birthday is more than 1⁄2 [17]. The mathematics behind this is being used to generate a well-known cryptographic attack called birthday attack.

To describe this, let us assume that the message digest of length n bits which provides 2^n 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 a cryptanalyst, the probability of finding a match between the two pools is approximated by,

$$p \approx 1 - 1/e^{x1x2}/^{2n}$$

where the approximation is more accurate for larger values of x_2 compared with that of x_1 [17].

6.2. Differential attack

This attack is applicable to block ciphers and hash functions and is based on the study of the relation between input and output differences. The attack is statistical as one search for input differences that are likely to cause a certain output difference. If the difference is equal to zero then a collision can be achieved [18][19].

In general, the differential attack especially in block ciphers is a kind of XOR differential attack which uses exclusive-or as the difference. The differential attack was introduced by E. Biham and A. Shamir to analyze the security of DES-like cryptosystems. E. Biham and A. Shamir [18], described that differential cryptanalysis is a method which analyzes the effect of particular differences in plain text pairs on the differences of the resultant cipher text pairs.

7. The MD5 hash Algorithm

The algorithm accepts an input message of arbitrary length and produces a 128-bit message digest, fingerprint or hash result.
The actual processing of the MD5 algorithm consists of the following 5 steps:

Step 1: Append padding bits
During this step, the message is extended or padded in such a way that its total length in bits is congruent to 448 modulo 512. This operation is always performed even if the message's length in bit is originally congruent to 448 modulo 512. We notice that 448 + 64 = 512, so the message is padded such that its length is now 64 bits less than an integer multiple of 512.

Step 2: Append length

A 64 − bit representation of the length in bits of the original message M (before the padding bits were added) is appended to the result of step 1. Here, there is a little trick in representing the length of message M in the case where it is greater than 2^{64}. If the length of the original message is indeed greater than, then only the low order 64 − bits of the length of message M are used. Hence, the field contains the length of the original message M modulo 2^{64}. These bits are appended as two 32 − bit words and appended low-order (least significant) word first.

Step 3: Initialize MD buffer

A 128 − bit (4 × 32 − bit) buffer (A, B, C, D) is used to hold intermediate and final result of the MD5 hash algorithm. These registers are initialized to the following 32−bit values in hexadecimal:

A = 67452301

B = efcdab89

C = 98badcfe

D = 10325476

These values are stored in little-endian format, meaning that the low -order bytes of a word is placed in the low-address byte position. The initialization values appear then as follows:

Word A = 01234567

Word B = 89abcdef

Word C = fedcba98

Word D = 76543210

Step 4: Processing message in 512-bit blocks

The main part of the algorithm is the compression function that consists of four rounds of processing. Each round takes as input the current 512-bit block being processed represented as m_i where i = 1,2,....,r and the 128-bit buffer value 'ABCD' which is updated each round. The four rounds are almost identical, with the main difference being that each round uses a different primitive logical function, denoted by F, G, H, and I in the specification.

Round	Primitive function	Steps(64)
1	$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$	$0 \leq j \leq 15$
2	$G(X,Y,Z)=(X \wedge Z) \vee (Y \wedge \neg Z)$	$16 \leq j \leq 31$
3	$H(X,Y,Z)=X \oplus Y \oplus Z$	$32 \leq j \leq 47$
4	$I(X,Y,Z)=Y \oplus (X \vee \neg Z)$	$48 \leq j \leq 63$

Table 1: The primitive functions of the MD5 compression algorithm.

where

$\wedge$ = AND, V = OR, $\oplus$ = XOR, $\neg X$ = NOT(X)

Each round takes as input the current 512-bit message block Mk and the 128-bit buffer value ABCD and produces as output an updated value of the buffer earlier referred to as the chaining variable CV_k. In addition, each round also uses one-fourth of a 64-element table denoted T[1···64] which is constructed from the sine function. It is constructed such that the i − th element of the table T , denoted T [i], is equal to the integer part of 2^{32} × abs(sin(i)), where i is in radian. Since abs × (sin(i)) is a number between 0 and 1, the elements of the table T are numbers less than or equal to 2^{32}, Hence, they can be represented in 32 bits.

Step 5: Output

The output from the very last round is the 128-bit hash result or message digest we obtain after we have incrementally processed all t 512-bit blocks of the message. The entire process can be summarized as follows:

$CV_0 = IV$

$CV_{k+1} = SUM_{32}(CV_k, RF_I[M_k, RF_H[M_k, RF_G[M_k, RF_F[M_k, CV_k]]]])$

MD5SUM = CV_t

Where

IV = the initial value of the ABCD buffer, defined by step 3

Mk = the k − th 512-bit block of the message

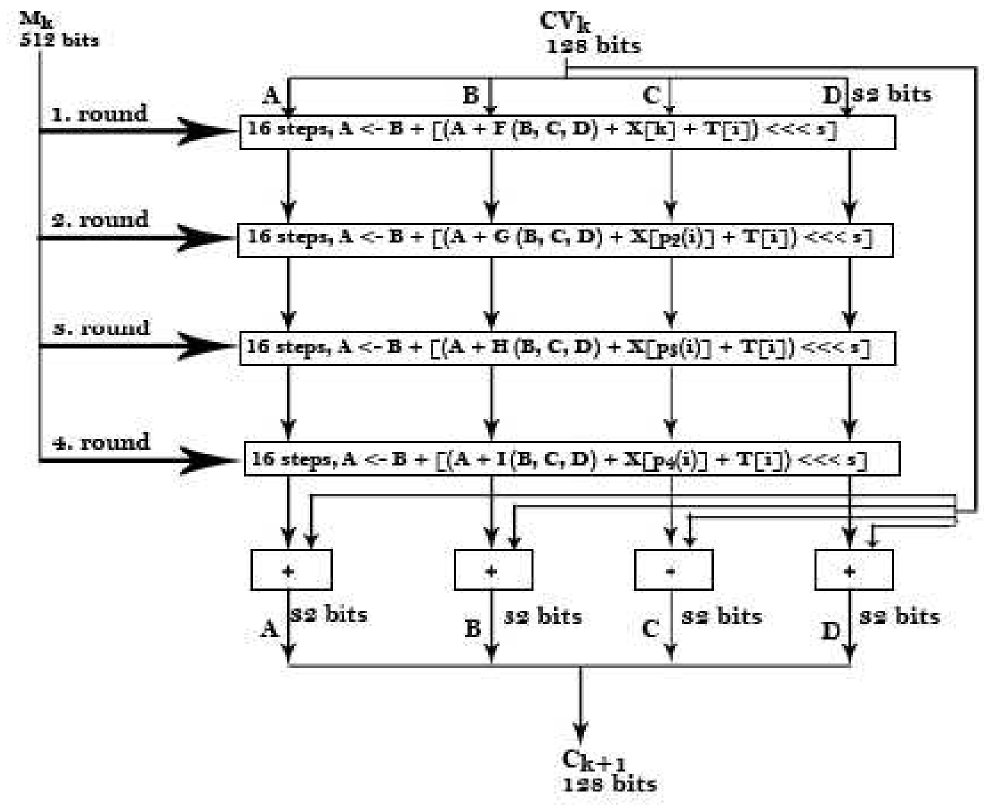
CV_k = the chaining variable processed with the k − th block of message

RF_x = the round function using primitive logical function x

$MD5_{SUM}$ = the final hash result or message digest

SUM32 = addition modulo 2^{32}

The final message digest is stored in the ABCD buffer (see figure 5). It is output by beginning with the low order byte of A and ending with the high order byte of D.



Mk 512 bits
CVk 128 bits
A
B
C
D 32 bits
1. round
16 steps, A <- B + [(A + F (B, C, D) + X[k] + T[i]) <<< s]
2. round
16 steps, A <- B + [(A + G (B, C, D) + X[p2(i)] + T[i]) <<< s]
3. round
16 steps, A <- B + [(A + H (B, C, D) + X[p3(i)] + T[i]) <<< s]
4. round
16 steps, A <- B + [(A + I (B, C, D) + X[p4(i)] + T[i]) <<< s]
+
A 32 bits
B 32 bits
C 32 bits
D 32 bits
Ck+1 128 bits

Figure 5: The MD5 compression of a single 512-bit message block

The four words A, B, C, D of the buffer are used in such a way that produces a word-level circular right shift of one word after every step. After each step, we update one of the 4 bytes of the ABCD buffer. Knowing that we have 16 steps, it results that each 32-bit word of the buffer is updated four times at the end of the fourth round and an additional fifth time to produce the final output (chaining variable) for the current block.

In round 1, the primitive function F is used. The 16 32-bits words of the 512-bit message block are use in their original order M[0] through M[15]. Each of the 16 words is used only once in each step.

8. Security of MD5

The MD5 algorithm has one interesting property that every bit of the output is a function of every bit of the input. The complexity in the repeated use of the primitive functions and the additive constant T [i] together with the circular left shifts unique to every round produce a well mixed hash result. This procedure makes it very unlikely that two messages that show a similar regularity will have the same hash result.

However, In August 2004 Xiaoyun Wang and Hongbo Yu of Shandong University in China published an article [19] in which they describe an algorithm that can find two different sequences of 128 bytes with the same MD5 hash. Their research was motivated by the possibility of finding a colliding pair of messages, each consisting of two blocks.

Further advances were made in breaking MD5 in 2005, 2006, and 2007 [20]. In an attack on MD5 published in December 2008, a group of researchers used this technique to fake SSL certificate validity.

9. The SHA-1 Algorithm

The SHA-1 algorithm accepts as input a message with a maximum length of 2^{64} -1 and produces a 160-bit message digest as output. The message is processed by the compression function in 512-bit block. Each block is divided further into sixteen 32-bit words denoted by Mt for t = 0, 1, · · · ,15. The compression function consists of four rounds, each round is made up of a sequence of twenty steps. A complete SHA-1 round consists of eighty steps where a block length of 512 bits is used together with a 160-bit chaining variable to finally produce a 160-bit hash value. The processing works as described in the following steps:

Step 1: Append padding bits

The original message is padded so that its length is congruent to 448 modulo 512. Again, padding is always added although the message already has the desired length. Padding consists of a single 1 followed by the necessary number of 0 bits.

Step 2: Append length

A 64-bit block treated as an unsigned 64-bit integer (most significant byte first), and representing the length of the original message (before padding in step 1), is appended to the message. The entire message's length is now a multiple of 512.

Step 3: Initialize the buffer

A 160-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as five 32-bit registers (A, B, C, D and E). These registers are initialized with the following 32-bit hexadecimal values:

A: 67452301

B: EFCDAB89

C: 98BADCFE

D: 10325476

E: C3D2E1F0

Step 4: Processing message in 512-bit blocks

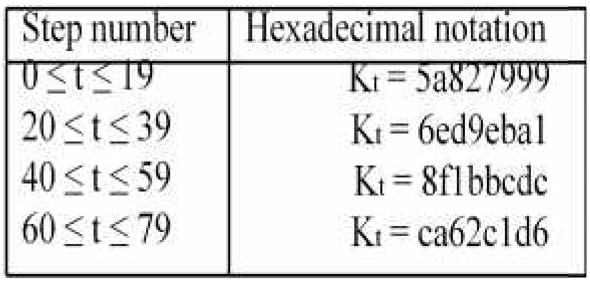
The heart of the algorithm is the module that has four similar rounds of processing each of 20 steps. The inputs of each round are the 512-bit message

block currently being processed and the 160-bit buffer value ABCDE. The contents of the buffer are updated as the process continues. Each round have a similar structure, but each uses a different primitive logical function which are refereed as P, Q, R and S. These are defined as in table 2.
The output of the fourth round is added to the input to the first round in a way such that bits of the input are added to the corresponding bits of the output. This addition is similar to that of in the MD5 process.

Step	Primitive logic function ~	~(t,B,C,D)
(0 ≤ t ≤ 19)	P(t,B,C,D)	(B ∧ C) V (B ∧ D)
(20 ≤ t ≤ 39)	Q(t,B,C,D)	B⊕C⊕D
(40 ≤ t ≤ 59)	R(t,B,C,D)	(B ∧ C) V (B ∧ D) V (C ∧ D)
(60 ≤ t ≤ 79)	S(t,B,C,D)	B⊕C⊕D

Table 2: Primitive logic functions used in SHA-1.

The compression function is divided into twenty sequential steps composed of four rounds of processing where each round is made up of twenty steps. The four rounds are structurally similar to one another with the only difference that each round uses a different Boolean function, which we refer to as P,Q,R,S above and one of four different additive constants Kt (0 ≤ t ≤ 79) which depends on the step under consideration. The values of the four dictinct additives constant are given in table 3 below.



Step number	Hexadecimal notation
$0 \leq t \leq 19$	K_t = 5a827999
$20 \leq t \leq 39$	K_t = 6ed9eba1
$40 \leq t \leq 59$	K_t = 8f1bbcdc
$60 \leq t \leq 79$	K_t = ca62c1d6

Table 3: The four additive constants used in SHA-1 algorithm

Every step updates two of the five registers. The step operation which updates the value of the E register and rotates the value of the B register by 30 bit position to the left is of the following form:

A, B, C, D, E ← (E + fr(t, B, C, D) + [A <<< 5] + Mt + Kt), A, [B <<< 30], C,D

Where

A,B,C,D,E	= the five registers of the SHA-1 buffer
t	= thestepnumber,$0 \le t \le 79$
fr	= the primitive logical function used in step t and round r
<<< s	= the circular left shift of the 32-bit word by s bits
Mt	= a 32-bit word derived from the current 512-bit input block
Kt	= one of four additive constants
+	= addition modulo 2^{32}

Each 512-bit message block comprises 16 32-bit words (16 × 32 = 512). During the step which processes the message in 512-bit blocks, the first 16 words of every message block is taken and used directly as it appears. The additional 64 blocks are derived by following the algorithm given by:

$M_t = (M_{t-16} \oplus M_{t-14} \oplus M_{t-8} \oplus M_{t-3}) <<< 1$

This means that if we assume that word M_0 through M_{15} represent the first 16 words (used in the first 16 steps), then for the step 17 the word M_{16} is given by:

$M_{16} = (M_0 \oplus M_2 \oplus M_8 \oplus M_{13}) <<< 1$

Step 5: Output

After all the blocks of the message are processed in this way, the output of the last stage is of a 160-bit message digest.

10. Security of SHA-1

We know that SHA-1 produces a 160-bit message digest. If one cannot find collision in less than 280 operations, then SHA-1 is considered secure and can still be used. But recently, a group of Chinese cryptographers [21] were able to find collisions in SHA-1 in 2^{69} calculations. This is about 2000 times faster than a brute-force search attack.

Chapter III

Implementation

Provably secure constructions of cryptographic hash functions consist of two ingredients, which may be studied independently of each other. The first component is a compression function that maps a fixed-length input to a fixed-length output. The second component of a construction is a domain extender that, given a compression function, produces a function with arbitrary-length input.

Compression function. From the theorist's point of view, a one-way function is the most basic primitive, from which many other cryptographic tools can be derived. The main idea is to use a compression function, i.e. a function that maps longer, but fixed size intputs to shorter outputs.

The compression function takes the intermediate result or chaining value and a block of input data and calculates the next intermediate result.

Domain extender. The domain extender is a generic construction that transforms a compression function with fixed-length input into a hash function with arbitrary input. The simplest and most commonly used domain extender is called the Merkle-Damgard construction and it works as follows:

```
Input: message M
1. Break M into m-bit blocks $M_1$,. . . ,$M_k$, padding if necessary;
2. Let $M_{k+1}$ be encoding of |M|;
3. Let $h_0$ = IV;
4. For i=1 to k+1 let hi =C($h_{i-1}$, Mi);
5. Output $h_{k+1}$.
```

The construction iterates the compression function C: the output of C, together with the next block of the message, becomes the input to the next application of C. The hash of the last block, which contains an encoding of the length of the message, is the hash of the entire message.

1.1. General Model

(a) High level view

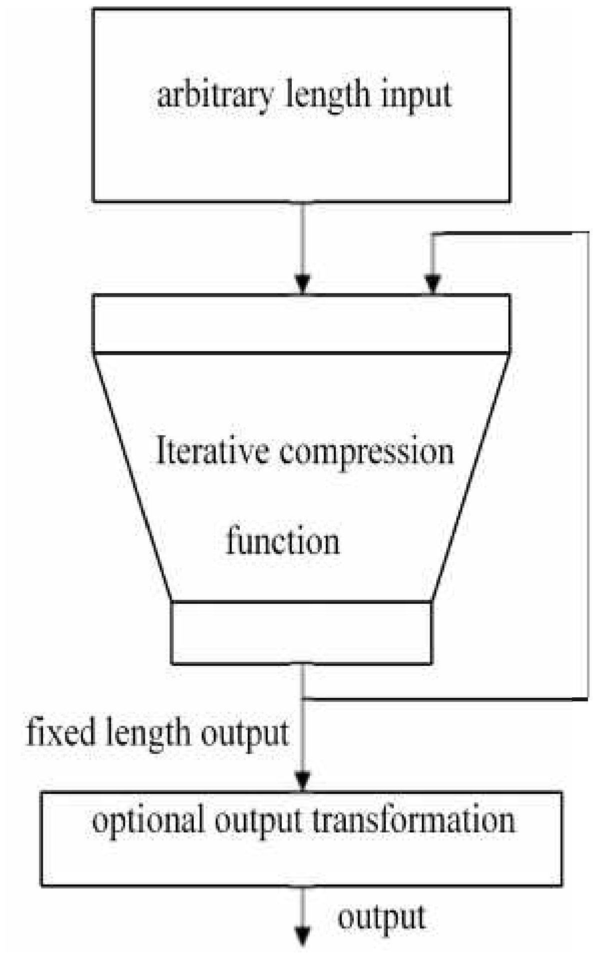

arbitrary length input
Iterative compression function
fixed length output
optional output transformation
output

Figure 6: The Architecture of Cryptographic hash model [3]

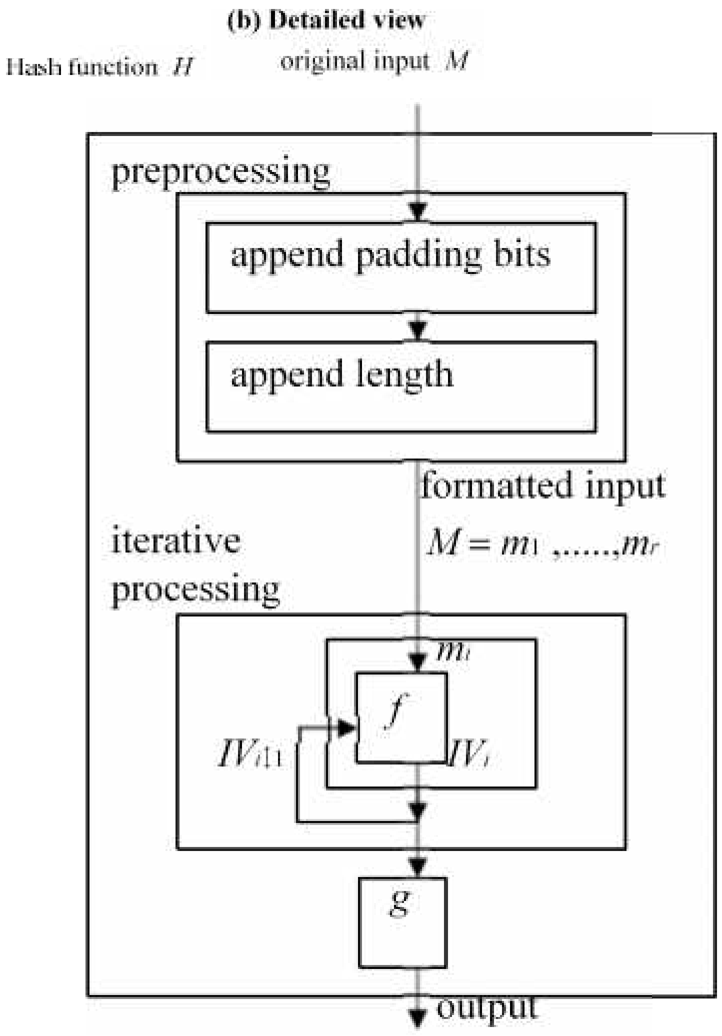


Figure 7: Detailed view of Cryptographic hash model [3]

Each block of the message M represented as m_i where i = 1,2,.....,r serves as input to an internal fixed size hash function f , known as the compression function of H . The iterative processing starts with a predefined initial value, the initialization vector IV_0 . That is, the first round of the iterative process takes IV_0 and m_1 as inputs and computes an n -bit intermediate value for some fixed n, this in turn serves as an input to the second round along with the second block of the message m_2. This process is continued r times and the final output IV_r is of n -bit length, which is generally known as the message digest.

1.2. MD5 Algorithm

As we can see from the figure below, the entire message to be hashed is first divided into n blocks of equal length. The message is then padded, always, such that its length is a multiple of some specific number. The padding is done by adding after the last bit of the last message block a single 1-bit followed by the necessary number of 0-bits. The length padding which consists of appending a k-bit representation the length in bits of the original message (that is, the message before any padding has been applied) takes place in such a way that the padding length bits are added as the last bits of the padded message block prior to being processed by the compression function. Every block is processed by the compression function in the same iterative manner.

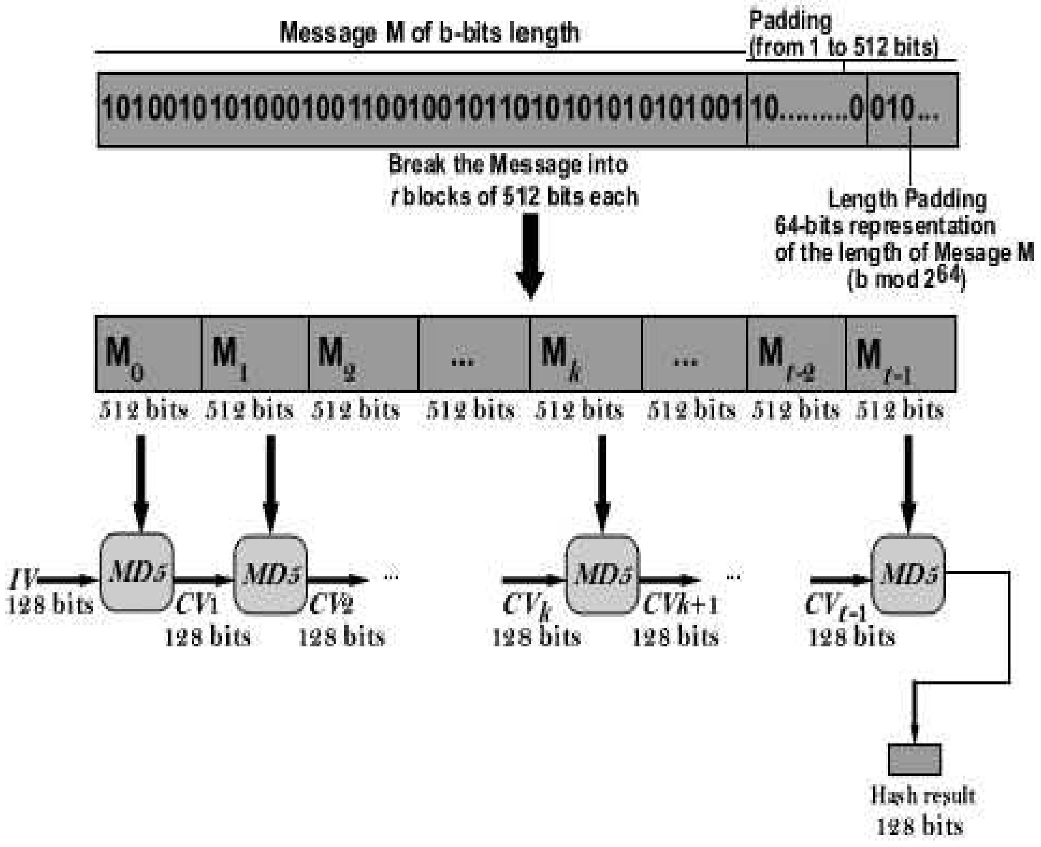
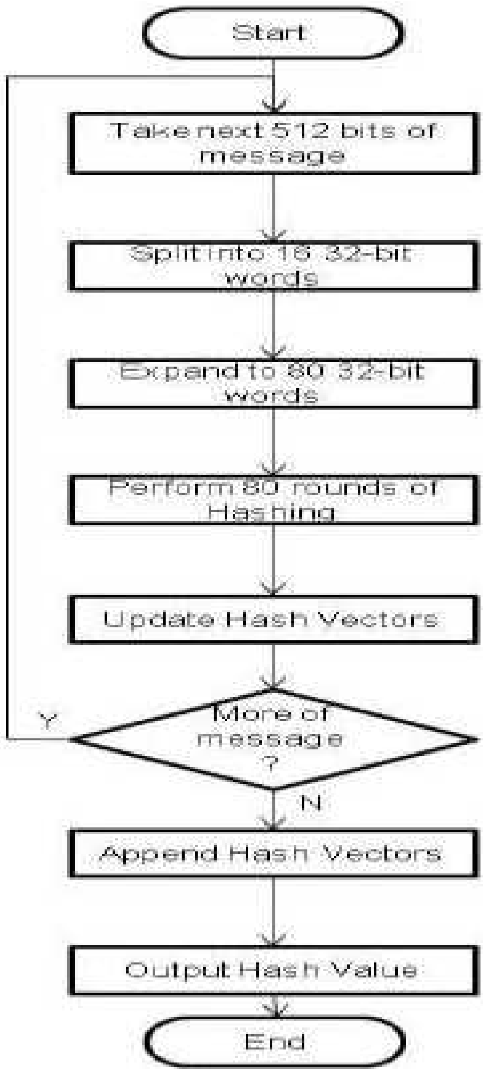


Figure 8: The MD5 Algorithm [10]

1.3. SHA-1 Algorithm

1.3.1. High Level Design



Start
Take next 512 bits of message
Split into 16 32-bit words
Expand to 80 32-bit words
Perform 80 rounds of Hashing
Update Hash Vectors
More of message ?
Y
N
Append Hash Vectors
Output Hash Value
End

Figure 9: Flow chart of SHA-1 Algorithm

The heart of the algorithm is the module that has four similar rounds of processing each of 20 steps. The compression functions consist of two basic components, message expansion and round operations. The compression function of SHA-1 operates on 512-bit message blocks, and utilizes a 160 bit state variable, represented by five 32-bit words, denoted A, B, C, D, E. The block of 512 bits is expanded to 2560 bits, represented by 80 words of 32 bits. Each of these words is used to update the internal state in a round update function. MD5 follows a similar structure, but uses a 128 bit state variable, and has 64 rounds instead of 80.

CHAPTER V
TESTING AND ANALYSIS

Since the execution speed of a hash function depends on many circumstances, it can not be determined by a single test. Like any other statistics, all test results have to be interpreted thoroughly. The tests presented here focus on comparing the actual speed of the two hash functions in normal environments.

1.1. Influencing factors

Needless to say, the speed of a cryptographic hash function depends on many factors. Additionally, measurements are always based on particular circumstances, therefore generalizations about a slow or fast hash function are vague and unspecific.

In software, hashing speed is heavily determined by several factors:

- The hash function and the desired security. In general, short hash functions are faster than longer and more secure hashes. Increasing the number of rounds or choosing longer or more secure variants increases the running time.
- The software implementation along with the compiler. The capabilities of the compiler and the optimizability of the source code also have an enormous effect on the achievable speed. Most hash functions are implemented in standard C to easily incorporate different hardware platforms and compilers, exchanging speed for easy portability. Therefore, code tailored towards a specific platform can give an advantage, as can a suitable and potent compiler.
- The hardware platform and the CPU. Obviously, the choice of the processor has a huge significance, as it dictates the instruction set and the word size. Furthermore, the clock frequency has a linear influence on the speed of a hash function when comparing CPUs of the same type. However, the processor accounts for several other important aspects:

The internal registers of the processor are very limited, but they can be accessed very quickly. If not all variables of a hash function can be stored in registers, the execution is slowed down considerably. An Intel x86 CPU has, for example, only four general-purpose registers, which is not enough to hold all variables and temporary results of the internal functions of MD5 or SHA-1. The RISC processors from MIPS, on the other hand, have 32 64-bit-wide general-purpose registers. Whenever data does not reside in registers, it has to be fetched from the caches. This does not only apply to the rather small set of internal and temporary values, but also to the message words.

In order to compare the performance of software implementations of hash functions, an average speed has been compiled in Table 5. Instead of testing in many processors, testing has been done only on two processors repeatedly. All timings were performed on a 2.2 GHz Intel and Dual 1.0 GHz PowerPC G4 Processor.

On multitasking systems, an exact measurement of the efficiency of a program is not easy. We can measure execution time in java.

The lower speed of SHA-1 is a result of the message expansion, and the noticeably more complicated routine of each step than in MD5. Its higher security is paid by slower hashing rates, the difference is not unexpected.

Algorithm	Average Speed (Mbits/s)	
	Intel processor	PowerPC G4
MD5	753.4	675.2
SHA-1	570.2	510

Table 5: Average running speed of MD5 and SHA-1

The following comparison shows the average hashing speed (given in Mbits/second) for addressed hash function.

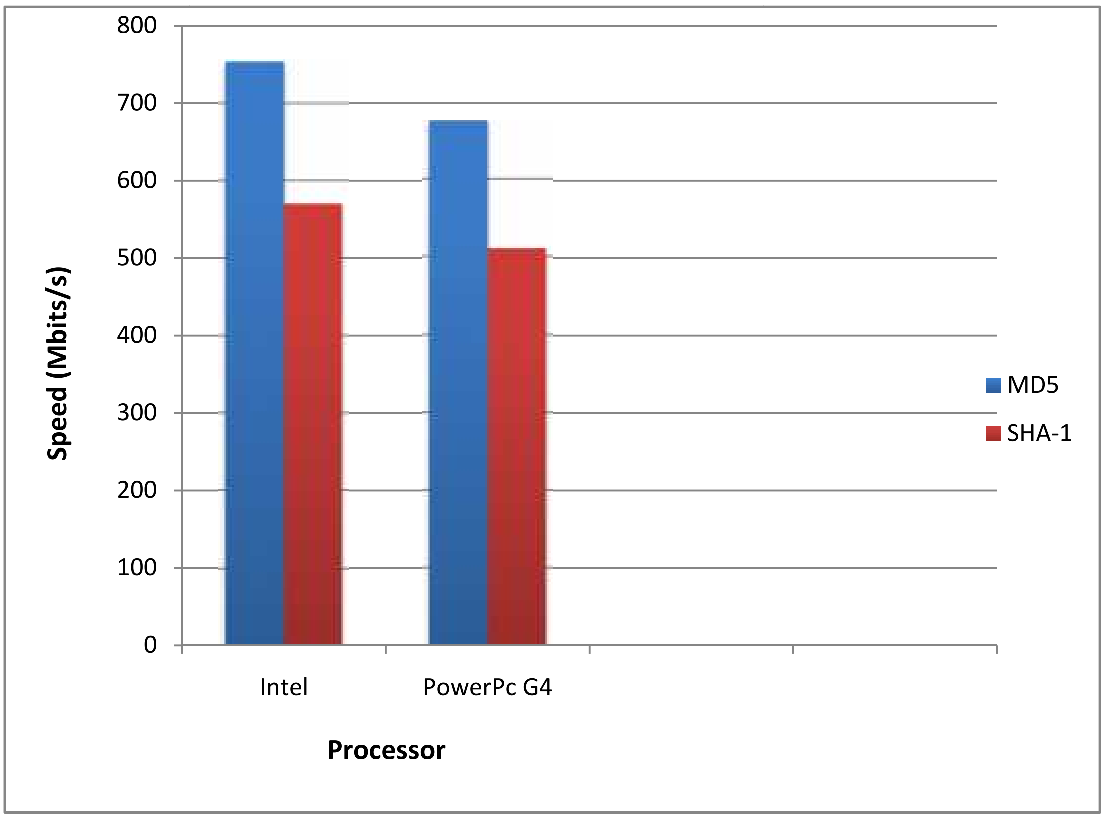


Figure 10: Chart showing speed in Mbits/s for MD5 and SHA-1

1.2. Algorithm comparison

From performance evaluation MD5 algorithm runs faster than SHA-1 in both platforms. SHA-1 uses the same padding algorithm, breaking the message into 512-bit blocks and encoding the length as a 64-bit number. The size of its internal state and its output length are 160 bits, which is substantially longer than MD5's 128 bits. Although its round functions are simpler and less varied than those of MD5, there are more of them—80 instead of 64. The lower speed of SHA-1 is a result of the message expansion, and the noticeably more complicated routine of each step than in MD5. Its higher security is paid by slower hashing rates, the difference is not unexpected.

SHA-1 uses a more complex procedure for deriving 32-bit sub blocks from the 512-bit message. If one bit of the message is flipped, more than a half of the sub blocks get changed.

CHAPTER V

SUMMARY AND FURTHER WORK

A hash function is a mechanism that maps strings of arbitrary length to strings of fixed length. This means that whether the input data is just a few words or the whole video file a few gigabytes long, the output of the function is always of the same length. There are many applications in which different kinds of hash functions are used, ranging from data structures such as hash tables through pattern-matching algorithms to checksum algorithms that help detecting accidental errors in data. They all rely on the fundamental property that most of the times different input values yield different output values, so the output of the hash function can be treated as a kind of a "fingerprint" of the input data that somehow identifies it. The dissertation shows cryptographic hash functions and their construction. The focus has been on explaining in great detail what hash functions are, where they can be used and how they are constructed. A test of the speeds of two cryptographic hash functions has shown that MD5 runs faster than SHA-1 on system with similar processors and architectures.

Hash functions are important because of their wide variety of applications. Digital signatures and MAC are the major and historical application of hash functions. Apart from digital signature some of the major applications of hash functions are data integrity, group signature, password table, digital watermarking, etc. Hash Algorithm is a subject of continued research as hash algorithm considered secure before10 years could be very weak in current time because of increasing computing speed.

Chapter VI
References

[1] Praveen Gauravaram. Cryptographic hash functions: Cryptanalysis, Design and Applications (Ph.D. Thesis, Queensland University of Technology, Brisbane, Australiya, 2007).

[2] Gary C. Kessler. An Overwiew of Cryptography. 30 June 2010.

[3] Murali Krishna Reddy Danda, Design And Analysis of Hash Functions, Victoria University, 2007

[4] X. Wang, A. Yao, F. Yao. "New Collision search for SHA-1". Crypto 2005.

[5] krystian Matusiewicz, Analysis of Modern Dedicated Cryptographic Hash Functions (Ph.D. Thesis, Macquarie University, 2007).

[6] Edward Schaefer, An introduction to cryptography and cryptanalysis Santa Clara University, 2005.

[7] Bart Preneel, Analysis and Design of Cryptographic Hash Functions, February 2003.

[8] Bert den Boer and Antoon Bosselaers. An attack on the last two rounds of MD4, 1991.

[9] Bart Preneel, CRYPTOGRAPHIC HASH FUNCTIONS, Katholieke Universiteit Leuven 24 August, 1994.

[10] Joseph Sterling Grah, Hash Functions in Cryptography, University of Bergen, June 1, 2008

[11] Michael Szydlo and Yiqun Lisa Yin, Collision Resistant use of MD5 and SHA-1 via Message preprocessing, 2006

[12] Tim Grembowski and Roar Lien, Comparative Analysis of the Hardware Implementation of Hash Functions.

[13] P. Rogaway and T. Shrimpton, Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance, 5 January 2004

[14] Diplomarbeit von Christian Knopf, Cryptographic Hash Functions, November 2007

[15] Bart Preneel, Design principles for dedicated hash functions, Katholieke University Leuven, 1994

[16] Free Software Foundation, The GNU C Library, Processor And CPU Time, CPU TimeInquiry,August2007. http://www.gnu.org/software/libc/manual/html_node/CPU-Time. Html

[17] William Feller, An Introduction to Probability Theory and Its Applications, Princeton University

[18] Eli Biham and Adi Shamir, Differential Cryptanalysis of DES-like Cryptosystems, The Weizmann Institute of Science, Department of Applied Mathematics, July 19, 1990

[19] Xiaoyun Wang and Hongbo Yu, How to Break MD5 and Other Hash Functions, Shandong University, Jinan 250100, China

[20] Marc Stevens, Arjen Lenstra, Benne de Weger: Vulnerability of software integrity and code signing applications to chosen-prefix collisions for MD5, Nov 30, 2007. Retrieved Jul 27, 2008.

[21] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, Finding Collisions in the Full SHA-1, Shandong University, Jinan 250100, China

[22] Dejan Jovanovi´c and Predrag Jani˘ci´c, Logical Analysis of Hash Functions, Verlag Berlin Heidelberg 2005

[23] Secure Hash Standard, FIPS PUB 180-3, National Institute of Standards and Technology, October 2008

http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf