NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE

DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

INTERSISCIPLINARY POSTGRADUADE DEGREE PROGRAM IN

“Management and Economics of Telecommunications Networks’’

MSc THESIS

Location Based Security in Mobile IoT

Dimitra K. Zisimopoulou

Supervisors: **Eustathios Hadjiefthymiades, Professor**

Anestis Papakotoulas, PhD Student

Athens

August 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΗ ΔΙΟΙΚΗΣΗ ΚΑΙ ΟΙΚΟΝΟΜΙΚΗ ΤΩΝ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΔΙΚΤΥΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Location Based Security in Mobile IoT

Δήμητρα Κ. Ζησιμοπούλου

Επιβλέποντες: **Χατζηευθυμιάδης Ευστάθιος ,** Καθηγητής

Ανέστης Παπακοτούλας, Υποψήφιος Διδάκτωρ

ΑΘΗΝΑ

Αύγουστος 2020

MSc THESIS

Location Based Security in Mobile IoT

Dimitra K. Zisimopoulou
S.N.: MOP519

Supervisors: **Eustathios Hadjiefthymiades, Professor**
Anestis Papakotoulas, PhD Student

August 2020

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Location Based Security in Mobile IoT

Δήμητρα Κ. Ζησιμοπούλου
Α.Μ.: ΜΟΠ 519

ΕΠΙΒΛΕΠΩΝ: **Χατζηευθυμιάδης Ευστάθιος ,** Καθηγητής
Ανέστης Παπακοτούλας, Υποψήφιος Διδάκτωρ

Αύγουστος 2020

ABSTRACT

Hash functions are important tool in information security over the internet. The cryptographic hash functions are versatile cryptographic building blocks which are used in many different security applications such as the protection of the authenticity of information and digital signatures. The main purpose for what they are designed is to improve message integrity. The first part of this thesis gives an overview of existing hash functions and the different methods of designing these, MAC algorithms and digital signatures. Next, we present hash techniques on Recent Hash functions and the constructions of them are examined where we focus mainly on the popular attacks on hash functions and their vulnerability results. After a research on papers relevant with the order preserving minimal perfect hash function, we describe an algorithm for generating order preserving minimal perfect hash function, involving generation of acyclic random graphs with a three- step algorithm. This method uses a Mapping, Ordering, Searching (MOS) approach and we will try to describe an algorithm that will capture the movement of a node in an encrypted environment, where each node can move in the three-dimensional space (x, y, z). The main purpose is the node to encryptly send the area in which it is located and the recipient to perceive, to where the node moved depending on the values received. One point of particular interest is the combination of random graphs which can generate a minimal perfect hash function with dynamic hashing for further future work.

SUBJECT AREA: cryptography, hash functions security analysis

KEYWORDS: message integrity, hash techniques, order preserving, minimal perfect hash functions, location based services.

ΠΕΡΙΛΗΨΗ

Οι συναρτήσεις κατακερματισμού (hash) είναι σημαντικό εργαλείο για την ασφάλεια πληροφοριών μέσω του Διαδικτύου. Οι κρυπτογραφικές hash συναρτήσεις είναι ευέλικτα κρυπτογραφικά δομικά στοιχεία που χρησιμοποιούνται σε πολλές διαφορετικές εφαρμογές ασφαλείας, όπως η προστασία της αυθεντικότητας των πληροφοριών και των ψηφιακών υπογραφών. Ο κύριος σκοπός για τον οποίο έχουν σχεδιαστεί είναι να βελτιώσουν την ακεραιότητα των μηνυμάτων. Το πρώτο μέρος αυτής της εργασίας παρέχει μια επισκόπηση των υπαρχόντων hash συναρτήσεων και των διαφορετικών μεθόδων σχεδίασης αυτών, αλγορίθμων MAC και ψηφιακών υπογραφών. Στη συνέχεια, παρουσιάζουμε τις hash τεχνικές πάνω στις πρόσφατες Hash συναρτήσεις και οι κατασκευές αυτών εξετάζονται με επίκεντρο κυρίως στις δημοφιλείς επιθέσεις των hash συναρτήσεων και στα αποτελέσματα ευπάθειας τους. Μετά από μια έρευνα σε δημοσιεύσεις που σχετίζονται με τις order preserving minimal perfect hash συναρτήσεις, περιγράφουμε έναν αλγόριθμο για τη δημιουργία ordering που διατηρεί την ελάχιστη perfect hash συνάρτηση και περιλαμβάνει τη δημιουργία ακυκλικών τυχαίων γραφημάτων με έναν αλγόριθμο τριών βημάτων. Αυτή η μέθοδος χρησιμοποιεί μια προσέγγιση χαρτογράφησης, ordering, αναζήτησης (MOS) και θα προσπαθήσουμε να περιγράψουμε έναν αλγόριθμο που θα συλλάβει την κίνηση ενός κόμβου σε ένα κρυπτογραφημένο περιβάλλον, όπου κάθε κόμβος μπορεί να κινηθεί στον τρισδιάστατο χώρο (x, y, ζ). Ο κύριος σκοπός είναι ο κόμβος να στείλει κρυπτογραφικά την περιοχή στην οποία βρίσκεται και ο παραλήπτης να αντιληφθεί, στο σημείο όπου ο κόμβος μετακινήθηκε ανάλογα με τις τιμές που λαμβάνονται. Ένα σημείο ιδιαίτερου ενδιαφέροντος είναι ο συνδυασμός τυχαίων γραφημάτων που μπορούν να δημιουργήσουν μια minimal perfect hash συνάρτηση με δυναμικό κατακερματισμό για περαιτέρω μελλοντικές εργασίες.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: κρυπτογραφία, ανάλυση ασφάλειας hash συναρτήσεων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: ακεραιότητα μηνυμάτων, hash τεχνικές, order preserving, minimal perfect hash συναρτήσεις, υπηρεσίες εντοπισμού θέσης.

ACKNOWLEDGEMENTS

It is a pleasure for me to thank all the people who have helped me to realize this Msc Thesis.

In the first place, I want to express my gratitude to Prof. Eustathios Hadjiefthymiades and PhD Student Anestis Papakotoulas, for being the promoters of this thesis. I'd like to express my special thanks to them for their consistent support and guidance during the running of this thesis. I appreciate the opportunity they gave me to finish this work and I feel very fortunate to work with some of the best scientists of the Department.

Finally, I must express my very profound gratitude to my parents Konstantinos and Sophia, my brother Harris and to my husband Akis for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Author

Dimitra Zisimopoulou

Στον πατέρα μου που με ενέπνευσε

&

Στη μητέρα μου που πίστεψε σε μένα

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The science and art of information security is Cryptography, and serves as a core for all secure communication and network information exchange. Cryptography, originally coming from two Greek words “κρυπτός” “kryptós” meaning "hidden or secret" and “γράφειν” “graphein” "to write". It is the study of these schemes that are used to convert the original plaintext into the corresponding ciphertext using a specific key. Encrypting the information is a way to keep it secure so that only an authorized recipient can extract and read the original plaintext. This allows messages to be sent without the sender worrying about contents becoming available to an unauthorized person as the information/content would be meaningless to someone who don’t be authorized.

Firstly, we will begin, describing the symmetric and asymmetric cryptography and the differences between them as an introduction before the cryptographic hash functions.

Cryptographic hash functions are important tool of modern cryptography. Their importance was first perceived with the invention of public key cryptography (PKC) by Diffie – Hellman and they became an important part of computer security.

These functions provide message integrity and they make sure that the receiver is receiving the content that the sender has already sent without the message has been modified. It is a mathematical operation that it is easy to perform, but extremely difficult to reverse. They take the arbitrary length input and produce a small output, called hash. It is designed to act as a one-way function, where you can put data into a hashing algorithm and get a unique string. The hash value can be described like a digital fingerprint of a message or a file, because two different messages cannot hash the same hash value. It is exactly like a person who has only one fingerprint and a small change in the message lead to totally change in the digest value. This is the reason why, nowadays, the hash functions are so useful for different applications such as digital signatures, password protection etc.

There is security properties that a hash function is expected to preserve in order to avoid attacks. Attacking a hash function means breaking one of the security properties of the hash functions, focusing on structure of them or on algorithm of compression function.

Many systems and application have to ensure in express access to information and objects in large network databases. When the fastest possible direct search is craved we usually apply hashing.

1.1 Motivation and objective

The main objective of this paper is to describe hash functions and more precisely to describe an order preserving minimal perfect hash function with a three- step algorithm for generating minimal perfect hash functions with the mapping- ordering- searching scheme. This algorithm captures the movement of a node which is moving in the three-dimensional space (x,y,z) in which it is located, and gives the opportunity to the recipient to perceive, to where the node moved depending on the values received. In order to achieve a framework with the above requirements we have to study the below:

- To study the basic cryptographic techniques
- To study the modern cryptographic techniques that are used in the networks
- To learn the concepts of transferring information securely
- To study the most well-known attacks on existing hash functions

- To study the vulnerability of the hash functions
- To study the order preserving minimal perfect hash functions
- To design and explain an order preserving minimal perfect hash function algorithm
- To study an algorithm which capture the movement of a node in an encrypted environment

1.2 Thesis structure and reading guide

The following table gives an overview of the research structure of this paper. We begin with a short introduction in field of cryptography and specifically in symmetric and asymmetric cryptography. We continue with the cryptographic hash functions, the hash techniques, and a presentation of the recent hash functions and their designs.

We present the main security properties, the most common attacks and the vulnerability of the hash functions.

Finally, our proposed framework about secure order preserving minimal perfect hash function algorithm is described.

Table 1: Thesis Chapter Structure

Chapter	Subsections	Overview
1. Introduction	Motivation and objective thesis structure reading guide thesis concept	An introduction of the concept and main objective of the paper and of its structure.
2. Introducing cryptography	Symmetric Asymmetric cryptography differences	An introduction to cryptography, basic elements and the differences between symmetric and asymmetric cryptography
3. Message integrity and digital signatures	Cryptographic hash functions Hash techniques Recent hash functions MAC Digital Signatures	A presentation of the cryptographic hash functions, their techniques and some recent trends. Presentation of the MAC and the Digital Signatures.
4. Hash functions security Analysis	Security properties Attacks on hash functions Vulnerability analysis	A presentation of the security properties, the most common attacks and the vulnerability analysis of them.
5. OPMPHF approaches	Order preserving minimal perfect hash function Related OPMPHF'S	A presentation of an order preserving minimal perfect hash function
6. Proposed Secure OPMPHF Algorithm	Mapping- Ordering- Searching Scheme The Proposed Algorithm Technical summaries for Privacy-preserving proximity- based security systems for location based services	Simulation of the proposed algorithm. A proximity- based authentication key generation strategy, without involving any trusted authority A dynamic privacy- preserving key management scheme
7. Conclusion	Conclusion and future work	Conclusion and future work

1.3 Thesis concept

In chapters 2-3 of this thesis we describe the properties and uses of hash functions in order to be efficient and we present the classification of hash functions, categorized them as keyed or unkeyed on the basis of the criterion whether they may or may not be the use of a key for designing a hash function. Secretly keyed hash functions are usually used to build Message Authentication Codes (MAC), with the canonical example is HMAC.

In the second part different modes of constructing a hash function are represented, such as Merkle- Damgård construction, HAIFA construction, Sponge construction, Wide- Pipe construction and Tree Based construction. It is also discussed few existing popular hash functions like MD5, SHA-1, SHA-2, BLAKE, Whirlpool etc. In addition, some of recent hash functions designs are described such as Whirlpool, JH Hash, Blake-256, Blake2,SHA-3 (Keccak), Streebog and KangarooTwelve, that are using the new modified design architecture.

We provide a discussion about the three classical hash function security properties in chapter 4, which are collision resistance, pre- image resistance and second pre- image resistance, explaining the goals of security and the most common attacks that exist on hash functions and the vulnerability of them.

In the chapters 5-6 we discuss about minimal perfect hash functions that preserving the order of the key, and we describe an algorithm for finding an order preserving minimal perfect hash function with the use of Mapping, Ordering, Searching Scheme (MOS) where a node send the area in which it is located and the recipient to perceive, to where the node moved depending on the values received.

In chapter 6, we describe a proximity- based authentication key generation strategy, without involving any trusted authority, pre-shared secret or public key infrastructure for mobile users in Wireless Networks and a dynamic privacy- preserving key management scheme for location – based services in vehicular ad hoc networks (VANETs).

2 INTRODUCING CRYPTOGRAPHY

2.1 Overview

The importance of information and communication systems for society globally is intensifying with the increasing value and quantity of data that is transmitted and stored on those systems. Those systems and data are also increasingly vulnerable to a variety of threats, such as unauthorized access and use, misappropriation, alteration and destruction. To hide any data and keep safe information, the technique that is mainly used is Cryptography.

Cryptography is the science of secret writing with the goal of hiding the meaning of the message [1]. In addition, it can be stated that is the science of protecting data and provide methods of converting data into an unreadable form, so that only the valid user can access information with the using of mathematics to encrypt and decrypt data [2].

Cryptography separated into three main sections, Symmetric Ciphers, Asymmetric Ciphers and Cryptographic Protocols. Symmetric and Asymmetric Ciphers can be treated as constituent elements wherewith secure internet communication can be achieved. As a cryptographic protocol we can quote as an example the Transport Layer Security (TLS) which is used in every Web browser [3].

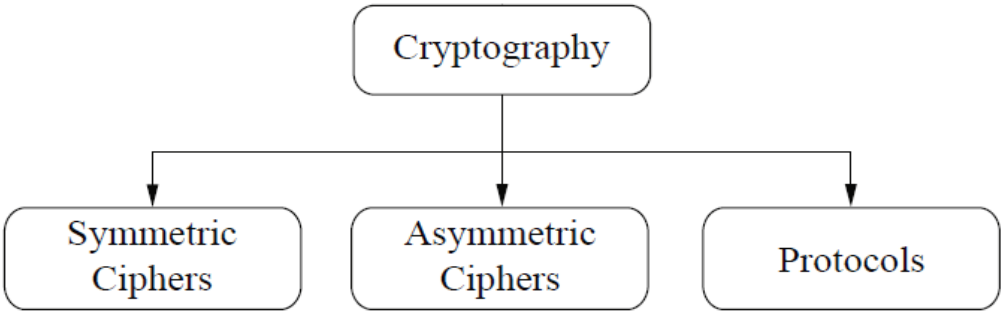


Figure 1: Overview of the field of Cryptography [3]

Prior to the beginning we describe some terms that are commonly used in the science of cryptography, as we describe below:

Plaintext is called the information we need to hide and it is about the original text which it could be in a form of characters, numerical data or any kind of information. For example the plaintext is the sending of a message in the sender before encryption, or it is the text at the receiver after decryption [2].

Cipher text is called the data that will be transmitted. It's a term refers to the string of 'meaningless' data, or unclear text that nobody must understand, except the recipients [2].

Cipher is the algorithm that is used to transform plaintext to cipher text. This method is called **encryption**, stating differently, it is a procedure of converting readable and understandable data into a 'without meaning' data [2].

The **key** is an input to the encryption algorithm, and this value must be independent of the plaintext. This input is used to transform the plaintext into cipher text, so different keys will produce a different cipher text. In the decipher side, the inverse of the key will

be used inside the algorithm instead of the key. There are two different types of keys the private key and the public key [2].

Computer security is a generic term for a collection of tools designed to protect any data from hackers, corruption or natural disaster while allowing these data to be available to the users at the same time. The NIST Computer Security Handbook defines the term as follow: 'The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability, and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications)' [4].

Network security refers to any activity designed to protect the usability, integrity, reliability and safety of data during their transmission on a Network [2].

Internet security is measures and procedures used to protect data during their transmission over a collection of interconnected networks, while information security is about how to prevent attacks, and to detect attacks on information- based systems [3].

Thanks to the use of cryptography many goals can be accomplished. These goals can be either all achieved at the same time in one application or only one of them. These goals are assigned as below:

Confidentiality: it is the main focus, that ensures that nobody can understand the received message except the one who has the decipher key [2].

Authentication: is the process of providing the identity that assures the communicating entity is the one that it claimed to be. This implies that the user or the system can prove their own identities to other parties who do not have personal knowledge of their identities [2].

Data Integrity: ensures that the received message has not been changed in any way from its original form. The data may get modified by an unauthorized entity intentionally or accidently. Integrity service confirms that whether data is intact or not since it was last created, transmitted, or stored by an authorized user. This can be achieved by using hashing at both sides the sender and the recipient in order to create a unique message digest and compare it with the one that received [2].

Non-Repudiation: it is a mechanism used to prove that the sender really sent this message, and the message was received by the specified party, so the recipient cannot claim that the message was not sent. For example, once an order is placed electronically, a purchaser cannot deny the purchase order, if non-repudiation service was enabled in this transaction [2].

Access Control: it is the process of preventing an unauthorized use of resources. This goal controls who can have access to the resources, if one can access, under which restrictions and conditions the access can be occurred, and what is the permission level of a given access [2].

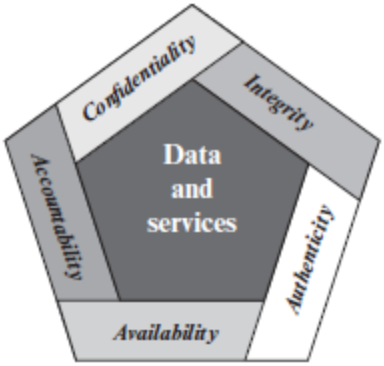


Figure 2: Computer security requirements [2]

2.2 Symmetric Cryptography

Symmetric cryptography is also referred as symmetric- key, secret key and single- key cryptography. In symmetric key cryptography a secret key may be held by one person or exchanged between the sender and the receiver of a message. If private key cryptography is used to send secret messages between two parties, both the sender and the receiver must have a copy of the secret key.

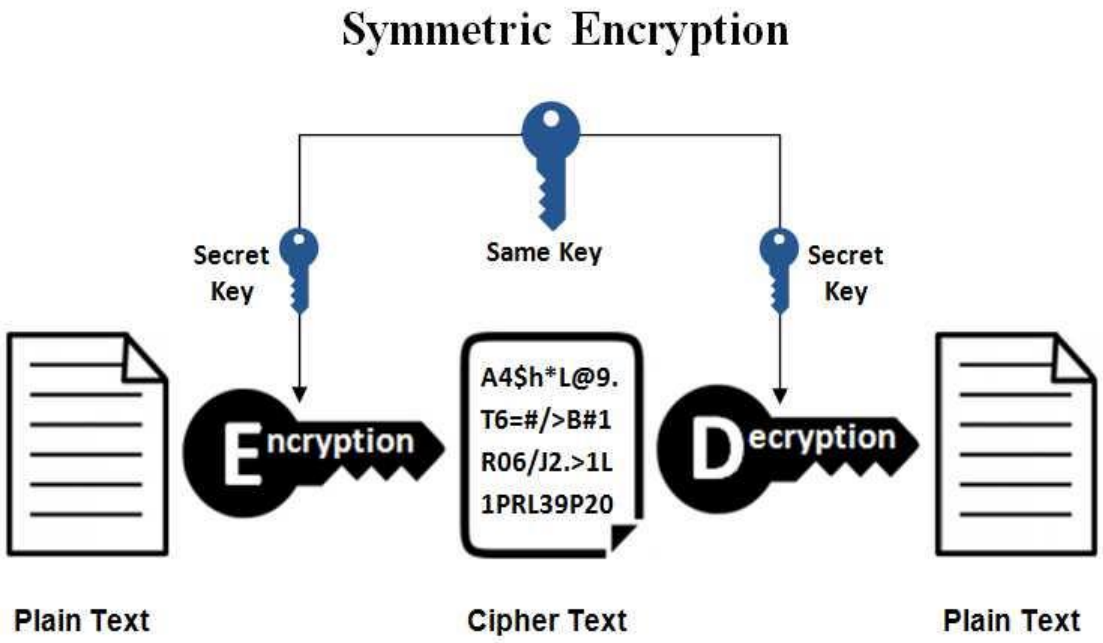


Figure 3: Symmetric Cryptosystem [2]

A Symmetric encryption scheme has five components Plaintext, Encryption Algorithm, Secret Key, Cipher-text and Decryption Algorithm. The Secret Key is shared by both, the sender and the receiver which they must have obtained in a secure fashion & should keep the key hidden, lest anyone who finds the key would be able to extract the hidden message. The Symmetric encryption was the only type of encryption in use prior to the development of Public-Key encryption in the 1970s. It remains by far the most widely used of the two types of encryption [2]. It was in use way before the computer era, and can be traced back to the ancient Rome & Egypt. The ciphers which were in use before the advent of the computers are termed as the classical encryption

algorithms. They were all very intriguing & worked on texts, but now we have bits & bytes [3].

There is a risk with hidden messages that somebody finds the hidden message and exposes it to unfriendly parties. This is especially the case for regular communications. Eventually the idea arose to encrypt messages so that they cannot be read even if intercepted and this has led to the development of cryptography. Ciphers allow parties to securely communicate by encrypting their messages with some secret knowledge (the secret key) into unreadable cipher text that can only be read by parties that possess the same secret knowledge. This form of encryption using a single secret key known to both sender and receiver is called symmetric encryption. [2]

Authentication of messages, the act of confirming that a message really has been sent by a certain party, usually was achieved by a combination of inspecting the message (e.g. verify its signature). Symmetric encryption also provides some form of authentication. After all, no one else knows the secret key and is able to encrypt messages with it. However, this does not prevent the encrypted message from being purposely changed or simply repeated at an opportune moment by unfriendly parties. In general, authentication of the sender of something is achieved through mutual knowledge (such as a secret password), possession of a physical token (such as the king's seal) and/or distinguished marks (such as a known birthmark) [3],[2].

2.2.1 Symmetric-key cryptography ciphers

There are two generous classes of Symmetric encryption techniques **block ciphers** or **stream ciphers.**

A **block cipher** enciphers input in blocks of plaintext as opposed to individual characters, the input form used by a stream cipher. A block cipher is a deterministic algorithm operating on fixed-length groups of bits, called blocks, with a uniform transformation that is specified by a symmetric key. Block ciphers operate as important elementary components in the design of many cryptographic protocols, and are widely used to implement encryption of bulk data. Nowadays they are used in many secure Internet protocols, including PGP (for secure e-mail), SSL (for securing TCP connections), and IPsec (for securing the network-layer transport) [5].

The following diagram describes block ciphers based general construction of a compression function for hash functions:

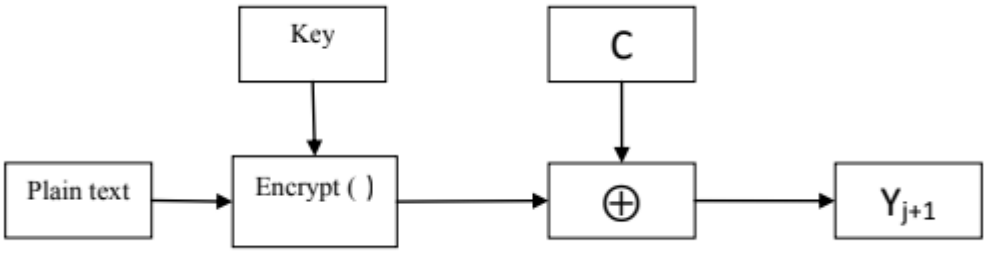


Figure 4: Simple design of block cipher based hash functions compression function [5]

Nowadays there are a number of well-known block ciphers, including DES (standing for Data Encryption Standard), 3DES, and AES (standing for Advanced Encryption Standard). Each of these algorithms also uses a string of bits for a key.

For example, DES, which stands for Data Encryption Standard, used to be the most popular block cipher worldwide. The DES algorithm became a standard in the US in 1977. However, it's already been proven to be vulnerable to brute force attacks and other cryptanalytic methods. DES is a 64-bit cipher that works with a 64-bit key. Actually, 8 of the 64 bits in the key are parity bits, so the key size is technically 56 bits long.

AES or Advanced Encryption Standard is the most widely used block cipher in the world. It has a block size of 128 bits and supports three possible key sizes - 128, 192, and 256 bits. The longer size of the key, the stronger the encryption. However, longer keys also result in longer processes of encryption [5].

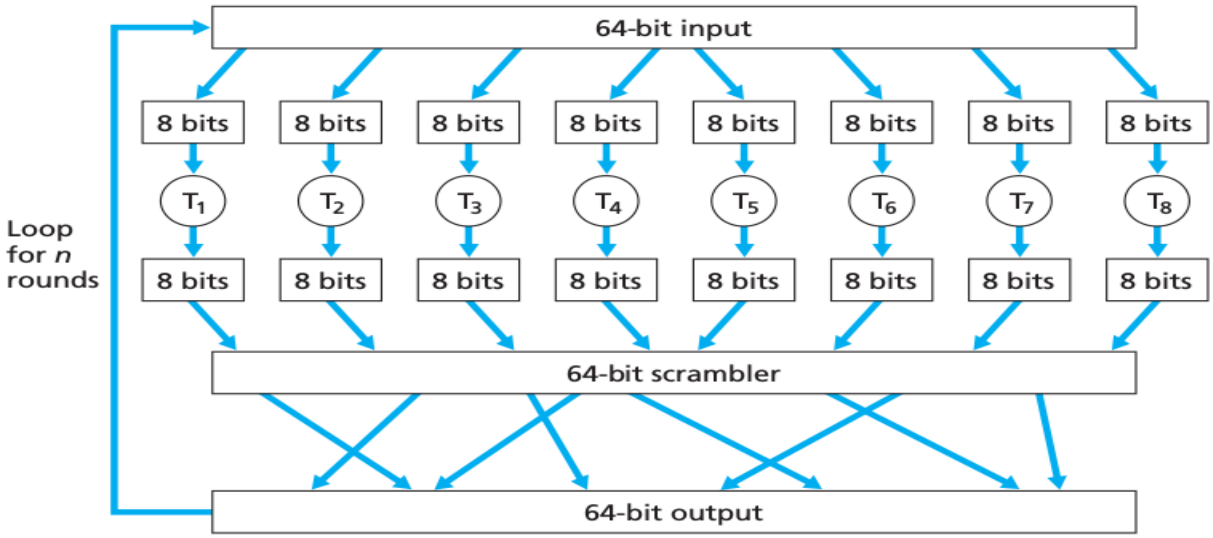


Figure 5: An example of block cipher [5]

A **stream cipher** is a symmetric key cipher where plaintext digits are combined with a pseudorandom cipher digit stream (keystream). In a stream cipher, each plaintext digit is encrypted one at a time with the corresponding digit of the keystream, to give a digit of the cipher text stream. Since encryption of each digit is dependent on the current state of the cipher, it is also known as state cipher. In practice, a digit is typically a bit and the combining operation an exclusive-or (XOR). Stream ciphers are designed to approximate an idealized cipher, known as the One-Time Pad [5].

The One-Time Pad, can potentially achieve "perfect secrecy" and it's supposed to be fully immune to brute force attacks. But there is a problem with the one- time pad. The problem is that, in order to create such a cipher, its key should be as long or longer than the plaintext [5].

RC4, which stands for Rivest Cipher 4, also known as ARCFOUR or ARC4, is the most widely used of all stream ciphers. RC4 steam chiphers have been used in various protocols like WEP and WPA (both security protocols for wireless networks) as well as in TLS. Although recent studies have revealed vulnerabilities in RC4, prompting Mozilla and Microsoft to recommend that it be disabled where possible [5].

2.3 Asymmetric Cryptography

Whitfield Diffie and Martin Hellman, researchers at Stanford University, first publicly proposed asymmetric encryption in their 1977 paper, "New Directions in Cryptography".

Asymmetric cryptography, also known as public key cryptography, uses two different keys to encrypt and decrypt data, the public key and the private key. The keys are simply large numbers that have been paired together but are not identical (asymmetric). The public key can be shared with everyone and the private key is kept secret. In the two-key system is also known as the public key system, one key encrypts the information and another, mathematically related key decrypts it. The computer sending an encrypted message uses a chosen private key that is never shared and so is known only to the sender. If a sender first encrypts the message with the intended receiver's public key and again with the sender's secret private key, then the receiver may decrypt the message, first using its secret key and then the sender's public key. Using this public-key cryptographic method, the sender and receiver are able to authenticate one another as well as protect the secrecy of the message.

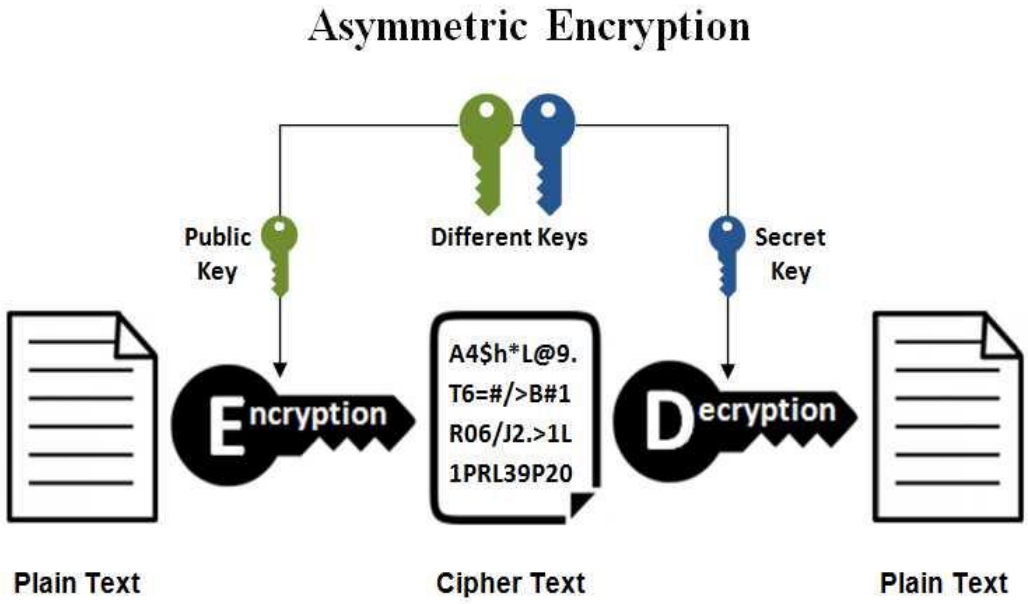


Figure 6: Asymmetric Cryptosystem [2]

The two participants in the asymmetric encryption are the sender and the receiver. First, the sender obtains the receiver's public key. Then the plaintext is encrypted with the asymmetric encryption algorithm using the recipient's public key, creating the ciphertext. The ciphertext is then sent to the receiver, who decrypts the ciphertext with his private key so he can access the sender's plaintext. Because of the one-way nature of the encryption function, one sender is unable to read the messages of another sender, even though each has the public key of the receiver.

So, securely sending a message to someone is possible without first exchanging a secret key in advance anymore, as you simply encrypt your message with his public key as found in the listing. Only a specific participant can now decrypt the cipher text using the private key known only to him. This invention, builds on the fact that no one can derive the private key from the public key [6].

2.3.1 RSA algorithm

RSA (Rivest-Shamir-Adleman) is the most common used asymmetric algorithm. This is fixed in the SSL/TSL protocols which are used to provide communications security over a computer network. RSA derives its security from the computational difficulty of

factoring large integers that are the product of two large prime numbers. Multiplying two large primes is easy, but the difficulty of determining the original numbers from the product forms the basis of public key cryptography security. The time it takes to factor the product of two sufficiently large primes is considered to be beyond the capabilities of most attackers, excluding nation-state actors who may have access to sufficient computing power. RSA keys are typically 1024- or 2048-bits long, but experts believe that 1024-bit keys could be broken in the near future, which is why government and industry are moving to a minimum key length of 2048-bits [7].

RSA makes extensive use of arithmetic operations using modulo-n arithmetic. Recall that x mod n simply means the remainder of x when divided by n. In modular arithmetic, one performs the usual operations of addition, multiplication, and exponentiation. However, the result of each operation is replaced by the integer remainder that is left when the result is divided by n. Adding and multiplying with modular arithmetic is facilitated with the following handy facts:

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$$

$$[(a \bmod n) – (b \bmod n)] \bmod n = (a – b) \bmod n$$

$$[(a \bmod n) \cdot (b \bmod n)] \bmod n = (a \cdot b) \bmod n$$

It results from the fact that (a mod n) d mod n = ad mod n.

There are interrelated components of RSA:

- The choice of the public key and the private key
- The encryption and decryption algorithm.

To generate the public and private RSA keys, we have the following steps:

1. Choose two large prime numbers, p and q. The larger the values, the more difficult it is to break RSA, but the longer it takes to perform the encoding and decoding.

2. Compute n = p q and z = (p – 1)* (q – 1).

3. Choose a number, e, less than n, that has no common factors (other than 1) with z. (In this case, e and z are said to be relatively prime.) The letter e is used since this value will be used in encryption.

4. Find a number, d, such that ed– 1 is exactly divisible (that is, with no remainder) by z. The letter d is used because this value will be used in decryption. Put another way, given e, we choose d such that ed mod z = 1

5. The public key is the pair of numbers (n, e) and the private key is the pair of numbers (n, d).

The encryption by A and the decryption by B are done as follows:

• Suppose A wants to send B a bit pattern represented by the integer number m (with m < n). To encode, A performs the exponentiation m^e, and then computes the integer remainder when me is divided by n. In other words, the encrypted value, c, of A's plaintext message, m, is $c = m^e \bmod n$

The bit pattern corresponding to this ciphertext c is sent to B.

• To decrypt the received ciphertext message, c, B computes $m = c^d \bmod n$ which requires the use of his private key (n,d) [5], [7].

2.3.2 Elliptic curve Cryptography

Elliptic curve Cryptography (ECC) is an alternative to RSA for implementing public key cryptography. ECC is a public key encryption technique based on elliptic curve theory that can create faster, smaller, and more efficient cryptographic keys. ECC generates keys through the properties of the elliptic curve equation. To break ECC, we have to compute an elliptic curve discrete logarithm, and it turns out that this is a significantly more difficult problem than factoring. As a result, ECC key sizes can be significantly smaller than those required by RSA yet deliver equivalent security with lower computing power and battery resource usage making it more suitable for mobile applications than RSA.

The typical application for asymmetric cryptography is authenticating data through the use of digital signatures. Based on asymmetric cryptography, digital signatures can provide assurances of evidence to the origin, identity and status of an electronic document, transaction or message, as well as acknowledging informed consent by the signer. The SSL/TSL cryptographic protocols for establishing encrypted links between websites and browsers also make use of asymmetric encryption [7].

2.4 Differences between Symmetric and Asymmetric Cryptography

The main difference between these two methods of encryption is that asymmetric encryption algorithms makes use of two different keys, one key to encrypt the data and another key to decrypt it while symmetric encryption uses the same key to perform both the encryption and decryption functions.

Second difference is the length of the keys. In symmetric cryptography, the length of the keys is typically set at 128-bits or 256-bits, depending on the level of security that's needed. However, in asymmetric encryption, there has to be a mathematical relationship between the public and private keys. Asymmetric keys need to be much longer to offer the same level of security. The difference in the length of the keys is so pronounced that a 2048-bit asymmetric key and a 128-bit symmetric key provide just about an equivalent level of security. Finally, asymmetric encryption is slower than symmetric encryption, which has a faster execution speed.

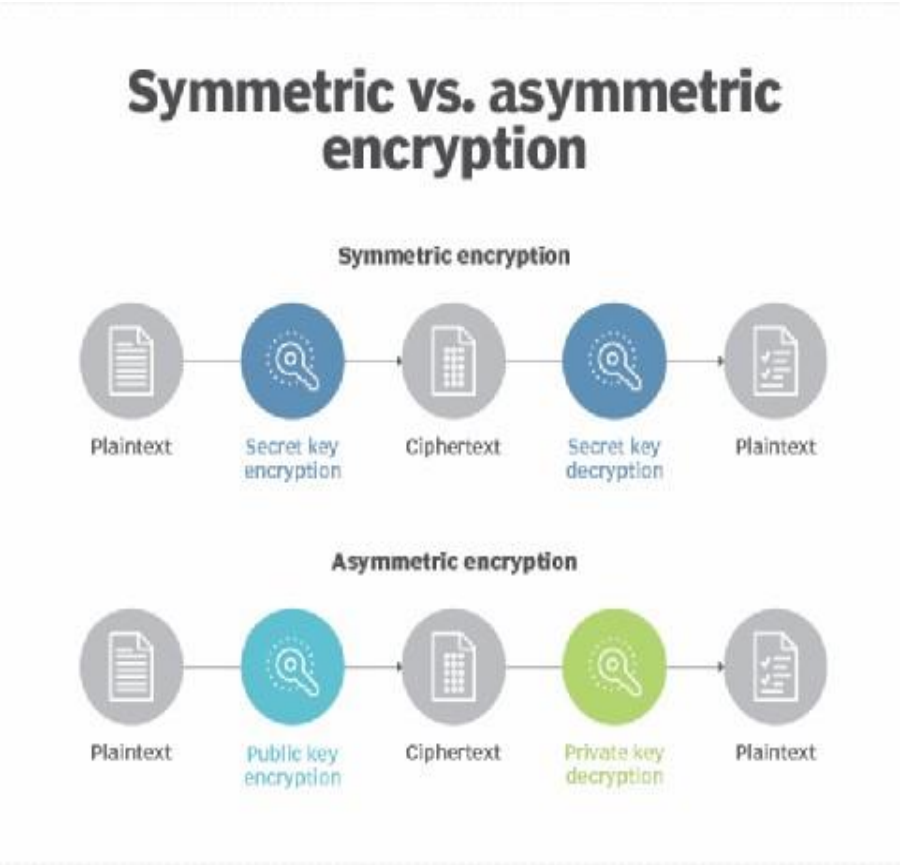


Figure 7: Symmetric vs Asymmetric encryption [8]

Generally we can summarize the differences between the symmetric and asymmetric encryption to the table below:

Table 2: the differences between the symmetric and asymmetric encryption

Symmetric Encryption	Asymmetric Encryption
The same algorithm and the same key is used for encryption and decryption.	One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption.
The sender and the receiver must share the algorithm and the key.	The sender and the receiver must each have one of the matched pair of keys (not the same one)
The key must be kept secret.	One of the two keys must be kept secret.
Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	Knowledge of the algorithm plus samples of ciphertext plus one of the keys must be insufficient to determine the other key.

3 MESSAGE INTEGRITY AND DIGITAL SIGNATURES

An equally important cryptography topic is the message integrity also known as message authentication. In this chapter, we will describe a popular message integrity technique that is used by many secure networking protocols. In addition we have to discuss another important topic in cryptography, the cryptographic hash functions [5].

3.1 Cryptographic Hash Functions

Hashing is a method of cryptography that converts any form of data into a unique string of text. It is a mathematical operation that is easy to perform, but extremely difficult to reverse. These functions take an arbitrary length input and produce a small output. This output is known as message digest or hash code or simply hash. Any piece of data can be hashed, regardless of the data's size, type or length. It is designed to act as a one-way function, where you can put data into a hashing algorithm and get a unique string. The hash value can be thought like a digital fingerprint of a message or file, because two different messages/ files cannot have the same hash value. It is exactly the same like a person who has only one unique fingerprint. The hash output depends on each character of input, so a small change in the message will lead to totally different digest value. This message digest is treated as a signature of that message.

For this reason hash functions are an indispensable tool in different types of applications such as digital signatures, Pseudo- Random Functions, Message Authentication, Data Integrity, password protection etc. [9], [10].

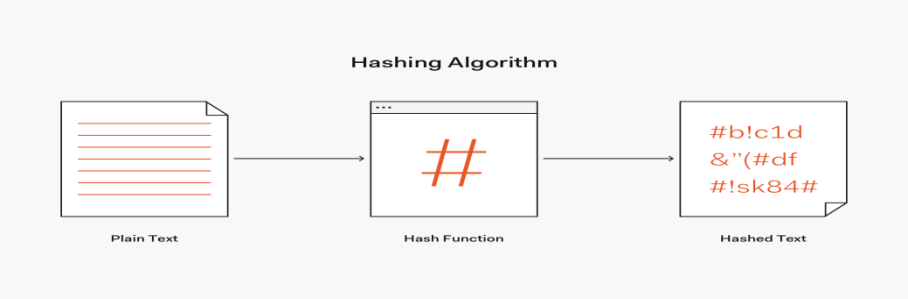


Figure 8: Working Mechanism of One Way Hash Function [9]

A generic cryptographic hash function has two inputs: the message it's going to compress or hash (x) & a public key (s) that represents the fixed-length output of our hash in alphanumeric characters. Our hashed result is termed the message digest or simply digest (x*).

This looks like the following:

$$H(s,x) = x^*$$

3.1.1 History

Commonly used hashing algorithms include Message Digest (MDx) algorithms, such as MD5, and Secure Hash Algorithms (SHA), such as SHA-1 and the SHA-2 family that includes the widely used SHA-256 algorithm. The recent attacks on MD4, MD5, SHA-0 and SHA-1 have enforced research in designing new cryptographic hash functions of existing ones. A lot of issues were announced about techniques for efficiently collisions

in MD5 and SHA-1 , and it is nowadays clear that the MD5 and SHA-1 are not as strong as we need and there was a requirement to shift towards in new hash functions with improved designs [11], [12].

The way of the new approach was based on designing new hash functions from scratch. The hash functions, like SHA-1 and MD5 have been for many years the most famous until 2005 when Wang et al. [13] found that collisions for MD5 can reducing the effort to find collisions on SHA-1 to 269 [13], [12].

Although a break with complexity of 269 is theoretical, it showed that SHA-1 is not as strong and collision-resistant as it is supposed to be. That's the reason why the National Institute of Standards and Technology (NIST) announced an open competition in order to select a new hash functions standard, to be named SHA-3 [14].

For this competition the 5 finalist candidates were BLAKE, Grøstl, JH, Keccak and Skein and the competition ended on October 2, 2012 when the NIST announced that Keccak would be the new SHA-3 hash algorithm [15].

As a consequence of this competition, both the theory and practice of hash functions will make a significant step forward [16].

3.1.2 Properties and uses of hash functions

Hashing algorithms must have the following properties in order to be efficient: They are deterministic, meaning that the same message always results in the same hash.

It is quick to compute the hash value for any given message.

it is practically infeasible to generate a message that yields a given hash value

a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value

It is infeasible to find two different messages with the same hash value

It must have very low probability of collisions.

Nowadays, hash functions are used for many different purposes. The average user encounters hashing daily in the context of passwords. In crypto currency blockchains today, we use hashing in order to write new transactions, timestamp them and add a reference to them in the previous block. For example, running a decentralized network such as Bitcoin requires both trustlessness and verification efficiency. A critical part of their security involves being able to compress large chunks of information into a short message standard, which can be efficiently verified if need be, known as hash [11], [12].

File verification

An important application of secure hashes is verification of message integrity. When we compare the hash digests over the message before and after calculation, transmission can determine whether any changes have been made to the message or the file.

Password verification

Password verification relies on cryptographic hashes. If we store all user passwords as clear text, this can result in a massive security breach if the password file is compromised. A way that the danger can be reduced is to store the hash digest of each password. For the authentication of a user, the password presented by the user is hashed and then compared with the stored hash. A password hash requires the use of a large random, non- secret salt value which can be stored with the password hash. The salt randomizes the output of the password hash, making it impossible for an adversary to store tables of passwords and precomputed hash values to which the password hash digest can be compared.

Digital Signature

The purpose for which cryptographic hash functions were originally designed is input preparation for digital signatures. The message is compressed using a hash function and the fingerprint is the input to the digital signature algorithm. The message is considered authentic if the signature verification succeeds given the signature and recalculated hash digest over the message. So the message integrity property of the cryptographic hash is used to create secure and efficient digital signature schemes. The attack of the digital signature can be happened if the hash function can be abused.

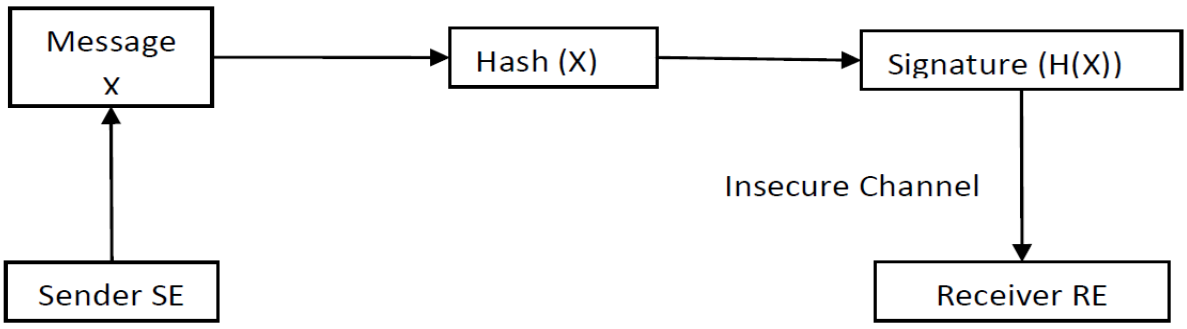


Figure 9: Use of Hash Function in Digital Signature [9]

Message Authentication Codes (HMAC)

An extra use of hash functions is for the authentication of high- speed message between parties who share a common secret. This can be done with the use of HMAC framework, where H is the hash function, K is the shared secret and M is the message to be authenticated [9].

$$HaFu(K1 + output\text{-}_{padding}, HaFu(K1 + inpu, X))$$

Where HaFu is the hash function

K1 is the secret key shared between sender and receiver

X is the message to be authenticated

Pseudo- Random Functions

Hash functions are often used as pseudo- random functions. They provide a deterministic mechanism for generating random- seeming bit streams from some input source without any information about the input. After a Diffie- Hellman exchange, a typical use is generating cipher keying material [17].

Data Fingerprinting

Hash functions can be used to produce fingerprints generally. Instead of digitally signing these fingerprints, the values are stored separately from the data and this permits later detection of changes to the original data [18], [19], [20].

3.1.3 Keyed and Unkeyed Hash Functions

Hash functions can be classified as Keyed or Unkeyed, based on whether the hash function is using a type of key or not in the processing. [21].

Keyed hash functions make use of a key in the process of generating a hash value.

Therefore, these functions require two specific inputs: (1) a message of arbitrary finite-length, and (2) a key of specific length. The fundamental approach behind this is that, if adversary does not know the key, he must not be able to forge the message. Such type of hash functions are also known as Message Authentication Codes (MAC). Output of MAC depends on both – the message and the key [22].

The definition of keyed hash functions is below:

"The map HASH :$\{0,1\}^* \times \{0,1\}^n \rightarrow \{0,1\}^m$ is said to be a keyed hash function with m -bit output and n -bit key if H is a deterministic function that takes two inputs, the first of an arbitrary length, the second of n -bit length and outputs a binary string of length m -bits. Where both n, m are positive integers. $\{0,1\}^m$ and $\{0,1\}^n$ are the sets of all binary strings of length m and n respectively and $\{0,1\}^*$ is a set of all finite binary strings. Keyed hash function or MACs are majorly concerned with message integrity and source authentication both"[23].

Unkeyed hash functions do not use any key as input to generate hash value. The majority of hash functions are unkeyed hash functions. By appending the digest to the message during the transmission, these hash functions are used for error detection. The error can be diagnosed if the digest of the received message at the receiving end is not equal to the received message digest.

The definition of unkeyed hash functions is below:

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

The Unkeyed hash functions may further be classified into categories, named as- One-

Way Hash Function (OWHF) and Collision Resistant Hash Function (CRHF). But, still a hash function must possess both qualities- one-way and collision resistance"[23].

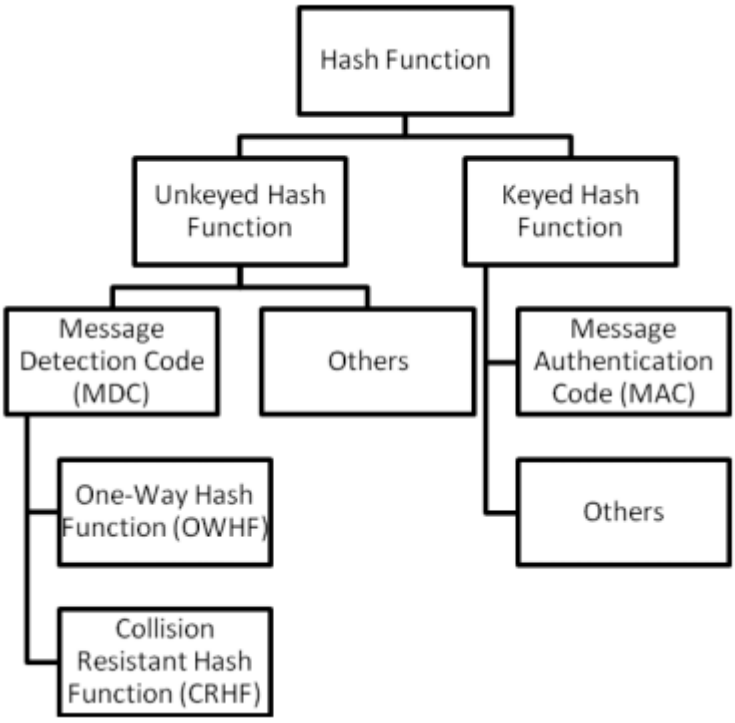


Figure 10: Simplified Broad Categories of Cryptographic Hash Function [24]

3.2 Hash Techniques

In this section many construction methods are explained. Hah functions that we use today are based on these constructions methods and we will describe and analyze the design of them.

3.2.1 Design of new Hash Functions

3.2.1.1 *Merkle–Damgård construction*

In 1989, The Merkle–Damgård construction was described by Ralph Merkle and Ivan Damgård, who independently proved that if an appropriate padding scheme is used and the compression function is collision-resistant, then the hash function will also be collision-resistant too [23]. A hash function built with the Merkle–Damgård construction is as resistant to collisions as is its compression function. Any collision for the full hash function can be traced back to a collision in the compression function. In order to make the construction secure, Merkle and Damgård proposed that messages be padded with a padding that encodes the length of the original message. Firstly, applies an MD-compliant padding function to create an input whose size is a multiple of a fixed number. The hash function then breaks the result into blocks of fixed size, and processes them one at a time with the compression function, each time combining a block of the input with the output of the previous round. The last block processed should also be unambiguously length padded. Most common hash functions, including SHA-1, MD5, JH- Function, Streebog, take this form [25].

Figure 11: The Merkle- Damgård hash construction [23]

Algorithm MD^f

$M \rightarrow M_1 \cdots M_l$

$y_0 = IV$

for i = 1 to l do

$y_i = f(M_i, y_{i-1})$

return y_l

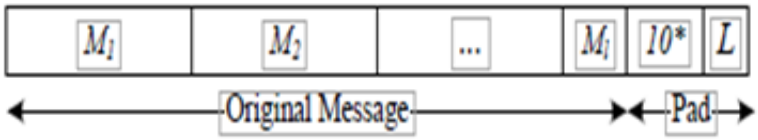


Figure 12: Merkle- Damgård Padding algorithm [23]

Algorithm $Pad_s(M)$

$d = M + 1 + 64 \, mod \, m$

$M \| 1 \| 0^d \langle M \rangle_{64} \rightarrow \widehat{M}$

$\widehat{M} \rightarrow M_1 \cdots M_l$

Figure 12 illustrates the padding algorithm, where L is a 64-bit encoding of the the length of the message and m is the length of a single block. The message is then iterated repeatedly by calling a Fixed- Input-Length (FIL) compression function $f : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$ accepting two inputs: a message block M_i (of length m) and either an

Initialisation Vector IV (when hashing the first block) or a chaining variable (which is the output of the previous f call), both of length n.

Message Digest 5 (MD5)

MD5 is a hash function designed by Ronald L. Rivest in 1992 as a more strength version of MD4. Taking an arbitrary length input message, the MD5 produces a single output of 128-bit length message digest. The input message is divided to multiple blocks of 512 bits each [26].

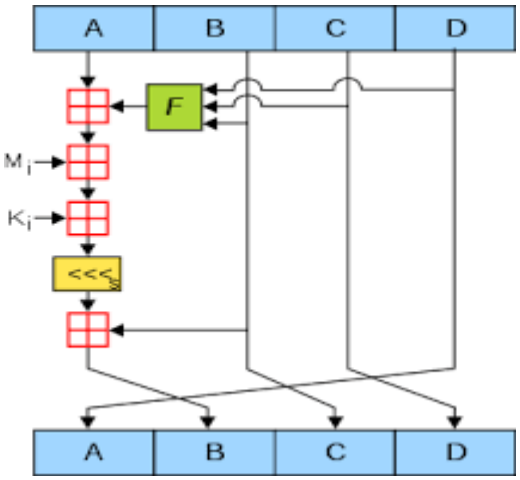


Figure 13: MD5 Hash Function 80]

Secure Hash Function 1 (SHA-1)

Secure Hash Algorithm (SHA-1) is based on MD4 , and was proposed by the U.S. National Institute for Standards & Technology (NIST) in 1995 for certain U.S federal government applications. The SHA-1 produces a single 160-bit length output from an arbitrary length input message. The input message is divided to multiple blocks each of 512 bits. Each message block is represented as a sequence of sixteen 32-bit words [27].

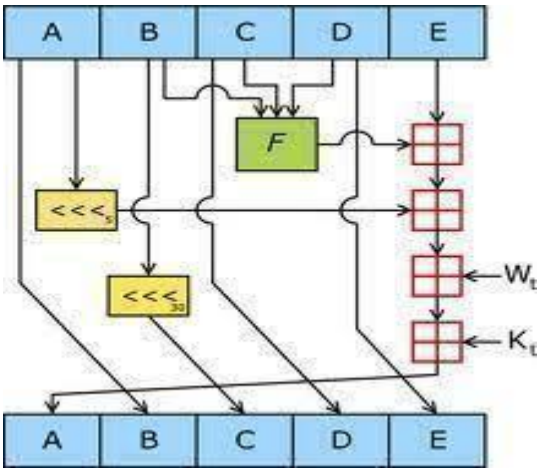


Figure 14: SHA-1 Hash Function [80]

Secure Hash Function 2 (SHA-2)

SHA-2 family is a set of Cryptographic Hash Function (SHA-224, SHA-256, SHA- 384, SHA-512, SHA-512/224 and SHA-512/256) designed by the U.S. National Security Agency (NSA). SHA-2 consists of a set of six hash functions with digests that are 224, 256, 384 or 512 bits. For SHA-224 and SHA- 256, each message block has 512 bits, which are represented as a sequence of 32-bit words. For SHA-384 and SHA-512, each message block has 1024 bits, which are represented as a sequence of 64-bit words. SHA-224 and SHA-256 operate on 32-bit words and SHA-384 and SHA-512 operate on 64-bit words. SHA-256 and SHA-512 are novel hash functions which use different shift amounts and additive constants, but their structures are otherwise virtually identical, differing only in number of rounds. SHA-224 and SHA-384 are simply the truncated versions ofSHA-256 and SHA-512 respectively. SHA-512/224 and SHA-512/256 are also truncated version of SHA-512 but the initial values are generated using the method described in FIPS PUB 180-4 [27].

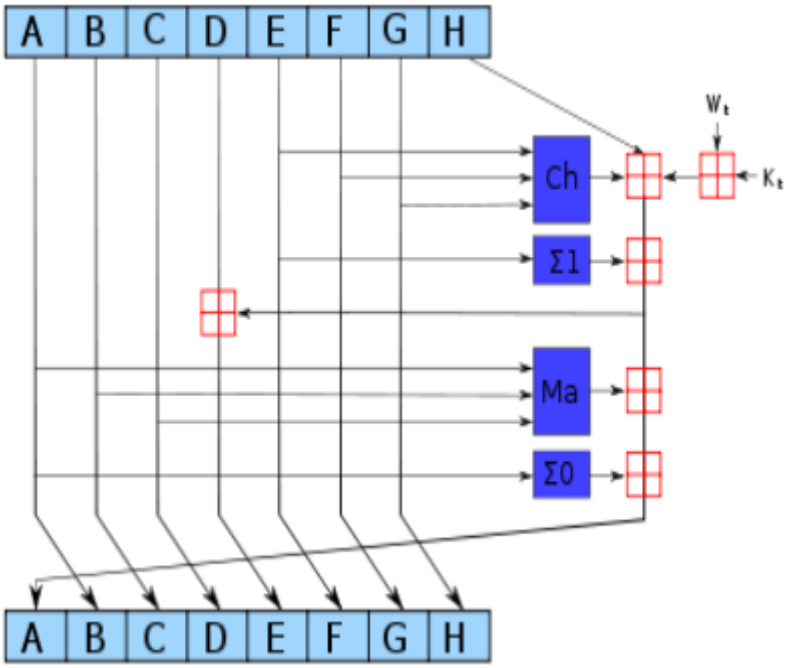


Figure 15: SHA-256 Hash Function [80]

3.2.1.2 *HAIFA construction*

The HAIFA construction (hash iterative framework) is a cryptographic structure designed by Eli Biham and Orr Dunkelman in 2007. It is one of the modern alternatives to the Merkle–Damgård construction, avoiding its weaknesses like length extension attacks. For example Blake-256 takes this form [28].

HAIFA modifies Merkle-Damgård by introducing extra input parameters to the compression function. Those parameters are a salt value (used as a key to create families of hash functions - if only one hash function is needed, the salt is set to 0), and the number of bits hashed so far. In fact, HAIFA can be considered a dedicated-key hash function [29]. The idea of adding additional input parameters to the compression function has been previously proposed by Rivest through a process called dithering [30], though a second pre-image attack against dithered hash functions was reported by

Andreeva et al. in [31]. An obvious drawback of HAIFA is efficiency degradation since the compression function now has more input parameters to process. Furthermore, HAIFA cannot be (easily) used to patch existing Merkle-Damgård based hash functions because a compression function designed for the Merkle-Damgård construction would not naturally accommodate the extra HAIFA parameter inputs.

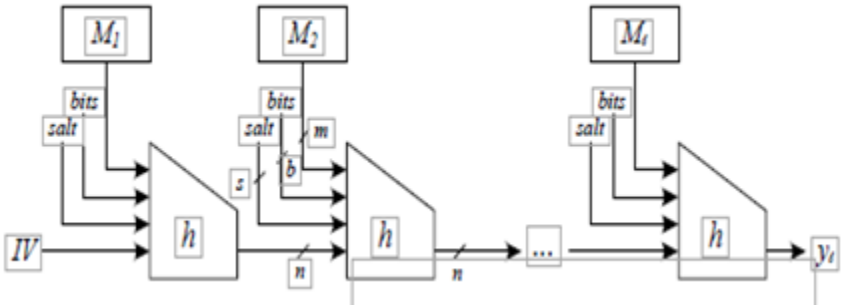


Figure 16: The HAIFA construction [29]

Algorithm HAIFA$_s^h$

$M \rightarrow M_1 \cdots M_l$

$y_0 = IV$

for i = 1 to l do

$y_i = f(M_i, y_{i-1}, b_i, s)$

return y_l

3.2.1.3 Sponge construction

This construction is totally different in design than Merkle-Damgård, and it's about a new and promising hashing construction [32]. In sponge hashing we have two phases, the first one is the absorbing phase and the second one is the squeezing phase.

A sponge construction is any of a class of algorithms with finite internal state that take an input bit stream of any length and produce an output bit stream of any desired length. A sponge function is built from three components:

A state memory, S, containing b bits,

A function f: $\{0,1\}^b \rightarrow \{0,1\}b$ that transforms the state memory (often it is a pseudorandom permutation of the 2^b state values)

A padding function P

The state memory is divided into two sections: one of size r (the bitrate) and the remaining part of size c (the capacity). These sections are denoted R and C respectively.

The padding function appends enough bits to the input string so that the length of the padded input is a whole multiple of the bitrate, r. The padded input can thus be broken into r-bit blocks. The sponge construction can also be used to build practical cryptographic primitives. For example, Keccak cryptographic sponge with a 1600-bit state has been selected by NIST as the winner in the SHA-3 competition [32].

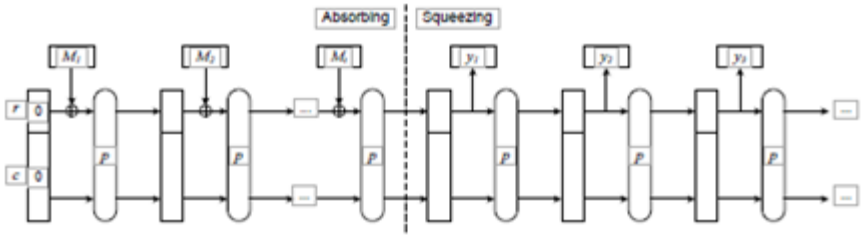


Figure 17: The sponge construction for hash functions [33]

Algorithm $\mathbf{Spng}_n^p$

$M \rightarrow M_1 \cdots M_l$

$r = 0, c = 0$

for i = 1 to l do

$p(r \overline{\oplus M_i}, c) = (r, c)$

for i = 1 to l do

$Y = Y \| r$

$p(r, c)$

return Y

As we notice before, the Sponge construction is totally different from the Merkle-Damgård, so the generic attacks that we have already discuss are not applicable in this construction. However, that does not mean that the sponge construction is not susceptible to other to other kinds of attacks like slide attack. An obvious disadvantage of sponge construction is that their relatively large states slows down the full diffusion of bits, hence the sponge construction may be more suitable for hashing large messages.

Secure Hash Function 3 (SHA-3)

After several successful collision attacks which were progressively reduced in complexity (such as MD5, SHA-1 and SHA-2), NIST, in the Federal Register, announced a public competition to develop SHA-3, a completely new hashing algorithm. In 2007, the announcement for the initiative was published. Then, four years later, on October 2nd, 2012, the winner of the competition Keccak, was announced. In 2014, NIST considered SHA-3 as a standard hash function. However, this algorithm is susceptible to first collision-finding attacks [34], [35]. On the other hand, the algorithm shows relatively low software performance compared to other hash functions [36].

3.2.1.4 Wide- Pipe Construction

One construction for hash function, which is called Wide- Pipe construction, was proposed as an improvement over Merkle- Damgård construction, by Stefan Lucks [33]. Its structure is quite similar to that of Merkle- Damgård design, but it has larger internal state size. Lucks [33] suggested that Joux [37]and length extension are mainly based on Internal collisions and internal collisions can be avoided if we widen the internal pipe from n bits to w >= n bits.

If a hash of n bits is desired, then two compression functions f_1, f_2 will be required:

- $f_1 : \{0,1\}^w X\{0,1\}^m \rightarrow \{0,1\}^w$
- $f_2 : \{0,1\}^w X\{0,1\}^n$

Then wide pipe iterated hash is constructed like this:

- For i = 1, ..., L : Computer $H_i = f_1 (H_{i-1} , M_i)$
- Finally Set $H(M) = f_2 (H_L)$

Compression function f_1 takes w bits (generally w = 2n) of chaining value and m bits of message (M) and compressed this to an output of w bits and in the last another compression function f_2, compresses the last internal hash value (w bits) to the final hash value (n bits). SHA- 224 and SHA-384 are based on the same design.

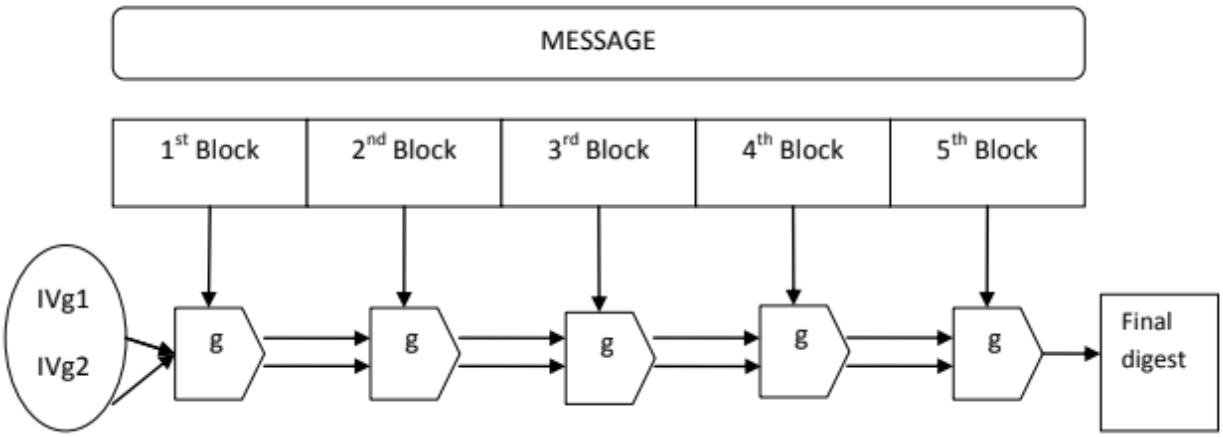


Figure 18: the Wide Pipe Hash Construction [33]

This structure is analogous to that of Merkle–Damgård construction with the only exception that its constitutional state size is large, due to which its internally used bit-length is also larger than Wide-pipe construction's output bit-length. The compression operation g takes 2n-bits of chaining variable and n bits of the message and compresses this to an output of 2n bit to produce a hash of n bits. Finally, one more compression function is used that compresses the final 2n bit long internal digest to the final hash value (n bits). Half of the last 2n-bit-output is simply abandoned. For example SHA- 224 derived from SHA- 256 and SHA-384 derived from SHA-512 and takes this form.

Mridul Nandi and Souradyuti Paul [38] established that the Wide pipe hash function can be made around two times faster. And for this, the wide pipe state should be split into two same parts as follows: first part should be given as input to the next compression

operation while the second part should be linked with the output of first part's compression operation. The important idea behind this design of hashing is to feed-forward half of the previous chaining value and then to XOR it to the result of the compression operation. While doing this in each of the iteration, the input message blocks to construction becomes longer than the original wide pipe construction method. Using the same function g as before, it takes c-bit chaining value and c + m bit of the message. However, this construction method demands extra memory to be used for feed-forward step. Lucks further suggested that to widen the internal state, the double pipe hash functions may also be used as an alternative approach. Two parallel iterations are processed in this approach. These two iterations can be initialized with different initialization vectors or they can use different compression functions or they can even iterate the message blocks in different permutations. Finally, the outputs of the two iterations are mixed to get the final digest value.

3.2.1.5 Tree- based Hash Functions

The Tree based hash functions are the most collocate category of hash constructions and they are firmly applicable for multi-core platforms in which various processors can independently but simultaneously perform on various parts of the message.

In this way, firstly the message is broken into blocks, and after individually randomization they are combined by an XOR type of operation. This structure can be used for building incremental functions at the same time its structure is similar to a two-level tree and it may be parallelized because of independence of the randomization process of the individual blocks. Different threads or processors are made responsible for this independence. Major limitation of Tree-based constructions is their non-suitability for low-end platforms like smart cards, because of this iterative functions are more popular and more usable. Skein [39] and MD6 [40] hash functions (SHA-3 candidates) provide a tree hashing mode.

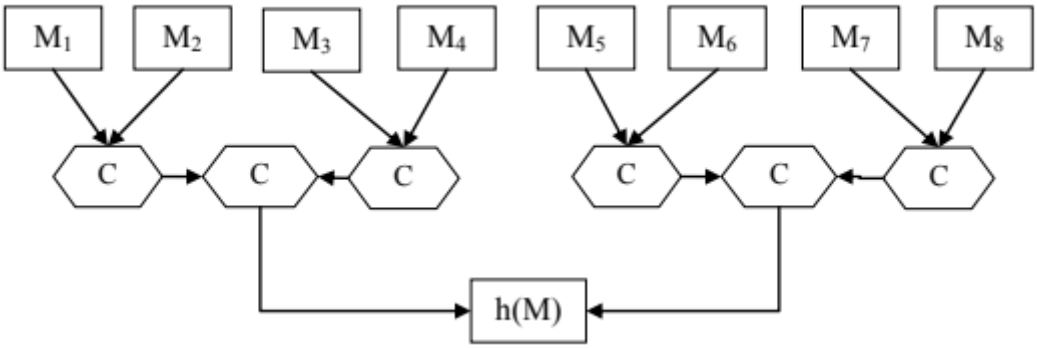


Figure 19: Tree- Based Hash Constructions [64]

3.3 Recent Hash Functions

Most of the hash functions are not enough safe from attacks. Nowadays, the aim of researchers is to find new hash functions probably with totally new designs. Some of recent hash functions designs such as Whirlpool, JH Hash, Blake-256, Blake2, SHA-3 (Keccak), Streebog and Kangaroo Twelve, that are using the new modified design architecture.

3.3.1 Whirlpool Secure Hash Function

Whirlpool is a cryptographic hash function that was designed by Vincent Rijmen and Paulo Barreto in 2000 and has been recommended by the NESSIE project. It is block cipher based secure hash algorithm, where block ciphers have disadvantages such as they are slow, but Whirlpool provides security and performance as good as hash functions based on non- block ciphers. Whirlpool takes a message of any length less than 2^{256} bits and returns a 512- bit message. This hash function is a Merkle–Damgård construction based on an AES like block cipher W [41] Because the output length is more than of SHA-1, we have a stronger result.

The Whirlpool hash function is given as:

H_0= initial value

$Hi = E(H_{i-1}, M_i) + H_{i-1} + M_i$

H_i = hash code value

The dedicated 512-bit block cipher W [K]: $M_{8\times8}[GF(2^8)] \rightarrow M_{8\times8}[GF(2^8)]$, parameterized by the 512-bit cipher key K, and it operates on a state of 4*16 bytes of Rijandeal. It becomes slow in speed, due to more numbers of rounds.

Whirlpool has good performance in terms of execution speed and can work with lesser memory requirements, because it does not require excessive storage space. It can be efficiently implemented in constrained environments like smart cards. Furthermore, it does not use expensive instructions for the building of the processor. The mathematical simplicity of the primitive resulting doing analysis easier. Finally, it has a very long hash length that provides protection against birthday attacks and offers a larger internal state of entropy containment, as needed for pseudo-random number generators [42].

3.3.2 JH Hash Function

JH Hash function was designed by Hongjun Wu to be submitted to NIST hash competition, in 2008. There are four JH hash algorithms, JH-224, JH-256, JH-384, JH-512, constructed from the same compression function. It processes message blocks of 512 bits and generates hash of 224,256,384,512 bits. This hash function is a Merkle–Damgård construction also, based on a generalized AES design methodology.

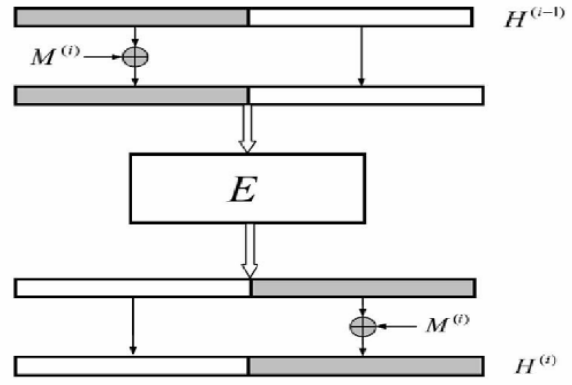


Figure 20: the JH compression function structure [43]

In each iteration the compression function f is used to update the chaining value of 1024 bits as follows: $H_i = f(H_{i-1}, M_i)$, where H_{i-1} is the previous value, M_i is the current message block. The compression function f is given as:

- $f(H_{i-1}, M_i) = E(H_{i-1} + M_i \text{ II } 0^{512}) + 0^{512} \text{ II } M_i$ where E is a permutation of 1024 bits, and 0^{512} means the string of 512 '0' bits.

For c≤ n/2, JH Hash Function using an ideal n bit permutation and producing c- bit outputs by truncation is collision resistant up to O ($2^{C/2}$). This bound implies that JH function provides the optimal collision resistance in the random permutation model. JH Hash functions are very efficient in S/W. With bit slice implementation using SSE2, the speed of JH is about 16.8 cycles/byte on Intel Core2 Duo microprocessor, running 64-bit Operating System, with Intel C++ compiler [44]

The simple JH compression function structure reduces the cost of security evaluation with respect to differential cryptanalysis. Because of enough confusion and diffusion after message modification, it is secure against differential attacks. We notice that it is resistant to second pre-image attack because we use 1024 bit hash value for JH- 512 while the reversible property of compression function is being taken into consideration.

3.3.3 BLAKE-256

Blake-256 hash function was developed by Jean- Philippe Aumasson, Luca Henzen, Willi Meier and Raphadel Phan in 2008 to be submitted as a competitor in NIST SHA-3 competition [45] The core BLAKE-256 compression function takes, as an input, 512 bits/16 words/64 bytes of message data, 256 bits/8 words/32 bytes of chaining value, 128 bits/4 words/ 16 bytes of salt, and additionally a counter that is 64 bits/2 words/8 bytes. A series of XORs, rotations and modular additions are used for generating new chaining values. Its compression function takes 512 bit input and 128 bit salt to produce 128 bit output by applying an invertible nonlinear transformation composed of 14 rounds, and each round uses a non-linear permutation G. It accepts four of 32 bit words, two message words and two constant words. It leads the simplicity of algorithm and performs fastly on software and hardware. As all of the blake-based algorithms, Blake 256 is based on and uses ChaCha stream cipher developed by Dan Bernstein. However, Blake 256 provides some additional features, like adding a rearranged copy of the input block, XORed with several round constants before each round of the ChaCha cipher. It can work for message less than 264 bit [45].

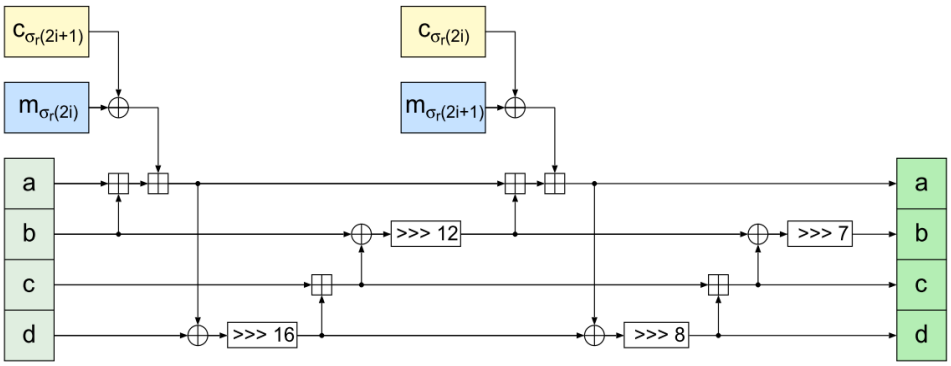


Figure 21: The Gi function of BLAKE-256 [46]

3.3.4 Keccak

Kavun and Yalcin reported the lightweight implementations of Keccak- f (200) and Keccak- f (400) permutations. They are variants of the SHA-3 hash function and the development of Keccak based on the sponge construction. Best known as a hash function, it nevertheless can also be used for authentication, (authenticated) encryption and pseudo-random number generation. Its structure is the extremely simple sponge construction and internally it uses the innovative Keccak-f cryptographic permutation [46].

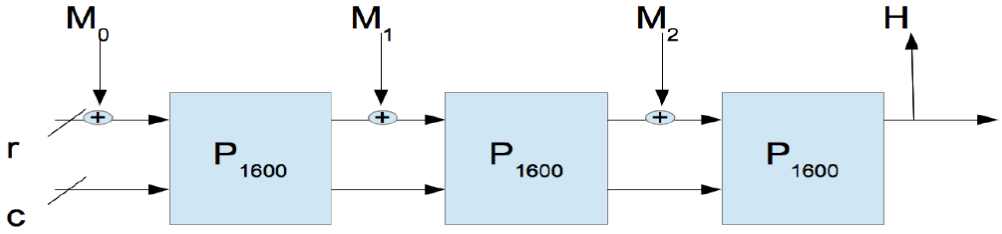


Figure 22: Keccak function [46]

Given an input bit string N, a padding function pad, a permutation function f that operates on bit blocks of width b, a rate r and an output length d, we have capacity c = b − r and the sponge construction Z = sponge[f,pad,r](N,d), yielding a bit string Z of length d, works as follows: [47]

- pad the input N using the pad function, yielding a padded bit string P with a length divisible by r (such that n = len(P)/r is integer)
- break P into n consecutive r-bit pieces P0, ..., Pn−1
- initialize the state S to a string of b zero bits
- absorb the input into the state: for each block Pi:
- extend Pi at the end by a string of c zero bits, yielding one of length b
- XOR that with S
- apply the block permutation f to the result, yielding a new state S
- initialize Z to be the empty string

while the length of Z is less than d:

- append the first r bits of S to Z
- if Z is still less than d bits long, apply f to S, yielding a new state S
- truncate Z to d bits

In SHA-3, the state S consists of a 5 × 5 array of w-bit words (with w=64), b = 5 × 5 × w = 5 × 5 × 64 = 1600 bits total. Keccak is also defined for smaller power-of-2 word sizes w down to 1 bit (total state of 25 bits). Small state sizes can be used to test cryptanalytic attacks, and intermediate state sizes (from w = 8, 200 bits, to w = 32, 800 bits) can be used in practical, lightweight applications [48],[32].

3.3.5 Streebog

Streebog is a family of two hash algorithms, Streebog-256 and Streebog-512, defined in the Russian national standard GOST R34.11-2012 Information Technology - Cryptographic Information Security - Hash Function. Streebog operates on 512-bit blocks of the input, using the Merkle–Damgård construction to handle inputs of arbitrary size. The high-level structure of the new hash function resembles the one from GOST R 34.11-94, however, the compression function was changed significantly.

The compression function operates in Miyaguchi–Preneel mode and employs a 12-round AES-like cipher with a 512-bit block and 512-bit key. (It uses an 8×8 matrix of bytes rather than AES's 4×4 matrix.)

Streebog-256 uses a different initial state than Streebog-512, and truncates the output hash, but is otherwise identical [49].

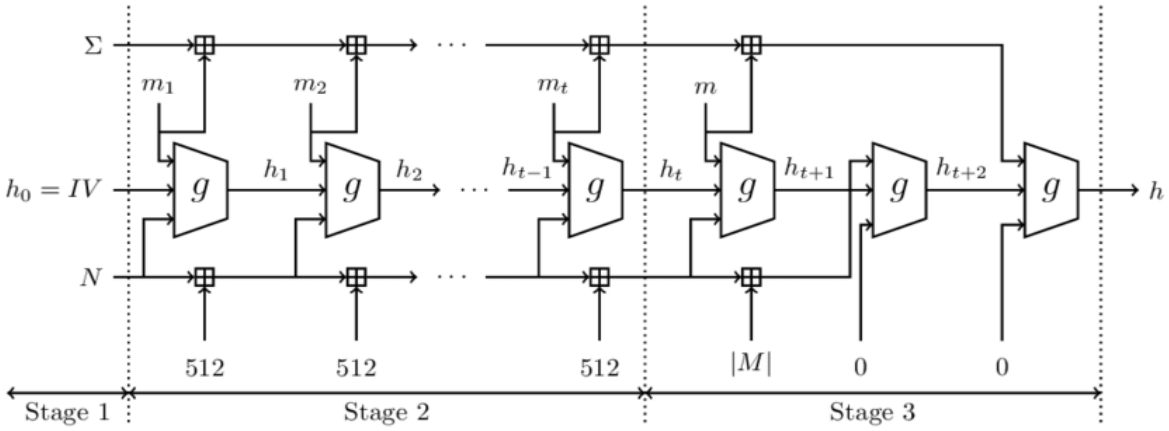


Figure 23: Streebog function [50]

Wang, et al, describe a collision attack on the compression function reduced to 9.5 rounds with 2176 time complexity and 2128 memory complexity [51].

Ma, et al, describe a preimage attack that takes 2496 time and 264 memory or 2504 time and 211 memory to find a single preimage of GOST-512 reduced to 6 rounds. They also describe a collision attack with 2181 time complexity and 264 memory requirement in the same paper [52].

Guo, et al, describe a second preimage attack on full Streebog-512 with total time complexity equivalent to 2266 compression function evaluations, if the message has more than 2259 blocks [50].

3.3.6 Kangaroo Twelve

Kangaroo Twelve is a fast and secure extendable-output function (XOF), the generalization of hash functions to arbitrary output lengths. Derived from Keccak, it aims at higher speeds than FIPS 202's SHA-3 and SHAKE functions, while retaining their flexibility and basis of security. Kangaroo Twelve is sharing many common features with SHAKE128, like the sponge construction, the extendable-output function (XOF), and the 128-bit security strength, but except from that it has major improvements.

On high-end platforms, it can exploit a high degree of parallelism, whether using multiple cores or the single-instruction multiple-data (SIMD) instruction set of modern processors. On Intel's® Haswell and Skylake architectures, Kangaroo Twelve tops at less than 1.5 cycles/byte for long messages on a single core, and at 0.55 cycles/byte on the SkylakeX architecture. On low-end platforms, as well as for short messages, it also benefits from about a factor two speed-up compared to the fastest FIPS 202.

Kangaroo Twelve is a higher-performance reduced-round (from 24 to 12 rounds) version of Keccak which claims to have 128 bits of security [53].

3.4 Message Authentication Codes (MAC)

A Message Authentication Code (MAC), is also known as a cryptographic checksum or a keyed hash function, and it is widely used in practice. MACs share some properties with digital signatures, and also provide message integrity and message authentication. In contrast with digital signatures, MACs are symmetric-key schemes and they do not provide non-repudiation. One benefit of MACs is that they are faster than digital signatures since they are based on either block ciphers or hash functions.

In cryptography, a message authentication code (MAC), is a short piece of information used to authenticate a message in different words, to confirm that the message came from the stated sender (its authenticity) and has not been changed [1],[2].

MAC defined over (K, M,T) is a pair of algorithms (S, V):

S(k, m): returns a message authentication code t which belongs to a set T

V(k, m, t): returns a value true or false depending on the correctness of the received authentication code where:

- M is a set of all possible messages m,
- K is a set of all possible keys k,
- T is a set of all possible authentication codes t

The simplest way to mark the authenticity of the message is to compute its checksum. One can attach the result to the transmitted message. The disadvantage of this method is the lack of protection against intentional modifications in the message content. The intruder can change the message, then calculate a new checksum, and eventually replace the original checksum by the new value. An ordinary CRC algorithm allows only to detect randomly damaged parts of messages (but not intentional changes made by the attacker).

We define the message integrity problem using, the A sender and B receiver. Suppose B receives a message (which may be encrypted or may be in plaintext) and it is believed that this message was sent by A. To authenticate this message, B needs to verify:

a. The message indeed originated from A.

b. The message was not tampered with on its way to B. [5]

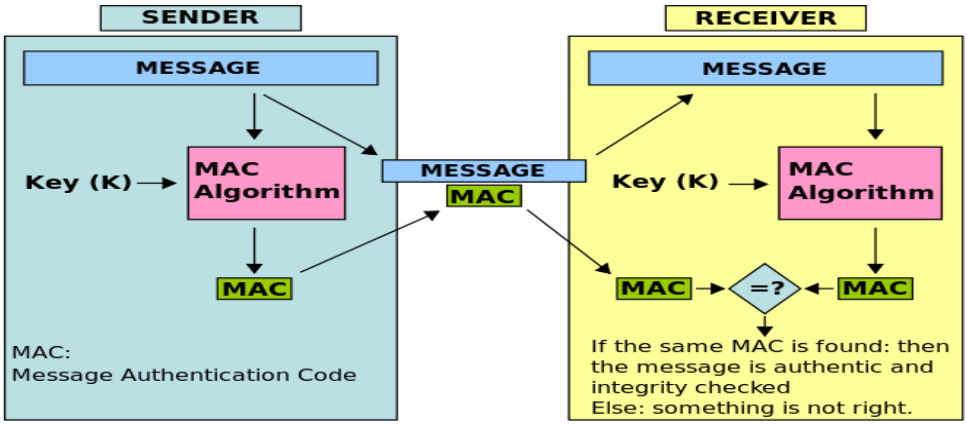


Figure 24: MAC Algorithm [5]

Properties of Message Authentication Codes

1. Cryptographic checksum A MAC generates a cryptographically secure authentication tag for a given message.
2. Symmetric MACs are based on secret symmetric keys. The signing and verifying parties must share a secret key.
3. Arbitrary message size MACs accept messages of arbitrary length.
4. Fixed output length MACs generate fixed-size authentication tags.
5. Message integrity MACs provide message integrity: Any manipulations of a message during transit will be detected by the receiver.
6. Message authentication The receiving party is assured of the origin of the message.
7. No nonrepudiation Since MACs are based on symmetric principles, they do not provide nonrepudiation [1].

3.4.1 HMAC

HMAC is a popular system of checking message integrity. It uses one-way hash functions to produce unique mac values.

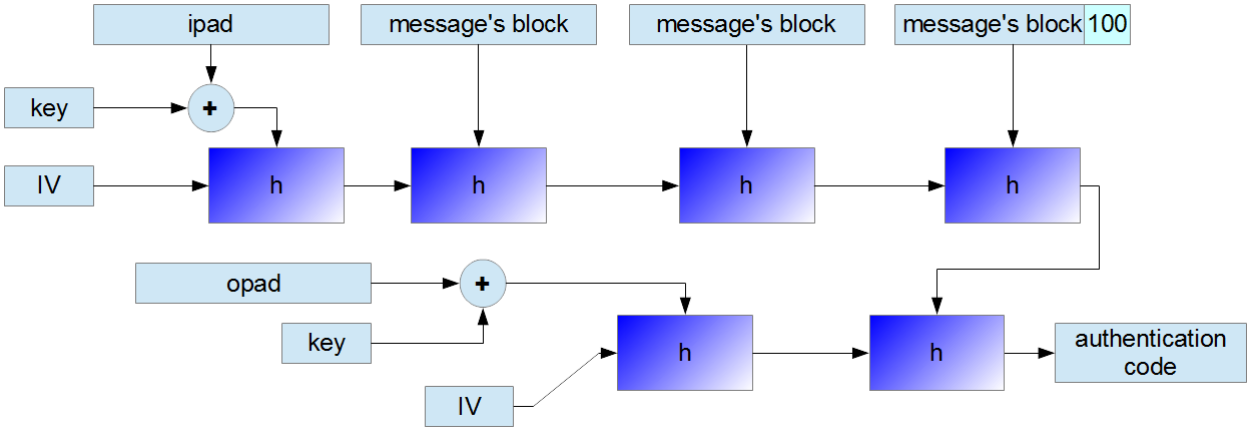


Figure 25: HMAC [5]

The input parameters ipad and opad are used to modify the secret key. They may have various values assigned. It is recommended to choose the values that would make both inputs to the hash functions look as different as possible. Using a secure hash function guarantees the security of the HMAC algorithm. Nowadays, the HMAC algorithm is used in many systems, including some popular Internet protocols (SSL, IPsec, SSH) [1].

3.5 Digital Signatures

Digital signatures are one of the most important cryptographic tools and they are widely used nowadays. A digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents. There must be a number of prerequisites satisfied in order a digital signature become valid. The digital signature gives to the recipient very strong reason to believe that the message was created by a known sender (authentication), and that the message was not altered in transit (integrity).

Applications for digital signatures range from digital certificates for secure e-commerce to legal signing of contracts to secure software updates. Together with key establishment over insecure channels, they form the most important instance for public-key cryptography.

As with conventional hand-written signatures, only the person who creates a digital message must be capable of generating a valid signature. In order to achieve this with cryptographic primitives, we have to apply public-key cryptography. The basic idea is that the person who signs the message uses a private key, and the receiving party uses the matching public key.

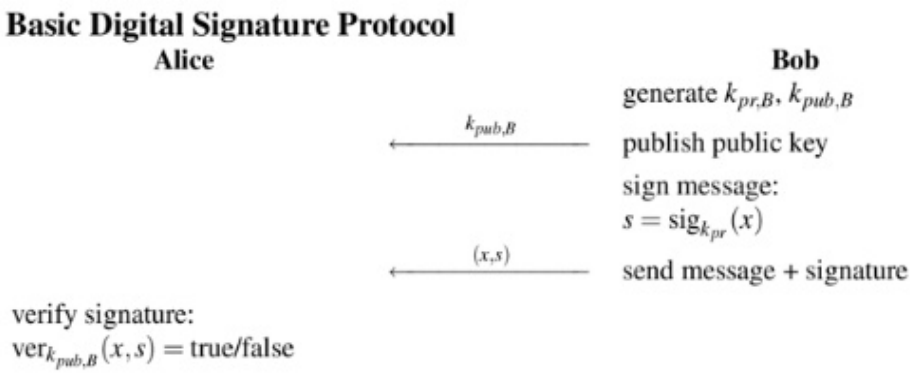


Figure 26: Basic Digital Signature Protocol

A signed message can unambiguously be traced back to its originator since a valid signature can only be computed with the unique signer's private key. Only the signer has the ability to generate a signature on his behalf. Hence, we can prove that the signing party has actually generated the message [1], [2],[7].

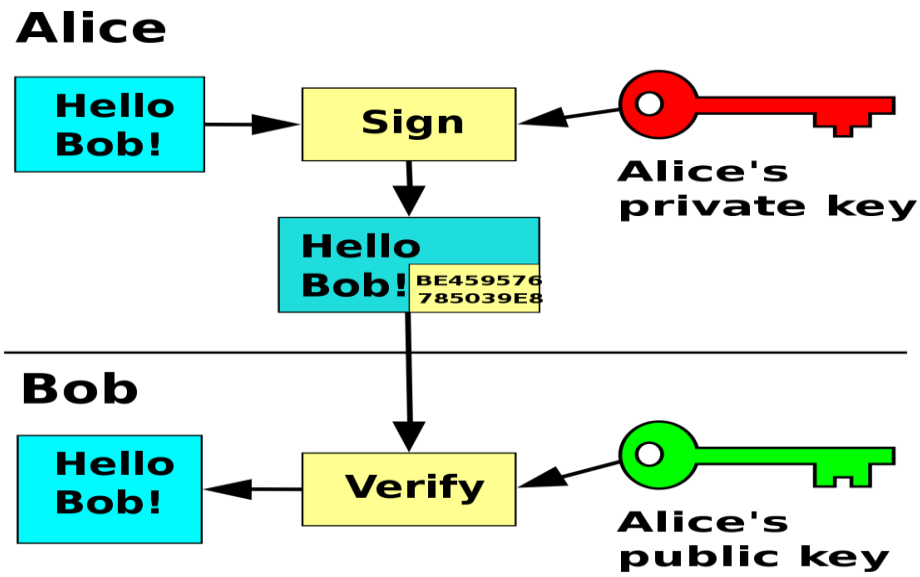


Figure 27: Digital Signature

3.5.1 El- Gamal Digital Signature Scheme

El- Gamal digital signature is the asymmetric approach of authentication mechanism. It is based on discrete logarithm and uses β as the universally known random number that serves as the generator, u as the universally known prime number that serves as the modulus, H() as the universally hash function [2], [54], [55], [56] .

As with Diffie–Hellman, the global elements of Elgamal are a prime number q and

a, which is a primitive root of q. User A generates a private/public key pair as follows:

1. Generate a random integer X_A, such that $1 < X_A 6 < q - 1$.
2. Compute $Y_A = a^{X_A} mod\ q$
3. A's private key is X_A and A's public key is {q, a, Y_A}.

Any user B that has access to A's public key can encrypt a message as follows:

1. Represent the message as an integer M in the range $0 \leq M \leq q - 1$.

Longer messages are sent as a sequence of blocks, with each block being an integer less than q.

2. Choose a random integer k such that $1 \leq k \leq q - 1$.
3. Compute a one-time key $K = {Y_A}^k mod\ q$
4. Encrypt M as the pair of integers (C1, C2) where

$$C_1 = a^k mod q \quad C_{2=} KM\ mod q$$

User A recovers the plaintext as follows:

1. Recover the key by computing ${C_1}^{X_A} mod\ q$
2. Compute $M = (C_2 K^{-1})\ mod\ q$

These steps are summarized in Figure 28. Alice generates a public/private key pair; Bob encrypts using Alice's public key; and Alice decrypts using her private key.

Let us demonstrate why the Elgamal scheme works. First, we show how K is recovered by the decryption process:

$$K = {Y_A}^k mod\ q \qquad K\ is\ defined\ during\ the\ encryption\ process$$
$$K = (a^{X_A} mod\ q)\ mod\ q \qquad substitude\ using\ Y_A = a^{X_A} mod\ q$$
$$K = a^{kX_A} mod\ q \qquad by\ the\ rules\ of\ modular\ arithmetic$$
$$K = {C_1}^{X_A} mod\ q \qquad substitude\ using\ C_1 = a^k mod q$$

Next, using K, we recover the plaintext as

$$C_2 = KM\ mod\ q$$
$$(C_2 K^{-1}) mod\ q = KMK^{-1} mod\ q = M\ mod\ q = M$$

We can restate the Elgamal process as follows, using Figure 28.

1. Bob generates a random integer k.

2. Bob generates a one-time key K using Alice's public-key components YA, q and k.

3. Bob encrypts k using the public-key component a, yielding C_1. C_1 provides sufficient information for Alice to recover K.

4. Bob encrypts the plaintext message M using K.

5. Alice recovers K from C_1 using her private key.

6. Alice uses K^{-1} to recover the plaintext message from C_2.

Global Public Elements	
q	prime number
α	$\alpha < q$ and α a primitive root of q

Key Generation by Alice	
Select private X_A	$X_A < q - 1$
Calculate Y_A	$Y_A = \alpha^{X_A} \bmod q$
Public key	$\{q, \alpha, Y_A\}$
Private key	X_A

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < q$
Select random integer k	$k < q$
Calculate K	$K = (Y_A)^k \bmod q$
Calculate C_1	$C_1 = \alpha^k \bmod q$
Calculate C_2	$C_2 = KM \bmod q$
Ciphertext:	(C_1, C_2)

Decryption by Alice with Alice's Private Key	
Ciphertext:	(C_1, C_2)
Calculate K	$K = (C_1)^{X_A} \bmod q$
Plaintext:	$M = (C_2 K^{-1}) \bmod q$

Figure 28: El- Gamal Cryptosystem [2]

Thus, K functions as a one-time key, used to encrypt and decrypt the message.

For example, let us start with the prime field GF (19); that is, q = 19. It has

Primitive roots {2, 3, 10, 13, 14, 15} We choose a = 10.

Alice generates a key pair as follows:

1. Alice chooses X_A = 5.
2. Then $Y_A = a^{X_A} mod\ q = a^5 mod\ 19 = 3$
3. Alice's private key is 5 and Alice's public key is {q, a, Y_A} = {19, 10, 3}.

Suppose Bob wants to send the message with the value M = 17. Then:

1. Bob chooses k = 6.
2. Then K = $(Y_A)^k$ mod q = 36 mod 19 = 729 mod 19 = 7.
3. So $C_1 = a^k$ mod q = a^6 mod 19 = 11

C_2 = KM mod q = 7 * 17 mod 19 = 119 mod 19 = 5

4. Bob sends the ciphertext (11, 5).

For decryption:

1. Alice calculates K = $(C_1)^{XA}$ mod q = 11^5 mod 19 = 161051 mod 19 = 7.

2. Then K^{-1} in GF (19) is 7^{-1} mod 19 = 11.

3. Finally, M = (C_2^{K-1}) mod q = 5 * 11 mod 19 = 55 mod 19 = 17.

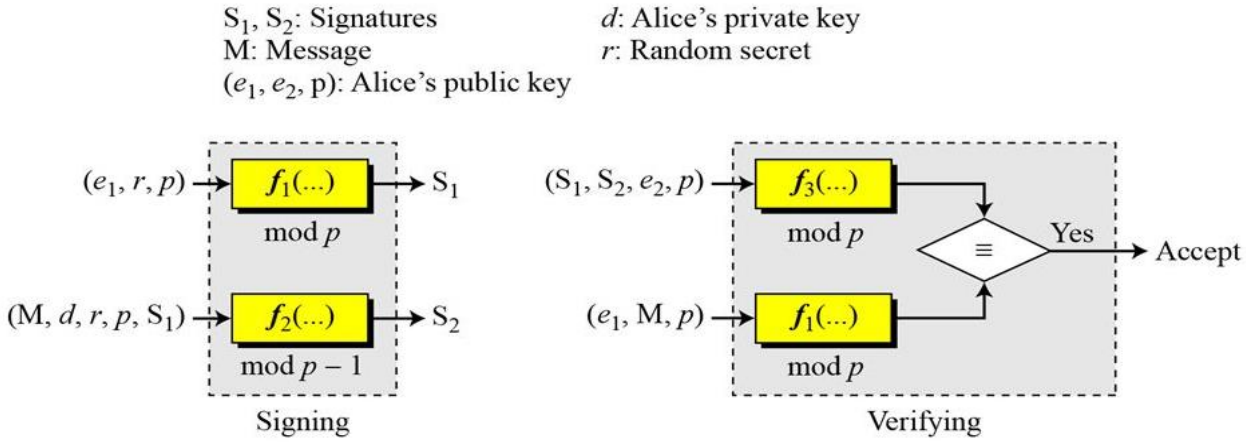


Figure 29: El- Gamal Digital Signature Scheme [2]

3.5.2 RSA Digital Signature Algorithm

This technique uses modulo arithmetic to sign a message digitally. Let B (sender) sends the message to A (receiver). This technique considers the public key of B and hash function H() is universally known [55].

Firstly, B performs the following:

i. Selects two prime numbers, U and V

ii. Computes N_B = U. V

iii. Selects P_B such that P_B has no division (factors) in common with [(U-1) (V-1)]

iv. Calculates the secret key S_B such that S_B P_B = 1 mod [(U-1) (V-1)]

The public key set of B contains N and P_B, using which B creates the signature of the message.

v. B hashes the msg [h= H(msg) h is the hash of the message msg]

vi. B creates the digital signature [sign = h ^{SB} mod N_B where sign is the signature]

Once the signature is created, B sends (msg, sign) to A.

vii. A uses the H() to obtain the h' (hash') [h' = H(msg')]

viii. A decrypts the signature to retrieve its hash (h) [$h = sign^{PB} \bmod N_b$]

ix. Alice finally checks if : h = h'

x. If the match is found in the hash value retrived and the hash value calculated, then A confirms the authenticity and integrity of the message along with the signature, else it is rejected.

3.5.3 Digital Signature Algorithm (DSA)

Digital signature algorithm is generated using various parameters like the private key x, per message secret key number k, data to be signed, and the hash function. Similarly it is verified using various parameters like the public key y which is mathematically calculated from x, the data to be verified and the same hash function used during signature generation [57].

The parameters used are as follows:

- p – a prime modulus
- q – a prime divisor of (p-1)
- g – a generator of the sub group of order q mod p.
- x - the private key is an randomly selected integer within the range [1, q-1].
- y – the public-key obtained through $y = g^x \bmod p$.
- k – the per message secret key (unique to each message) obtained randomly within the range [1,q-1].

Let N be the bit length of q. Let min (N, outlen) denote the minimum of the positive integers N and outlen, where outlen is the bit length of the hash function output block. The signature of message M contains pair of numbers r and s obtained using:

- $r = (g^k \bmod p) \bmod q$.
- z = the leftmost min(N, outlen) bits of Hash(M).
- $s = (k^{-1} (z + xr)) \bmod q$.

Once the signature (r,s) is generated, A may transmit message M, and (r,s) to B. Let M', r' and s' be the transmitted version of M, r and s.

To verify the signature B will perform the following steps:

i. B shall check that $0 < r' < q$ and $0 < s' < q$; if any one of the condition is violated, the signature is rejected.

ii. If both the conditions in step-i are satisfied, B computes- $w = (s')^{-1} \bmod q$, where $(s')^{-1}$ is the multiplicative inverse of s' mod q

z = the leftmost min(N, outlen) bits of Hash(M').

u1 = (zw) mod q.

u2 = ((r')w) mod q.

v = (((g)^{u1} (y)^{u2}) mod p) mod q.

iii. If v = r', then the signature is accepted else rejected.

3.5.4 Elliptic Curve Digital Signature Algorithm

This is the elliptic curve cryptographic version of Digital Signature Algorithm (ECDSA). This algorithm operates based on combination of three algorithms, key generation, signature generation and signature verification.

The key pair of an user (say A) is associated with a specific set of EC domain parameters D= (q, FR, a, b, G, n, h), where: E is an elliptic curve defined over Fq; P is a point of prime order n in E(Fq); q is a prime; FR is the Field Representation which is an indication for representation used for the elements of Fq; a and b are the two field elements in Fq which define the equation of the elliptic curve E over Fq'.

two field elements x_G and y_G in Fq which define a finite point G=(x_G, y_G) of prime order in E(Fq); the cofactor h= #E(Fq)/n [58].

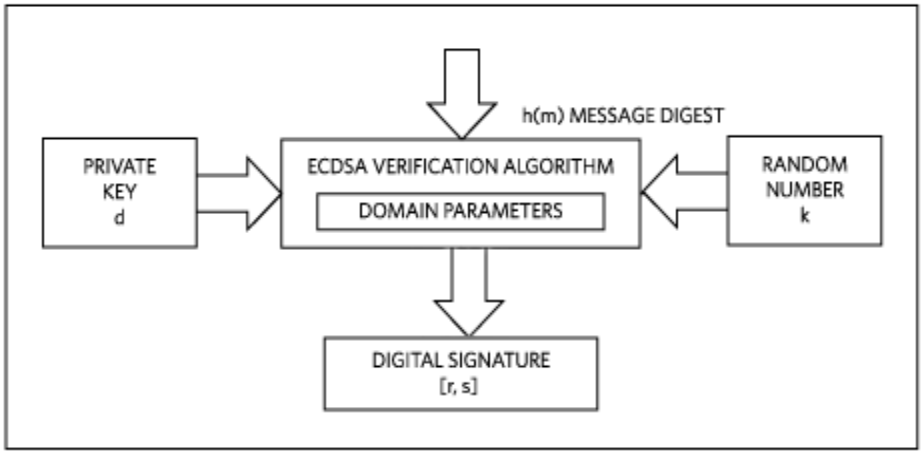


Figure 30: Elliptic Curve Digital Signature Algorithm [58]

3.5.5 Elliptic Curve ElGamal Digital Signature Scheme

Elliptic Curve Cryptography can be combined with ElGamal Digital signature algorithm to generate EC ElGamal Digital Signature Scheme. Entity A selects a random integer k_A from the interval (1, n-1) as the private key and computes the public key, A = k_A G [59].

i. Select random interger k from the interval (1, n-1).

ii. Compute R= kG = (x_R, y_R) where r = x_R mod n; if r = 0 go to step i.

iii. Compute e = h(M), where h is the hash function {0,1}* → F_n

iv. Compute s = k^{-1} (e + rkA) mod n; if then go to step i. (R,s) is the signature of message M. A sends the signature and the message to B for verification.

B performs the following to verify the signature: Verify that s is an integer in (1, n-1) and R = (x_R, y_R) ε E(F_q)

i. Compute V1 = sR

ii. Compute V2 = h (M)G + rA, where r = x_R

iii. If V1 = V2, then the signature is accepted by B, else declared as invalid.

4 HASH FUNCTIONS SECURITY ANALYSIS

4.1 Security properties

There are three properties a hash function is expected to preserve. These three properties are collision resistance, pre-image resistance and 2nd pre- image resistance.

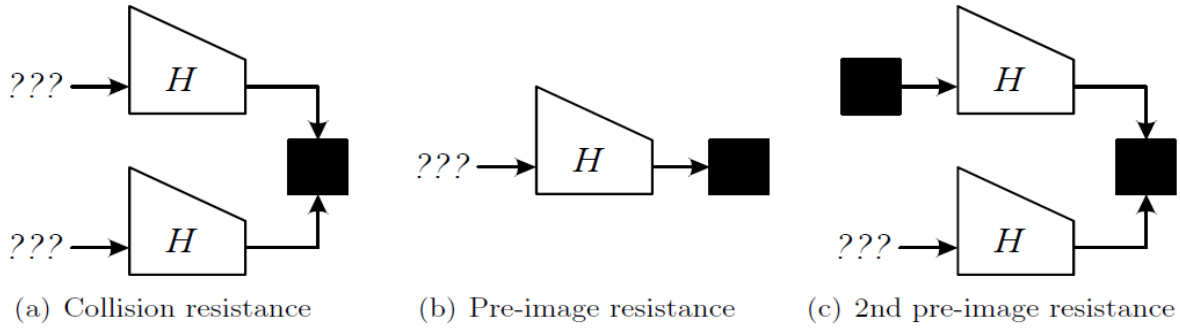


Figure 31: Security properties collision resistance, pre-image resistance, 2nd pre-image resistance [25], [60], [61]

The input to a secure hash function is called the pre-image and the output is called the image. A hash function collision is two different inputs (pre-images) which result in the same output. A hash function is collision-resistant if an adversary can't find any collision. A hash function is pre-image resistant if, given an output (image), an adversary can't find any input (pre-image) which results in that output. A hash function is second-pre-image resistant if, given one pre-image, an adversary can't find any other pre-image which results in the same image.

4.1.1 Collision – Resistance (CR)

Collision resistance is a property of cryptographic hash functions. A hash function H is collision resistant if it is difficult to find two inputs that hash to the same output. For example, any two inputs a and b such that H(a) = H(b), while a ≠ b. Collision resistance does not mean that no collisions exist but simply is hard to find [25].

A family of functions $\{h_k : \{0, 1\}^{m(k)} \rightarrow \{0, 1\}^{l(k)}\}$ generated by some algorithm G is a family of collision resistant hash functions, if $|m(k)| > |l(k)|$ for any k, i.e., h_k compresses the input string, and every h_k can be computed within polynomial time given k, but for any probabilistic polynomial algorithm A, we have

$\Pr [k \leftarrow G(1n), (x1, x2) \leftarrow A(k, 1n) \text{ s.t. } x1 \neq x2 \text{ but } h_k(x1) = h_k(x2)] < \text{negl}(n)$,

Where negl (·) denotes some negligible function, and n is the security parameter [60].

4.1.2 Pre- image Resistance (Pre)

Given a hash h it should be hard to find any message m such that h = hash (m). This concept is related to that of the one-way function. Functions that lack this property are vulnerable to pre-image attacks. So hash functions should be computationally non-

invertible, that means that when a message is hashed, it should be infeasible to retrieve the original message from which the hash value was obtained.

4.1.3 2nd Pre- image Resistance (Sec)

Given an input m1, it should be hard to find another input, m 2 (not equal to m1) such that hash(m1)=hash(m2). This property is sometimes referred to as weak collision resistance. Functions that lack this property are vulnerable to second pre-image attacks. The best attack against hash should be the brute force attack [61].

A graphical representation of the above attacks is shown in Figure 29 to help understand the concepts better.

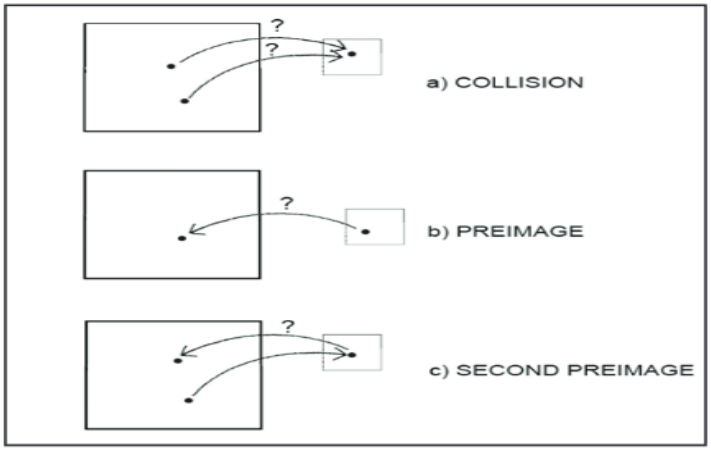


Figure 32: Different Types of Attacks on Hashing Algorithms

4.2 Attacks on Hash Functions

As per definition attacking a hash function means breaking one of the security properties of the hash functions. Attacks may focus on structure of hash function or on algorithm of compression function. As we can see in the figure below, from the classification of the attacks on Hash functions, hash functions can be classified on classifications based on properties and classifications based on attacking methodology.

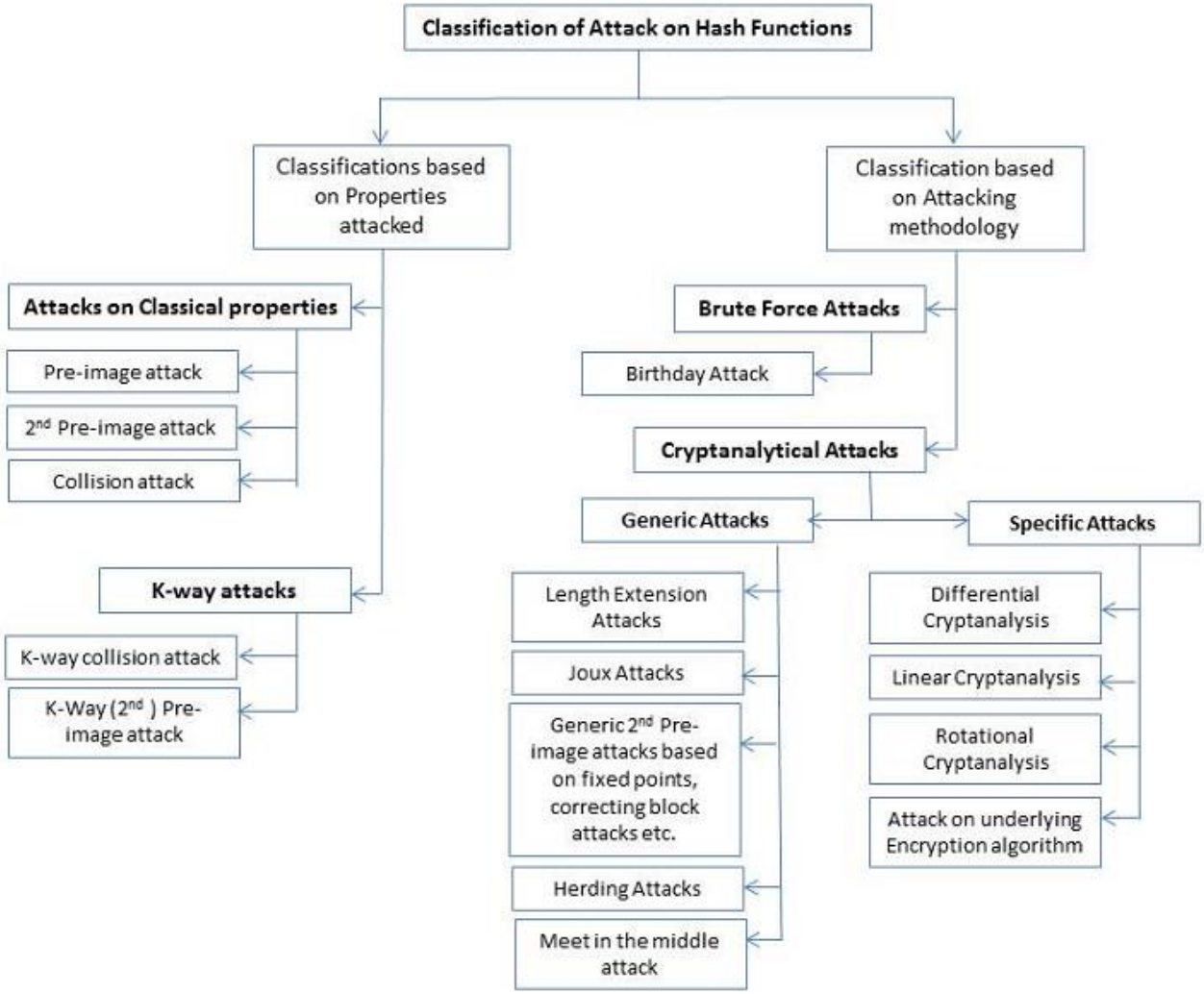


Figure 33: Classification of attacks on Hash Functions [62]

4.2.1 Tree Based Attack

Tree based hash functions are able to be made parallel for hash constructions and these are pertinent for multi-core platforms in which various processors can independently but simultaneously perform on various parts of the message. The first who suggested an early tree-based mode of operation, was Damgård [63], and Pal and Sarkar [64] advanced it. Equivalently, this way was also used by Rogaway and Bellare [61] who they designed non-keyed one-way hash functions along with Naor and Yung [65] but it could not be proved stronger than collision-resistant hash functions. Micciancio and Bellare [66] in they designed the way randomize- and- combine in which first the message is broken into blocks, and after separately randomization they are combined by an XOR type of operation.

This structure can be used for constructing accumulative functions, simultaneous its structure is similar to a two-level tree and it may be parallelized because of independence of the randomization process of the individual blocks. Different threads or processors are in charge for this independence. Extensive limitation of Tree-based constructions is their inappropriateness for low-end platforms such as RFID and smart cards, because of these iterative functions are more well-known and more working. Along with repetitive structure, Skein [39] and MD6 [40]hash functions also follow a tree hashing mode.

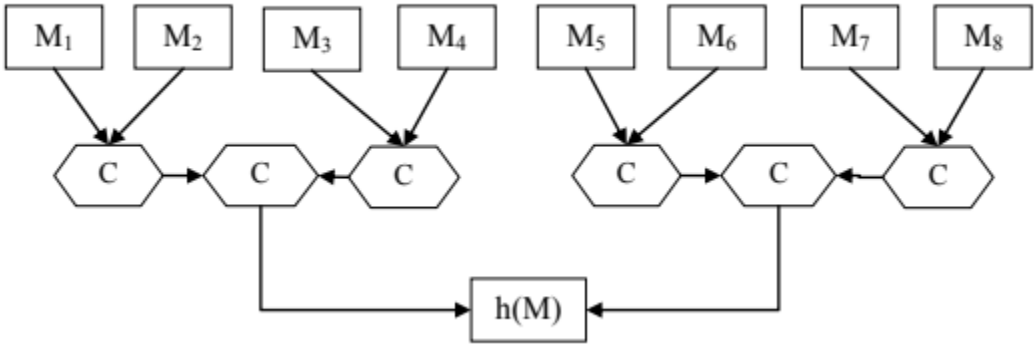


Figure 34: Tree- Based Hash Construction [64]

4.2.2 The most common attack - Brute Force Attack

Brute force attacks work on all hash functions independent of their structure and any other working details. They are similar to exhaustive search or brute- force key recovery attacks on the encryption schemes to extract the secret key of the encryption scheme. The security of any hash function lies in its output bit size. For a hash code of length n, the level of effort required to resist different brute force classical attacks on hash functions is as follow:

Pre-image attack: Effort required for brute force attack = 2n. In this attack, for a given n-bit digest h of the hash function H( ), the attacker evaluates H( ) with every possible input message M until the attacker obtains the value h.

2nd Pre-image attack: Effort required for brute force attack = 2n. In this attack, for a given message M and the hash function H( ), the attacker tries H( ) with every possible input message M' ≠ M until the attacker obtains the value H(M).

Collision attack: Effort required for brute force attack= 2n/2. In this attack, for a given hash function H, the attacker tries to find two messages M and M' such that M ≠ M' and H(M) = H(M'). On average the opponent would have to try 2n/ 2 (= 2n-1) messages to find one that matches the hash code of the intercepted message. However a chosen plain text attack (based on Birthday Paradox) is possible and in that case the effort required for collision in a Hash function is 2n/2 in place of 2n-1. It is also referred as Birthday Attack [22].

4.2.2.1 *Multi-* Preimage *Attack*

We briefly study about the multi- Preimage attack on Tree- based Hash functions. We can define the following attack for a Hash Function H : {0,1} * → $\{0,1\}^n$.

Given a Random y ϵ $\{0,1\}^n$, find a subset C = { $X_{1,\ldots,}X_r$} of size r (≥1) such that

$H(X_1)= \ldots =H(X_r)= y$.

The complexity for multi-preimage attack for a random function is $\Omega(r2^n)$ where for a Tree based hash function there is a r- way preimage attack which complexity is O ($2^{n/2}$). We have to mention that is very similar with the Multicollision attack and for what we are looking for is output value as given image y, and not finding the last collision. The last step's complexity is $O(2^n)$ which transcend to the complexity $r^2n2^{n/2}$ of Multicollision attack.

4.2.3 Merkle Damgård Construction

4.2.3.1 Joux's Multicollision Attack

A. Joux [71] found an algorithm to construct a 2^r-multicollision set on a classical iterated hash function, having time complexity $O(r\ 2^{n/2})$, which is a considerable improvement over the birthday attack. There is a 2^r-way collision attack for the classical iterated hash function based on a compression function, $f : \{0, 1\}^{n+n'} \rightarrow \{0, 1\}^n$, where the attack has complexity $O(r\ 2^{n/2})$. This complexity is much less than the complexity for the generalized birthday attack.

This is the basic idea of Joux's attack. Consider the set of n-tuples $\{0, 1\}^n$. We use the notation ***h → h' (***a labeled arc) to mean ***f(h,m***) = ***h',*** where |h| **=** |h'| **= *n*** and |m| = ***n'.*** The strategy of Joux's attack is to first find ***r*** successive collisions by performing ***r*** successive birthday attacks, as follows:

$$z_0 \overset{y_1^2}{\rightarrow} z_1\ and\ z_0 \overset{y_1^2}{\rightarrow} z_1\ for\ some\ z_1\ where\ y_1^1 \neq y_1^2$$

$$z_0 \overset{y_2^2}{\rightarrow} z_1\ and\ z_1 \overset{y_2^2}{\rightarrow} z_2\ for\ some\ z_2\ where\ y_2^1 \neq y_2^2$$

$$\vdots$$

$$z_{r-1} \overset{r}{\rightarrow} z_r\ and\ z_{r-1} \overset{y_r^2}{\rightarrow} z_r\ for\ some\ z_r\ where\ y_r^1 \neq y_r^2$$

$$then\ the\ set$$

$$\{y_1^1, y_1^2\}\ x\ \{y_2^1, y_2^2\}\ x\ \cdots x\{y_r^1, y_r^2\}$$

$$is\ a\ 2^r - multicollision$$

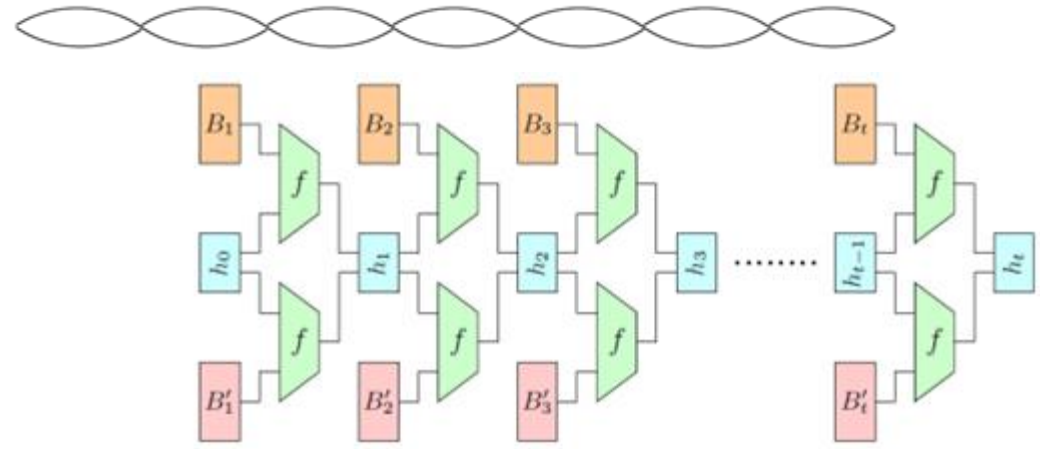


Figure 35: Joux's Multicollision Attack [67]

4.2.3.2 Second Preimage Attack on Merkle Damgård Construction

Kelsey and Kohno in 2006, published a generic second preimage attack for long messages against the Merkle Damgård Scheme. The attack complexity is 2^{n-k} compression function calls if the original given message is 2^k – block long.

We will describe the diamond structure. A diamond structure of size l is a multicollision with the shape of a tree of depth l with 2^l leaves. The tree nodes are labeled by the n-bit chaining values, and the edges are labeled by the m-bit message blocks. A message

block is mapped between two evolving states of the chaining value by the compression function f. Thus, there is a path labeled by the l message blocks from any one of the 2^l starting leaf nodes that leads to the same final chaining value h at the root of the tree [68].

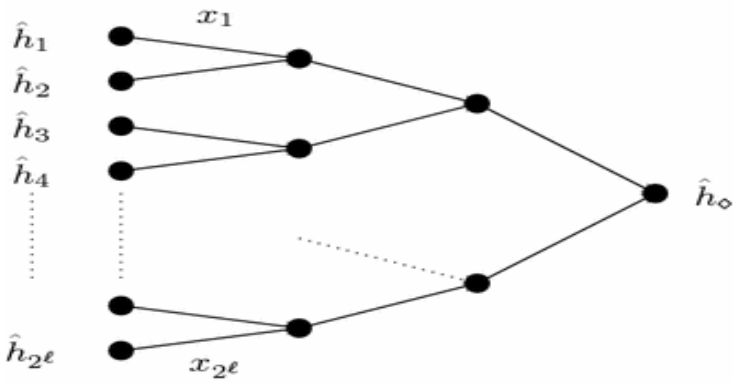


Figure 36: 2nd pre-image attack on Merkle –Damgård [68]

For the new second preimage attack, let M be a target message of length 2k blocks. The main idea of the attack is that connecting the target message to a precomputed collision tree of size l can be done with 2 n-l computations. In addition, connecting the root of the tree to one of the 2k chaining values encountered during the computation of H f (M) takes only 2 n-k compression function calls. Since a diamond structure can be computed in time much less than 2 n, we successfully launch a second preimage attack. The attack works in four steps. The messages M' and M are of equal length and hash to the same value before strengthening, so they produce the same hash value with the added Merkle-Damgård strengthening. The first step allows for precomputation and its time and space complexity is about 2(n+l)/2+2. The second step of the attack is carried out online with 2 n-k work, and the third step takes 2 n-l works. The total time complexity of the attack is then 2(n+l)/2+2 precomputation and 2 n-k + 2 n-l online computations and their sum is minimal when l= (n-4)/3 for a total of about 5* 2 2n/3 +2 n-k computations [31].

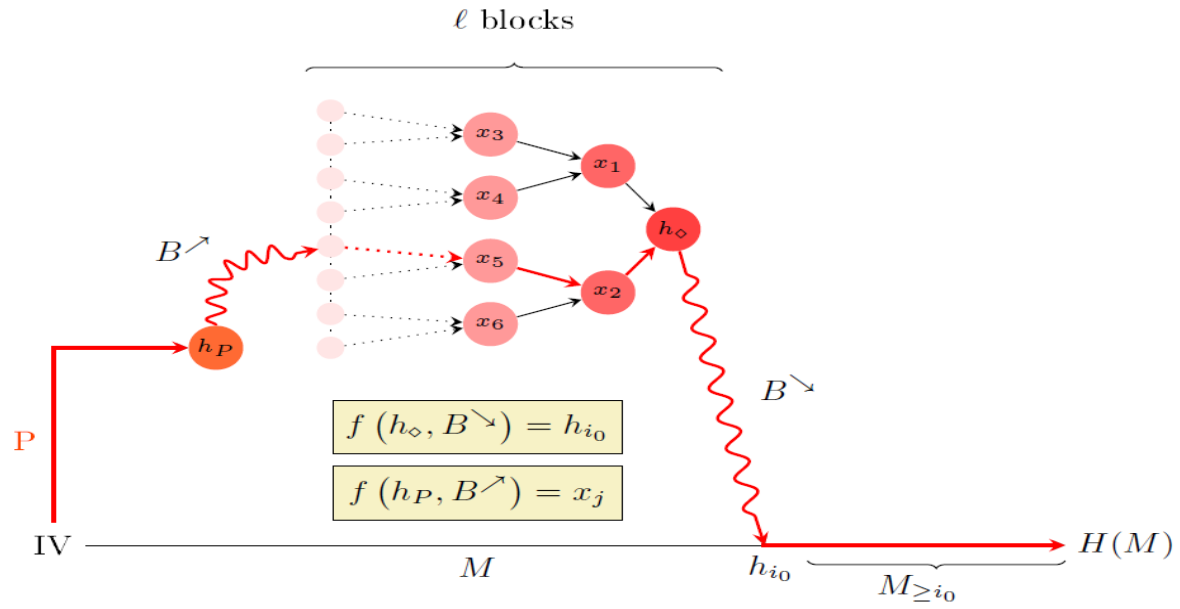


Figure 37: Representation of New Attack on Standard Merkle- Damgård [31]

4.2.4 HAIFA Construction

4.2.4.1 State- recovery attack HMAC with HAIFA

Firstly, we will describe the first internal state- recovery attack HMAC with HAIFA construction. The attack has a complexity of $\tilde{O}(2^{l-s})$ using messages of length 2^s, but this only applies with $s \leq l/5$, and the lowest complexity we can have is $2^{4l/5}$ [69].

The detailed attack is as follows:

Fix a message C of length 2s. Query the oracle with 2u messages Mi = [i] ‖ C.

Build an online diamond filter for the set of unknown states X, obtained after M_i.

Starting from 2^t arbitrary starting points, iterate the compression function with the fixed message C.

Test each image point x', against each of the unknown states of X. If a match is found, then with high probability the state reached after the corresponding M_i is x'.

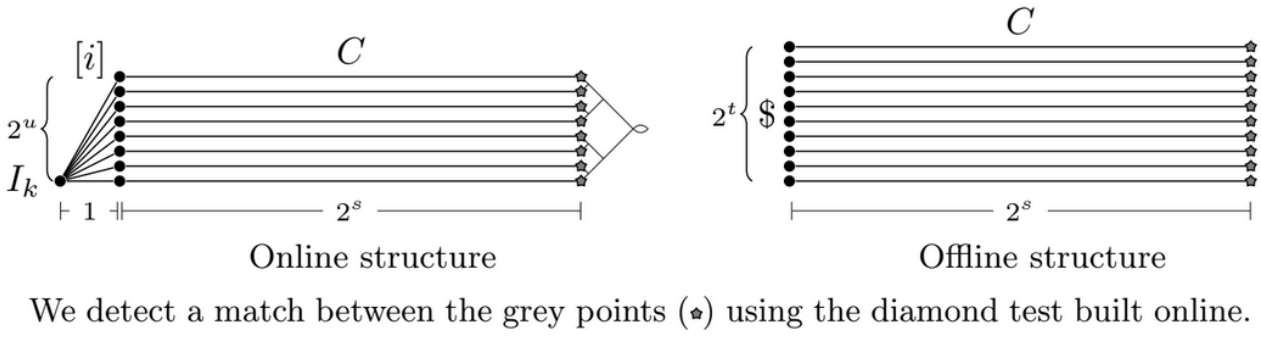


Figure 38: State- recovery attack HMAC with HAIFA [70]

Complexity analysis

In Step 3, we match the set X of size 2^u and a set of size 2^t. We compare 2^{t+u} pairs of points, and each pair collides with probability 2^{s-l}. The attack is successful with high

probability if t + u ≥ l− s. We now assume that t = l − s − u, and evaluate the complexity of each step of the attack:

Step 1: $2^{s+u/2+l/2}$

Step 2: $2^{s+t} = 2^{l-u}$

Step 3: $2^{t+u} \cdot u = 2^{l-s} \cdot u$

The lowest complexity is reached when all the steps of the attack have the same complexity, with s = l/5. Generally, we assume that s ≤ l/5 and we set u = s.

This give an attack with complexity O (2^{l-s}) since s + u/2 + l/2 = 3s/2 + l/2 ≤ 4l/5 ≤ l − s.

The Second attack that we will describe is short message attack for HMAC with HAIFA. The attack has a complexity of $\tilde{O}(2^{l-2s})$ using messages of length 2s, but this only applies with s ≤ l/10, and the lowest complexity we can have is $2^{4l/5}$.

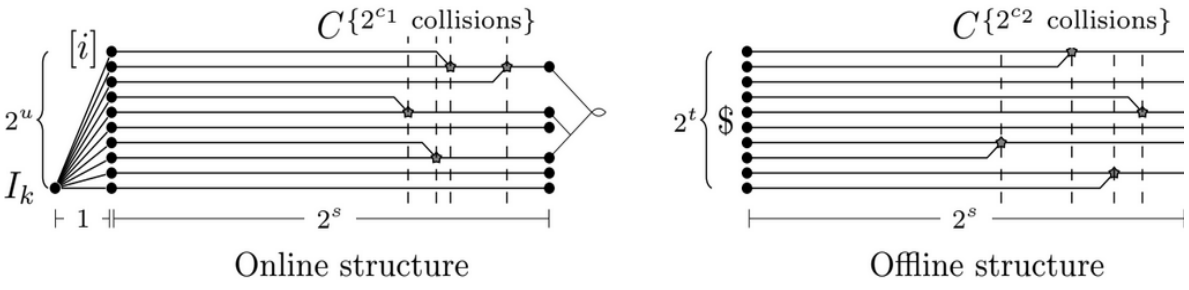
4.2.4.2 Short message attack for HMAC with HAIFA

1. Query the oracle with 2^u messages $M_i = [i] \| [0]^{2s}$, and locate 2^{c1} collisions. We fix an arbitrary suffix C of length 2s, and use Mi = [i] ‖ C.

2. For each collision (i, j), use a binary search to find the distance μ_{ij} from the starting point to the collision, and denote the state reach after M_i (or M_j) by y_{ij} . Denote the set of all y_{ij} (containing about 2^{c1} states) by Y. Build an online diamond filter for all the states in Y.

3. Run a fixed-offset collision search by iterating the compression function with C from 2^t starting points.

4. We match each offline collision x, only with online collisions that occur at the same offset as x. Thus, for each x, we test only the end point of its chain (at offset 2^s) with the corresponding states in Y. Note that each x is matched with $2^{c1} \cdot 2^{-s}$ states in Y on average.



We generate collisions and build an online diamond filter, and match them with offline collisions using the collision offset as a first filter.

Figure 39: Short message attack for HMAC with HAIFA [70]

4.2.5 Sponge Construction

4.2.5.1 Slide attacks on “extended” sponge constructions

Assume that H is an iterative hash function with an internal state of c words of p-bit each and a final output size of n bits. Let M = M1||M2|| · · · ||Ml be the m×p-bit blocks of the message to hash with Ml ≠ 0m×p (the message is padded before split into blocks). Let Mi be the message block hashed at each round i and Xi the internal state after

proceeding Mi, with X0 = IV . We then have Xi = F(S(Xi−1, Mi)), where F is the round function and S defines how the message is incorporated in the internal state. Once all the l message blocks have been processed, r blank rounds are applied Xi = F(Xi−1) and A := Xl+r is the final internal state. Finally, we derivate n output bits by using the final output function T (Xl+r). Such a hash function can be written as

$$H(M) = X^0 \xrightarrow{F\left(S(X^0,M^1)\right)} \cdots \xrightarrow{F\left(S(X^{l-1},M^l)\right)} X^l \xrightarrow{F\left((X^l)\right)} \cdots \xrightarrow{F\left((X^{l+r-1})\right)} X^{l+r} \xrightarrow{T(A)} T(A),$$

where T_A represents the hash output. In the original model, S introduces the message blocks by XORing them to particular positions of the internal state. However, in this situation, we can also consider a function S that replaces some bits of the internal state by the message bits. In addition, in the original model, the final output function T continues to apply some blank rounds and extract some bits from the internal state at the end of each application, until n bits have been received. In this situation we consider the case where the output bits come from a direct truncation of the final internal state A, and we call it truncated sponge. There first issue, related to the general design of sponge functions is invertibility. This means that we can run the function F into both directions. The second issue is self-similarity, where all the blank rounds behave identically, and even a normal round can behave as a blank round if we have $X^{i-1} = S(X^{i-1}, M^i)$. In the case of a XOR sponge we need Mi = 0 and in the case of an overwrite sponge we require that Mi is equal to the overwritten part of the internal state. We will exploit self-similarity for our slide attacks. The idea is that if one message M1 =M1|| . . . ||Ml is the prefix of another message M2 = M1|| . . . ||Ml ||Ml+1, the extended state after processing the first l blocks is the same. Now, if Xl+1 = S(Xl, Ml+1), processing the next message block Ml+1 for the longer message is the same as the first blank round when hashing the shorter message – the extended states remain identical. We call these two messages a slid pair: the two final internal states are just one permutation away B := Xjl+r+1 = F(X) (l+r). The slide attack is shown in Figure below:

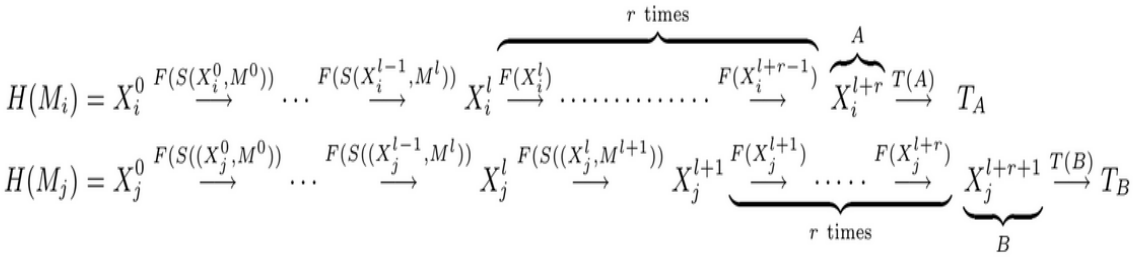


Figure 40: A slide attack on Hash Functions

Once we were able to generate a slid pair, we need to detect it. This fully depends on the output function T. When T is defined as in the original sponge framework, it is very easy to detect a slid pair: most of the output bits will be equal, just shifted by one round. If T is a truncation, we need to do a case by case analysis depending on the strength of the round function F and the number of bits thrown away. Yet finding and detecting a slid pair already allows us to differentiate the hash function from a random oracle. A step forward from this is by attacking a MAC with prefix key, i.e. MAC (K, M). Note that such a construction makes sense as using HMAC based on a sponge hash function will

turn out to be very inefficient. This is due to the fact that hashing very short messages is quite slow because of the blank rounds. Therefore, Bertoni et al. [28],[71] proposed to use prefix-MAC instead of HMAC.

Consider a secret key K. For simplicity and without loss of generality, we assume some K to be a uniformly distributed (k × m × p)-bit random value (i.e. k message words long), for some public integer constant k. We will write $K = (K^1, \ldots, K^m) \in (\{0, 1\}^{m \times p})^k$. The adversary is allowed to choose message challenges C_i, while the oracle replies $MAC(K, C_i) = H(K||C_i)$. Ideally, finding K in such a scenario would require the adversary to exhaustively search over the set of all possible $K \in \{0, 1\}^{k \times m \times p}$, thus taking $2^{k \times m \times p-1}$ units of time on average. Forging a valid MAC depends on the size of the hash output and the size of the key, with a generic attack it requires $\min\{2^{k \times m \times p-1}, 2^n\}$ units of time. A pair of challenges (C_i, C_j), with $C_i = {}^{C_i}{}^1 \text{ I } ||C_i^2 \ || \cdot \cdot \cdot ||C_i^l$ and $C_j = C_i ||C_j^l$ is called a slid pair for K if their final internal state are slid by one application of the blank round function as:

$$X_j^{k+l+r+1} = F(x_i^{k+l+r})$$

Provided that one can generate slid pairs and detect them, one can also try to retrieve the internal state X_i^{k+l+r} thanks to this information. Again, a case by case analysis is required here. When X_i^{k+l+r} is known, one can invert all the blank rounds and get X_i^{k+l}. Note that with this information, an attacker can directly forge valid MACs for any message that contains M as prefix (exactly like the extension attacks against MD-based hash functions). If the round function with the message is also invertible, we can continue to invert all the challenge rounds and get X_i^k. This will allow us to recover some non trivial information on the secret key K.

A general outline of the attack is as follows:

1. Find and detect slid pairs of messages
2. Recover the internal state
3. Uncover some part of the secret key or forge valid MACs

The padding is very important. For the XOR sponge functions, an appropriate padding can avoid slide attacks. Indeed, in that case, we require $M^l = 0^{m \times p}$ to get a slid pair. This gives an explanation why the condition $Ml \neq 0^{m \times p}$ is needed for the indifferentiability proofs of XOR sponge functions. However, for the truncated sponge function, a padding is ineffective to avoid slide attacks.

4.2.6 Wide –Pipe Construction

4.2.6.1 State recovery

We present an internal-state-recovery attack that is applicable to wide-pipe hash functions.

We observe that if walk A and walk B follow the structure in Figure 38, then for any query in the cycle of walk A, denoted as q^A, the inner hash value $H K_{in}(q^A)$ is necessarily equal to some query in the cycle of walk B, denoted as q^B. The goal is therefore to find this query among all q^B, all the members of walk B that belong to the cycle. That means that we want to coordinate the two cycles from walk A and walk B, which we already know that they have the same length.

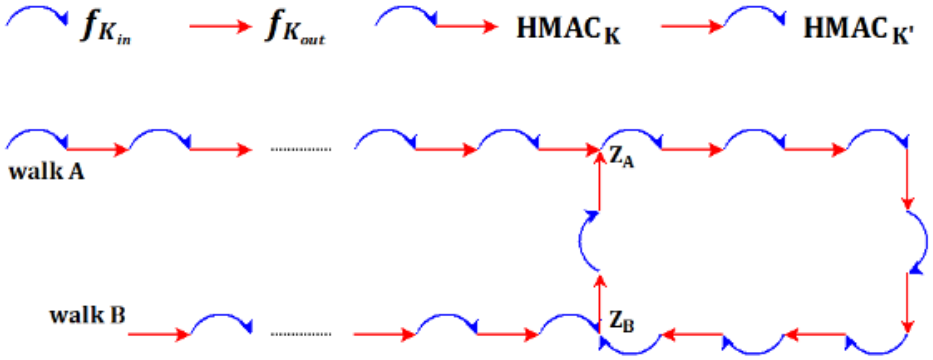


Figure 41: the cycle structure built with access to oracles f Kout◦ fKin and fKin ◦ fKout.

Mainly, even if we know that walk A and walk B have the same length and they are actually doing the same computations, it seems difficult to synchronize the two cycles because we do not know where the tail in walk A and in walk B is entering the cycle. However, in the special case where the collision between walk A and walk B happens in the tail (and not in the cycle), then we know that the tails are entering the cycle at the same position n that case, the cycles are directly synchronized and the attacker knows all the successive hash output values for every computation in the cycle. [72]

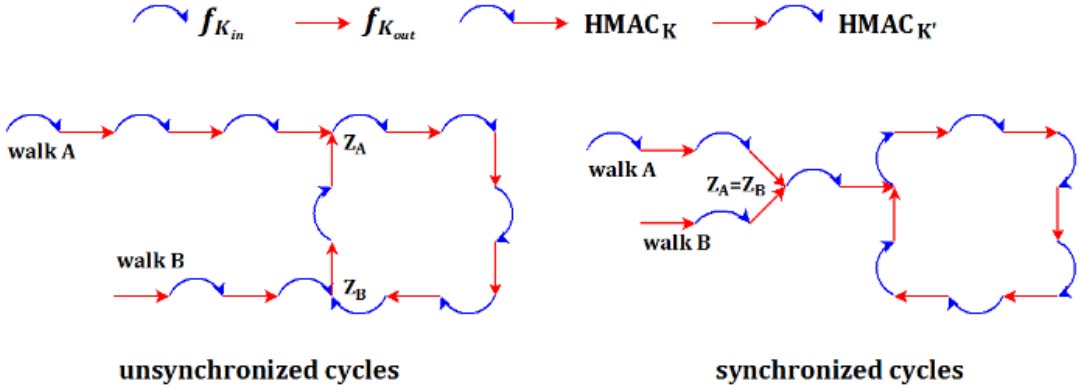


Figure 42: Two walks A and B colliding and sharing a cycle. The left example shows unsynchronized cycles (the collision happens in the cycle, thus ZA≠ZB), the right shows synchronized cycles (the collision happens before the cycle, in the tails, thus ZA=ZB).

The first and second phases of the attack will be devoted to building a walk A and walk B with a rather long tail, such that during the third phase there is a good chance to get a collision between an element of the tail of walk A and an element of the tail of walk B. In order to recover an internal state, he will focus on one randomly chosen value belonging to the cycle, denoted q_A, and its next hash output q_B, with $q_B=H(K_{in},q_A)$. Then he will try to guess the internal hash value $X=h(h(IV,K_{in}),q_A||pad_1)$ that led to q_B, i.e.g $(X) =q_B$. We assume that $g(\cdot)$ is easy to invert (given an output u, it is easy to find all preimages leading to u) and that it is balanced (given an output value, there exists 2^{l-n} corresponding input values through g). Inverting g provides 2^{l-n} candidates X_i such that $g(X_i) =q_B$. For each of these candidates, we will apply a

filter to remove the bad guesses. The filter is based on an offline extension of the computation of $H_{K_{in}}$ [72].

4.3 Vulnerability analysis of recent hash functions

Generally is preferable to be impossible to break security properties. A hash function is called broken when there exists a known explicit attack that is faster than the general attack for a security property. It must be noted that even unbroken hash functions may be insecure in the real-world. The best known general attack to break Pre, aPre, ePre, Sec, aSec and eSec is a brute force search, where hashes f(M′) are computed for randomly chosen messages M′ until a message M′ is found where f(M′) is the target hash value (and M′≠ M for Sec, aSec, eSec). For a hash function (family) with an output hash size of N bits, this attack succeeds after approximately 2N evaluations of the hash function. Already for N ≥ 100 this attack is clearly infeasible in the real world for the present day and near future.

4.3.1 Comparative analysis between (MD5, SHA-1, SHA-2)

Attacks on MD5

In 1993, B. Den Boer and A. Bosselaers found a kind of Pseudo-Collision with complexity 2^{16} for MD5 which consists of the same message with two different sets of initial values [73]

In 1996, H. Dobbertin presented a free start collision with complexity 2^{34} for MD5 during the rump session of EUROCRYPT'96 [74].

In 2005, Wang et.al found collisions with 2^{39} hash operations for MD5 [75].

In 2013, Xie Tao, Fanbaoliu and Dengguo published an attack that breaks MD5 collision resistance in 2^{18}. This attack runs in less than a second on a typical modern computer [76].

Attacks on SHA-1

In 2005, Biham et al published a theoretical attack on a reduced version of SHA-1(58 out of 80 rounds) which finds collision with a computational effort of 2^{75} operations (fewer than 2^{80} operations) [11].

In 2005, Wang et. al published an improvement on the SHA-1 attack at the CRYPTO 2005 rump session, lowering the complexity required for finding a collision in SHA-1 to 2^{69} [12].

In 2010 Marc Steven presents an identical prefix collision attack against up to 46 rounds of SHA-256 attack on SHA-1 with complexities equivalent to approximately 2^{61}(theoretical) [77].

Attacks on SHA-2

In 2011, Mario Lamberger and Florian Mendel published the best attack, which is Pseudo Collision attack against up to 46 rounds of SHA-256 [78].

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Word size (bits)	Rounds	Operations	Collisions found
SHA-0		160	160	512	$2^{64} - 1$	32	80	+,and,or,xor,rot	Yes
SHA-1		160	160	512	$2^{64} - 1$	32	80	+,and,or,xor,rot	None (2^{52} attack)
SHA-2	*SHA-256/224*	256/224	256	512	$2^{64} - 1$	32	64	+,and,or,xor,shr,rot	None
	SHA-512/384	512/384	512	1024	$2^{128} - 1$	64	80	+,and,or,xor,shr,rot	None

Figure 43: Differences in SHA family

It is clear that new hash functions or new methods of employing hash functions are necessary. Some popular hash functions, which are widely used are MD5, SHA-1 but after finding collisions in them, the designers focus in the creation of new secure and faster hash functions. It has been observed that MD5 is fast but proven inadequate, as now it no longer remains collision resistance. Security of SHA-1 is also questionable. So in this paper new hash algorithms are proposed like Whirlpool, BLAKE-256, JH Hash, Keccak, Streebog and Kangaroo Twelve are products of such new generation of hash functions.

Therefore, it can be concluded that a hash and authenticity, must be designed and made into a priority.

The MD5, SHA-1, SHA-2 are very popular hash functions. But, after finding collisions in popular hash functions as MD-5 and SHA-1, focus got shifted towards designing new secure and faster hashes functions. BLAKE- 256, Whirlpool, JH Hash etc. are products of such new generation of hash functions. Although they used to follow, more or less, the same Merkle-Damgård construction, but each of design has modified this construction for better security and improved performance results, for example JH hash function has included bit slicing. The designers are also working on modification of these new designs for better performance, so that, the attacks which are possible as of now, may not perform in coming times and we may get full-proof hash functions.

4.3.2 Comparison with other functions

Table 3: Survey of the best known attacks on secure hash functions

Name	Year	bits	cpb	Colission attacks				Sec preimage attacks				construction
				Safe?	comp	mem	ref	Safe?	comp	mem	ref	
MD2	89	128	638	no	2^{64}	2^{0}	[79]	yes	2^{72}	2^{72}	[75]	Merkle–Damgård
Snefru-2	90	128	?	no	2^{13}	2^{0}	[75]	no	2^{25}	2^{0}	[75]	Merkle–

[75]												Damgård
MD4	90	128	4.0	no	2^{2}	2^{0}	[75]	yes	2^{95}	2^{38}	[80]	Merkle–Damgård
RIPEMD	90	128	?	no	2^{18}	2^{0}	[75]	yes				Merkle–Damgård
MD5	92	128	5.1	no	2^{24}	2^{0}	[81]	yes	2^{123}	2^{48}	[82]	Merkle–Damgård
HAVAL-256-3 [75]	92	256	?	no	2^{29}	2^{0}	[83]	yes	2^{225}	2^{68}	[44]	Merkle–Damgård
SHA-0	93	160	?	no	2^{34}	2^{0}	[84]	yes	2^{189}	2^{8}		Merkle–Damgård
GOST	94	256	?	maybe	2^{105}	2^{0}	[71]	yes	2^{192}	2^{70}	[71]	AES design
SHA-1	95	160	18	no	2^{63}	2^{0}	[43]	yes				Merkle–Damgård
RIPEMD-160 [85]	96	160	17	maybe	2^{80}	2^{0}		yes				Merkle–Damgård
Tiger [86]	96	192	6.2	yes				yes	2^{189}	2^{8}	[28]	Merkle–Damgård
Panama [87]	98	512	2.5	no	2^{6}	2^{0}	[79]	yes				
Whirlpool [41]	00	512	50	yes				yes				Merkle–Dam

												gård
SHA-256 [75][86]	01	256	19	yes				yes				Merkle–Damgård
Radio Gatún [75]	06	256	?	yes				yes				ideal mangling function
Skein [75]	08	256	8.7	yes				yes				Unique Block Iteration
Blake 3 [75]	08	256	17	yes				yes				Merkle tree
Grøstl [75]	08	256	24	yes				yes				AES design
Keccak (SHA-3) [88]	08	256	16	yes				yes				sponge
JH [77]	08	256	20	yes				yes				Merkle–Damgård
BLAKE2 [86]	12	256	5.7	yes				yes				HAIFA structure
Streebog	12	256/512	12	yes				yes				Merkle–Damgård
KangarooTwelve	16	128	<1,5	yes				yes				Sponge construction

legend:

bit: the number of bits of output
cpb: cycles per byte [*]
comp: approximate computation required for the attack
mem: approximate memory required for the attack

The main result of this investigation is that there is a big gap between the historical successes of collision attacks and the almost non-existence successes of pre-image attacks. This is evidence that a cryptosystem invulnerable to collision attacks is much safer than one that is vulnerable to collision attacks (regardless of whether it is vulnerable to pre-image attacks).

Another interesting pattern in these results is that maybe sometime between 1995 (SHA-1) and 2000 (Whirlpool), humanity learned how to make collision-resistant hash functions, and none of the prominent secure hash functions designed since that era have succumbed to collision attacks. Maybe modern hash functions like SHA-256, SHA-3, and BLAKE2 will never be broken.

4.3.3 Summary of Vulnerability Analysis per Hash Construction

Table 4: Vulnerability Analysis per Hash Construction

Attacks/ Construction	Brute Force Attack	Pre-image	2nd Pre-image	Collision	Joux's Multicollision	Short message	State-recovery	Slide
Merkle-Damgård	✓	✓	✓	✓	✓	✓		
HAIFA	✓					✓	✓	
Sponge	✓							✓
Wide- Pipe	✓						✓	
Tree-Based	✓	✓			✓			

The majority of the popular hash functions are based on the famous Merkle –Damgård construction. As we notice in the table below, several weaknesses are found in this construction giving raise to a class of generic attacks that is applicable to any hash function based on the Merkle- Damgård construction.

Brute Force Attack is applicable on all hash functions independent of the structure.

Table 5: the timeline of attacks and their complexity

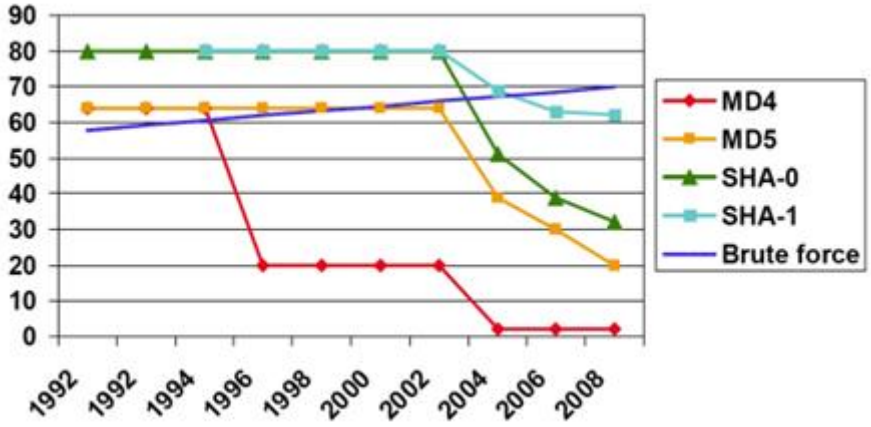


Table 6: Feature Comparison of Hashing Algorithms

FEATURES	Hashing Algorithm		
	MD-5	**SHA-1**	**BLAKE-2**
Security	Less secure than SHA-1	More Secure	Secure as SHA-3
Length of message digest	128 bits	160 bits	256 bits or 512 bits
No. of attacks needed to find original message	$2^{123.4}$ bit operations required [75]	$2^{151.1}$ bit operations required [80]	2^{256} or 2^{512} (exhaustive search)
Attacks to try and find two message producing the same MD	$2^{49.8}$ bit operations required [75]	Between $2^{60.3}$ and $2^{65.3}$ bit operations [75]	2^{256} or 2^{256} (exhaustive search)
Speed	Faster 60 iterations	Slower 80 iterations	Faster than SHA and MD
Successful attacks reported	YES	YES	NO

5 ORDER PRESERVING MINIMAL PERFECT HASH FUNCTONS APROACHES

Many systems and applications have to ensure in express access to information and objects in large network databases. When the fastest possible direct search is craved we usually apply hashing.

5.1.1 PHF (Perfect Hash Functions)

We use optimal hashing techniques to make operations as efficient as possible, providing:

- One-probe access to a record, given its key
- No collisions to be resolved
- Full utilization of hash table space.

When we referred in Optimal speed for hashing means that each key from the key set will map to a unique location in the hash table thus avoiding time wasted in resolving collisions. That is achieved with a perfect hash function (PHF), whose operation is illustrated at the Figure 41 [89].

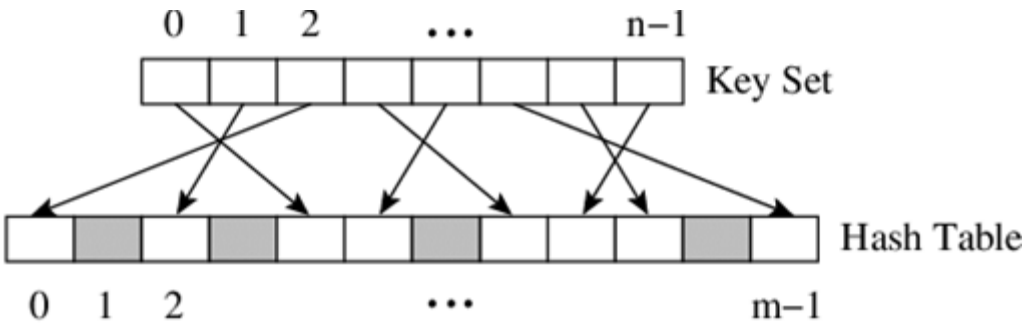


Figure 44: Perfect Hash Functions [89]

5.1.2 MPHF (Minimal Perfect Hash Functions)

When the hash table has minimal size, i.e. is fully loaded, with ISI=IT|, the hash function is called minimal. When both properties keep, we can say that we have a minimal perfect hash function (MPHF) as shown at the bottom of Figure. Note that, in reality, key set itself is usually neither ordered nor sequential, but can clearly be indexed by the integers (1.., n–1 for convenience of illustration [89]

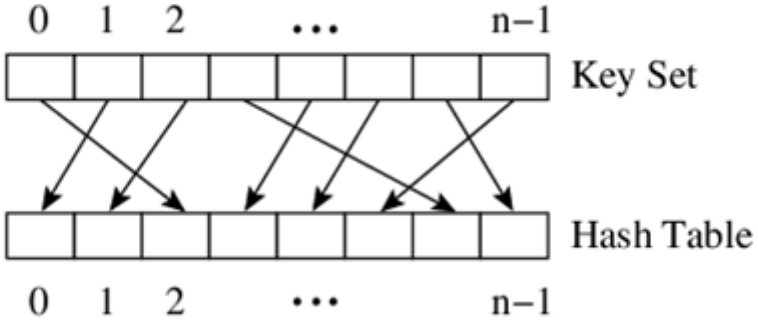


Figure 45: Minimal Perfect Hash Functions [89]

MPHF Algorithm

To facilitate discussion, we give a description of the terminology.

- U: key universe IUI=N.
- S: actual key set S c U, ISI=n<<N.
- T: hashtable IT|=m, m>n.
- h: hash function h:U → T
- h is a perfect hash function (PHF): no collisions, h is one-to-one on S.
- h is a minimal perfect hash function (MPHF):no collisions and m=n.

For a given key set S taken from universe U, we desire a MPHF h that will map any key k in S to a unique slot in hash table T. Actually, Mapping and Ordering steps are essentials so that the rapid Searching can take place [89]

The (MOS) scheme is illustrated in figure bellow:

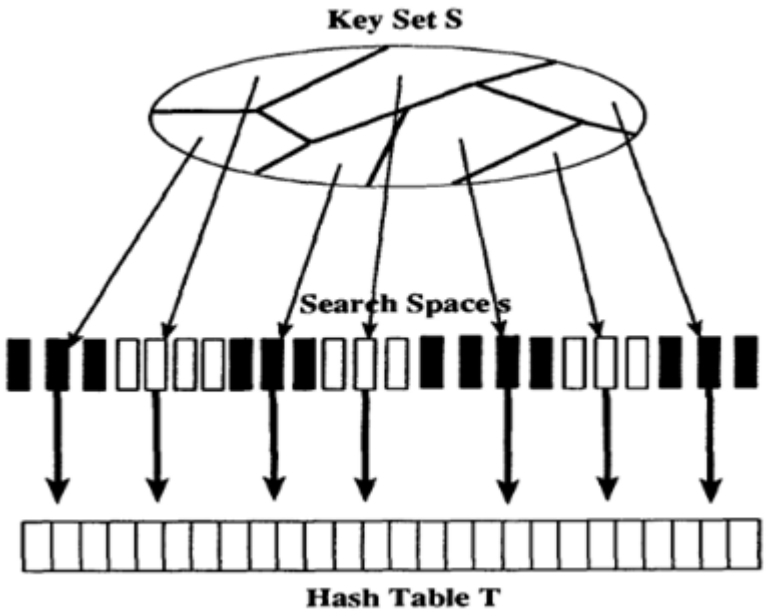


Figure 46: Illustration of the Key Concepts [89]

Mapping step converts the problem of hashing keys into a disparate problem, in a different space. Ordering step concretes the way for searching in the new space, in this way so the locations can be identified in the hash table. Hashing after that associates mapping from the keys into the new space, and adopting the results of Searching to find the proper hash table location [89].

This basic algorithm

- Is a probabilistic algorithm
- Is based on ordering the vertices in a bipartite dependency graph
- Requires expected linear running time

- Handles large sets containing millions of keys and
- Yields MPHFs of size $c \log_2 n$ bits per key ($0.5 < c < 1$).

We have to mention that this Algorithm requires less than one word of specification space for each key in S. However, this is significantly more space than the theoretical lower bound, which is roughly 1.5 bits per key [89].

5.1.3 OPMPHF (Order Preserving Minimal Perfect Hash Functions)

While dynamic hashing generally does not preserve the original key ordering, we can use order-preserving key transformations, which are appropriate for dynamic key sets as long as the key distributions are or can be made to be stable [GARG86]. In contrast, we made the very useful assumption that our key sets are static, and investigated published algorithms for finding minimal perfect hash functions MPHFs [DATT88]).

Our interest focuses on MPHFs that also have the property of preserving the order of the input key set. To specify what is implied, consider Figure below. A function must be obtained that maps keys, usually in the form of character strings or concatenations of several numeric fields, into hash table locations. In brief, the i_{th} key is mapped into the i_{th} hash table location.

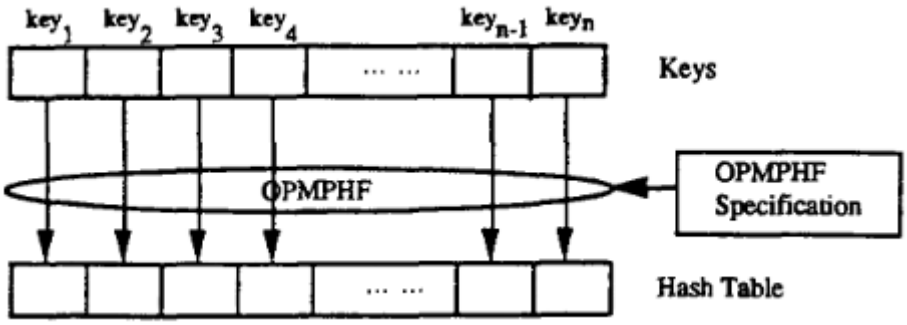


Figure 47: Illustration of the Key Concepts

5.1.4 Order preserving encryption (OPE)

An order-preserving symmetric encryption (OPE) scheme is a deterministic symmetric encryption scheme whose encryption algorithm produces ciphertexts that preserve numerical ordering of the plaintexts. OPE was proposed in the database community by Agrawal et al. [d] in 2004 as a tool to support efficient range queries on encrypted data.

We want to have deterministic encryption schemes that preserve numerical ordering on their plaintext-space. For $A, B \subseteq N$ with $|A| \leq |B|$, a function $f : A \rightarrow B$ is order-preserving if for all $i, j \in A$, $f(i) > f(j)$ if $i > j$.

We say that deterministic encryption scheme SE = (K, Enc, Dec) with plaintext and ciphertext-spaces D, R is order-preserving if Enc(K, ·) is an order-preserving function from D to R for all K output by K (with elements of D, R interpreted as numbers, encoded as strings). Unless otherwise stated, we assume the plaintext-space is [M] and the ciphertext- space is [N] for some $N \geq M \in N$ [90].

5.2 Related OPMPHF'S

5.2.1 A method for MPHF's (Pascal reserved words)

A method is presented for computing machine independent, minimal perfect hash functions known as Pascal's Reserved Words. The space S consists of words and divided according to the frequencies of the occurrences of the letters. The associated values of each letter have the form below:

hash value ← key length + the associated value

of the key's first character + the associated value of the key's last character

For Pascal's 36 reserved words, there is a specific list that defines the associated value for each letter and the corresponding hash table with hash values running from 2 through 37.

For example consider that the associated value for letter C is 1 and the value for letter E is 0 and we want the computation for word "CASE". So we have the procedure:

(1← "C") + (0 ←"E") + (4 ← length ("CASE")) = 5.

After the words have been put in order by character occurrence frequencies, the order is modified of the list such that any word whose hash value is determined by assigning the associated character values already determined by The backtracking search procedure then attempts to find a set of associated values which will permit the unique referencing of all members of the key word list. It does this by trying the words one at a time in order previous words is placed next. Each "try" tests whether the given hash value is already assigned and, if not, reserves the value and assigns the letters.

The search time for computing such functions is related to the number of identifiers to be placed, the maximum value which is allowed to be associated with a character, and the density of the resultant hash table. If the table density is one (i.e., a minimal perfect hash) and the maximum associated value is allowed to be the count of distinct first and last letter occurrences (21 for Pascal's reserved words), then the above procedure finds a solution for Pascal's reserved words in about seven seconds on a DEC PDP-11/45 using a straightforward implementation of the algorithm in Pascal. Incorporation of the above hash function into a Pascal cross-reference program yielded a 10 percent reduction in total run time for processing large programs [91].

5.2.2 Random Order preserving hash function (ROPF)

An OPE scheme is secure if oracle access to its encryption function is indistinguishable from oracle access to a random order-preserving function (ROPF) on the same domain and range. Any secure OPE scheme (including the only currently known block cipher-based scheme should "closely" imitate the behavior of an ROPF. So, a good idea is to focus on analyzing the ideal object, an ROPF.

We define the "ideal" ROPF scheme as follows:

Let $OPF_{D,R}$ denote the set of all order-preserving functions from D to R. Define

$ROPF_{D,R} = (Kr, Enc_r, Dec_r)$ as the following encryption scheme:

- K_r returns a random element g of $OPF_{D,R}$.
- Enc_r takes the key and a plaintext m to return g(m).

• Dec_r takes the key and a ciphertext c to return $g^{-1}(c)$.

The above scheme is not computationally efficient, but our goal is its security analysis for the purpose of clarifying security of all POPF-secure constructions.

Most Likely Plaintext: Fix a symmetric encryption scheme SED,R = (K, Enc,Dec). For given c ∈ R, if mc ∈ D is a message such that

$$\Pr [K \leftarrow K : Enc(K, m) = c]$$

achieves a maximum at m = m_c, then we call m_c a (if unique, “the”) most likely plaintext for c.

Most Likely Plaintext Distance: Fix a symmetric encryption scheme $SE_{[M],[N]}$ = (K, Enc, Dec). For given c_1, c_2 ∈ R, if $d_{c1,c2}$ ∈ {0, 1,...,M − 1}

Such that Pr [K $^{\$}$ ← K : (c_1, c_2) = Enc(K,(m_1, m_2)) ; m_2 − m_1 mod M = d]

achieves a maximum at d = $d_{c1,c2}$, then we call $d_{c1,c2}$ a (if unique, “the”) most likely plaintext distance from c_1 to c_2 [90].

5.2.3 Content Addressable Network (CAN)

A CAN network is a decentralized Peer-to-Peer infrastructure that can be represented as a d-dimensional coordinate space. Let us consider an overlay made of n peers. Each peer is responsible for the zone it holds in the network (a set of intervals in this space). All dimensions have a minimum and a maximum CAN-based value C_{min} and C_{max}.

For instance, Figure 38 presents a two-dimensional CAN overlay where each peer manages a zone bounded by an interval on each dimension. Each peer’s interval is constant and can only be modified during join or leave node operations.

Each peer can only communicate with its neighbors, thus routing from neighbor to neighbor has to be done in order to reach remote zones in the network. The CAN topology is a torus which means, in Figure 38, that peers p1 and p3 are neighbors on the horizontal dimension [92].

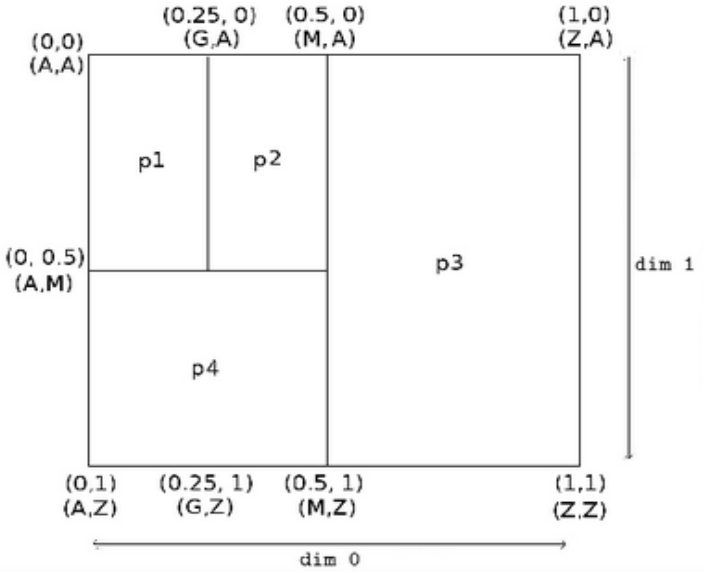


Figure 48: example of a two-dimensional unicode CAN storing items [92]

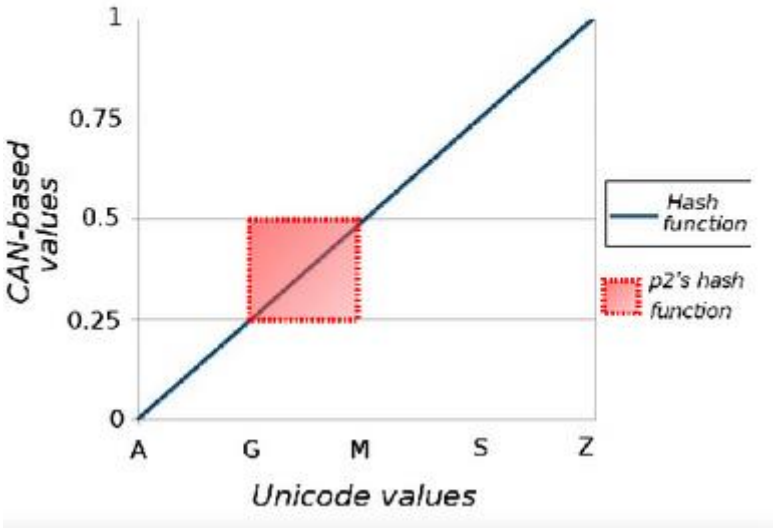


Figure 49: standard hash function on dimension 0 [92]

Minimum and maximum unicode values U_{min} and U_{max} are set to determine the unicode range that can be managed within the overlay: in Figure 45, U_{min} is equal to A and U_{max} is equal to Z. A unicode value is associated with each bound of a peer and a peer is responsible for storing the items whose unicode value falls between these bounds. For example, in Figure 45, p1 is responsible for data between [A; G[(coordinates included in [0; 0:25[) on the horizontal dimension, and [A; M[(coordinates included in [0; 0:5[) on the vertical one.

The mapping relation between overlay-based coordinates and unicode-based values can be seen as a form of hash function. Indeed, to each unicode value corresponds a coordinate between O_{min} and O_{max}, obtained by applying a given hash function on the unicode value. This hash function provides coordinates that determine where a data item should be stored. The default hash function for the CAN of Figure 45 is shown in Figure 46. The graph describes which CAN coordinate is associated with which unicode value, hence each peer is associated with a segment of the function [p2's segment is highlighted in Figure 46]. For instance, 'G' corresponds to coordinate 0.25 on the hash function graph, which means this value is managed by p2 on dimension 0 because its interval is [0.25; 0.5]

When a peer receives a new data item to insert or a query to execute, it has to convert the unicode-encoded values into coordinates to check whether it is responsible for this item/query or not. For example, the hashed value of string ProductType1 in the context of an overlay storing worldwide data (wide unicode range, up to code point value 220, as depicted in Figure 47 would be equal to coordinate 0.00004673 (i.e. at the far-left in the identifier space). By default, strings made of Latin characters have a low hashed value, whereas strings made of any East-Asian characters have high values (close to Omax) because such characters are located towards the end of the unicode table. If the hashed value does not match the peer's coordinates, it means the peer is not responsible for the corresponding item. In this case, the peer forwards the item/query to a neighbor managing an interval closer to the requested one.

The order-preserving storage technique presented above suffers from a major drawback regarding data distribution. Indeed, having a system covering the whole Unicode range means potential overloaded areas may appear, depending on data distribution. Figure 40 describes a system where only triples made of Latin characters are stored, which means only a small area of the CAN is targeted when inserting or querying data. In consequence, peer p1 becomes overloaded, while the rest of the network stores nothing as it is dedicated to other Unicode characters. Based on this observation, our contribution aims at dynamically adapting the size of skewed Unicode areas in a CAN, by changing hash functions to determine where data should be stored. We will present hereafter our notion of variable hash function and how it helps balance the load of a storage system [92].

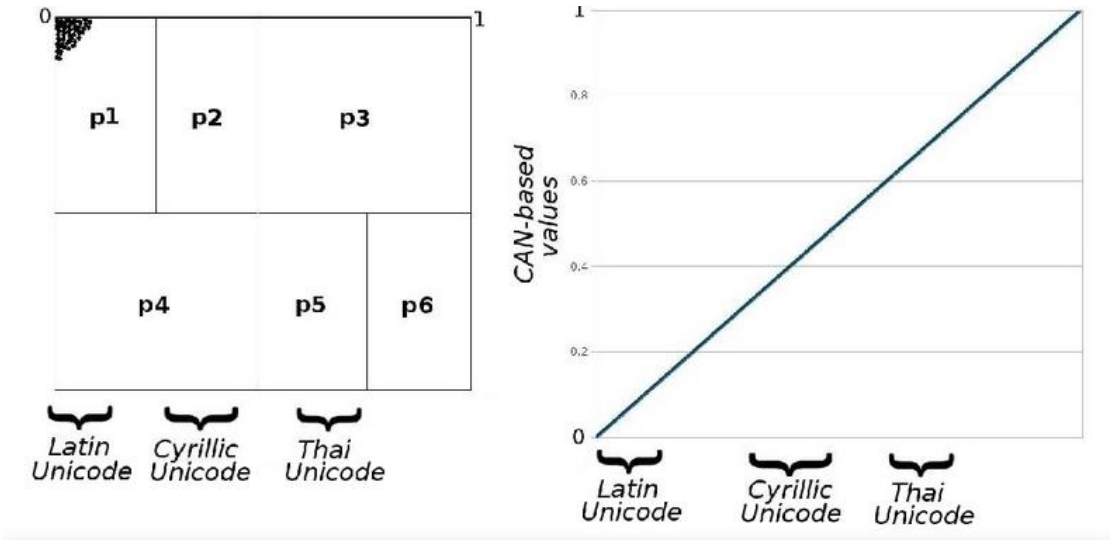


Figure 50: Default hash function inefficient to disseminate data items (represented as black dots at the top-left corner of the CAN) [92].

5.2.4 Acyclic Graphs

In this technique, it is constructed a bipartite graph G, enough large so that no cycles are present. A bipartite graph is a graph whose vertices can be divided in two disjoint and independent sets U,V such that every edge connects a vertex in U to one in V. Vertex sets U and V are usually called the parts of the graph. This method is based on the use of a large ratio (2r/n) which leads to the probability of having a cycle approach to zero.

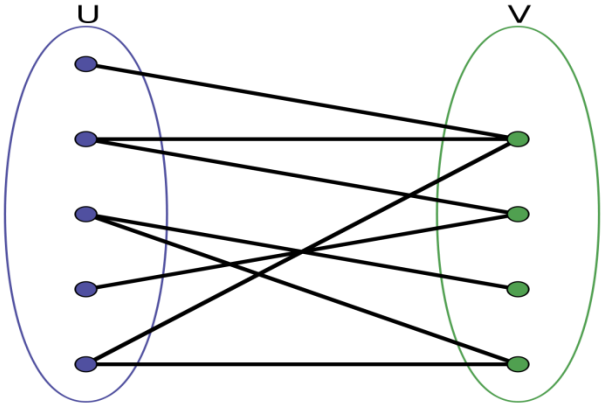


Figure 51: A Bipatrite Graph

We assume that a bipartite graph having 2r vertices on each side and having n random edges. Let Pr (2i) be the probability of having a cycle of length 2i formed in a particular vertex set of 2i vertices, with i vertices being on each side. There are $\frac{i!(i-1)!}{2}$

ways to form distinct cycles out of these 2ivertices and $\binom{n}{2i}(2i)!$ Ways to select 2i edges to form such a cycle. The remaining $n-2i$ edges can go into G in$(r^2)^{n-2i}$ different ways.

Thus in total there are $i!\ \frac{(i-1)!}{2}\binom{n}{2i}(2i)!\ (r^2)^{n-2i}$ ways to form the 2i edge in the vertex set. Given a total of $(r^2)^n$ possibilities,

$$\left(\frac{(i-1)!}{2}\binom{n}{2i}(2i)!\ (r^2)^{n-\frac{2i}{r^{2n}}}\right)$$

Let Z_{ij} be an indicator random variable. $Z_{ij} = 1$ if there is a 2i edge cycle in the j^{th} vertex set of 2i vertices, $Z_{ij} = 0$ otherwise. Clearly, there are $\binom{r}{i}^2$such sets in G

Each vertex set has the same probability of having 2i edge cycles.

Let X_i be a random variable counting the number of 2i edge cycles in G.

We have $X_i = \sum_{j=1}^{\binom{r}{i}^2} Z_{ij} = \binom{r}{i}^2(2i)$

Define $Y_c = \sum_{i=1} r = X_i$ as another random variable counting the number of cycles in G of length from 2 to 2^r.

$$E(Y_c) = \sum_{i=1}^{r} E(Xi)$$

$$= \sum_{i=1}^{r} \binom{r}{i}^2 \Pr(2i)$$

$$= \sum_{i=1}^{r} \binom{r}{i}^2 i! \, \frac{(i-1)!}{2r^{4i}} (2i)! \, \binom{n}{2i}$$

$$\cong \sum_{i=1}^{r} \left(\frac{r^i}{i!} \, e^{\frac{-i^2}{2r}} \right)^2 \frac{i!\,(i-1)!\,(2i)!}{2r^{4i}} \left(\frac{n^{2i}}{(2i)!} 2i \, e^{\frac{-2i^2}{2n}} \right)$$

$$= \sum_{i=1}^{r} \frac{1}{2i} \, \binom{n}{r}^{2i} \, e^{-i^2 \left(\frac{1}{r}+\frac{2}{n}\right)}$$

$$= \sum_{i=1}^{r} \frac{1}{2} \, \binom{n}{r}^{2i}$$

$$\leq \sum_{i=1}^{\infty} \frac{1}{2} \, \binom{n}{r}^{2i}$$

$$= \frac{1}{2} \frac{\binom{n}{r}^2}{\left(1-\binom{n}{r}^2\right)}$$

then $E(Y_c) \leq \frac{1}{\left(\binom{r}{n}^2 - 1\right)}$

when $r = n \, logn, E(Y_C) \to 0 \; as \; n \; \to \infty$

If there are no cycles, we have sufficient freedom during the Searching phase to select g values that will preserve any a priori key order. Because G is acyclic, we obtain an ordering of non-zero degree vertices v to yield levels K(v) following certain constraints which only contain one edge (one key). This is achieved through an edge traversal (e.g., depth-first or breadth-first) of all components in G, In figure 41, there is an acyclic bipartite graph, ordering obtained by depth-first traversal of first the left connected component and then the right might give the vertex sequence (VS)) : [v1, v5 , v0 , v2 , v6 , v3 , v1]. The corresponding levels of edges are given in the edge sequence: [{}, {e1}, {e0 }, {e3}, {e2 }, {}, {e4 }]. In this example, each level has at most one edge, which is only possible if G is acyclic [93].

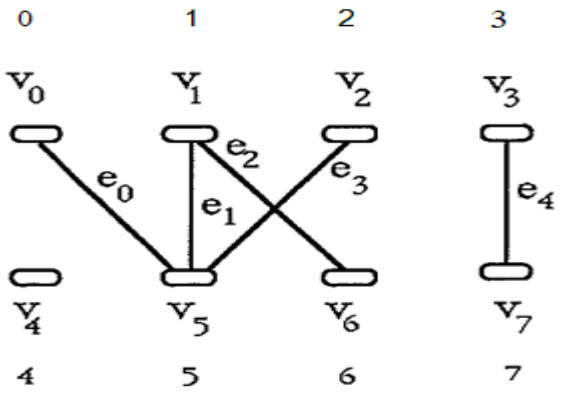


Figure 52: A Cycle Free Bipatrite Graph [93]

During the Searching phase, a single pass through the ordering can determine g values for all keys in a manner that preserves the original key ordering. This is possible since with only one edge being handled at each level, there are no interdependencies that would restrict the g value assignments [93].

5.2.5 Two Level Hashing

The second approach is to use two level hashing. We have the MPHF in the first level and an array of pointers in the second level. A hash value from the MPHF addresses the second level where the real locations of records are kept. The records are arranged in the desired order. This method uses at the first level 2r, and at the second, n computer words for the OPMPHF. Figure 42 illustrates the two level hashing schemes [93].

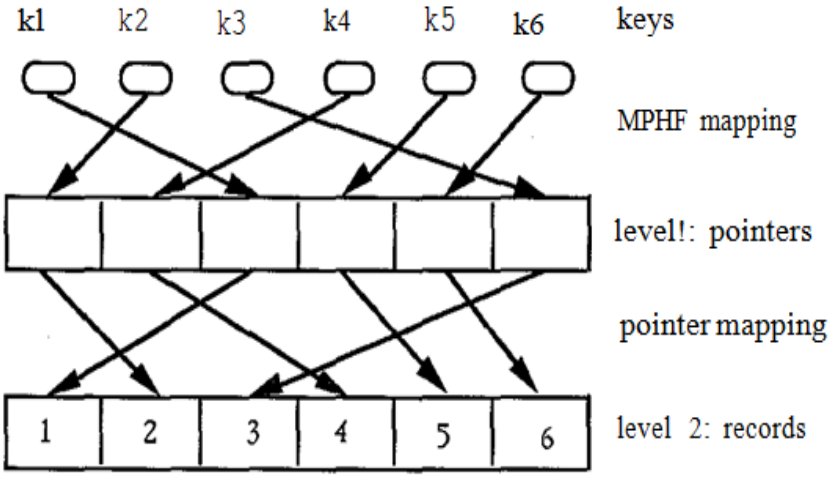


Figure 53: A two Level OPMPHF Scheme [93]

5.2.6 Using Direction

The third technique is based on the idea of using G to store the additional information required to specify an OPMPHF. For n keys, if the graph has somewhat more than n vertices (i.e., if ratio > 1), then there should be enough room to specify the OPMPHF. In

a random graph of this size, a significant number of vertices will have zero degree. We have to use indirection for some of the keys, in this case using the composition:

$h(k) = g(\{h_o(k) + g(h_1(k)) + g(h_2(k))\} \text{mod} 2r)$ while on the other hand, the desired location of a key that is, as before, found directly is determined by:

$h(k) = \{h_0 (k) + g(h_1 k)) + g(h_2(k))\} \text{ mod } n.$

We use the g function in two ways, one way for regular keys and the other way for keys that are handled through indirection. The actual distribution is binomial and can be approximated by the Poisson:

$$E(x = d) = \frac{\left\{2re^{-\frac{n}{r}} \left(\frac{n}{r}\right)^d\right\}}{d!}$$

$$E(x = d) = \left\{2re^{-\frac{n}{r}}\right\}$$

When 2r = n, about 13.5% of the vertices have zero-degree. If these zero-degree vertices can be used to record order information for a significant number of keys, then it is not necessary for G to be acyclic to generate an OPMPHF. Note that keys associated with edges e_0 and e_i can be indirectly hashed into zero-degree vertices v_6 and v_2. In general, an edge (key) is indirectly hashed when that situation is described by information associated with its two vertices, given by $h_1(k)$ and $h_2(k)$. Usually, indirection can be indicated using one bit per vertex that is decided at MPHF building time and that is subsequently kept for use during function application time [93].

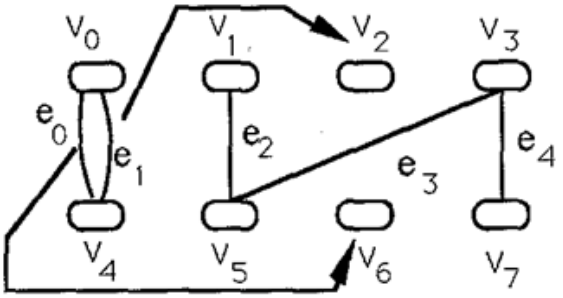


Figure 54: Zero Degree Vertices are Useful [93]

6 PROPOSED SECURE OPMPF ALGORITHM

6.1 Introduction

Hashing methods for no static sets of keys have a certain amount of wasted space and time. The space is wasted due to unused locations in a table and time is wasted because we have to resolve collisions when the keys are hashed to the same table location. But if the keys are static, then is possible to compute a hash function h(x) to find any key in the table with no collisions in this case and that is a perfect hash function. If a perfect hash function can also preserve an a priori key ordering, then is called an order preserving perfect hash function. A perfect hash function that can stores a set of records in a table of the size equal to the number keys is called minimal perfect hash function which avoids the wasted space and time.

Minimal perfect hash functions are used for memory efficient and fast retrieval of items from static sets, such as universal resource locations in Web search engines etc.

To find perfect hash functions may not be easy. According to Knuth [94], the total number of possible hash functions from S (|S|= n) into [0, m-1] (m ≥n) is m^n and only m(m -1) . . . (m-n+1) are perfect. Thus, the probability that no collisions occur is the ratio (m(m-1) . . . (m-n+ 1))/m^n which tends to zero very fast. For m = 13 and n = 10, the probability that no collisions occur is only 0.0074 [94].

We present a three- step algorithm for generating minimal perfect hash functions. This method uses a Mapping, Ordering, Searching (MOS) approach and the construction of a minimal perfect hash function is accomplished in three steps as below:

In the first step the mapping transforms a set of keys from the original universe to a universe of hash identifiers. A hash identifier is a collection of selected properties of a key, such as symbols comprising the key, their positions of occurrence in the key, the key length, etc. This step has to preserve “uniqueness”, i.e. if two keys are distinguishable in the original universe, they must also be distinguishable in the hash universe.

The second step is ordering and in this step the key is placed in a sequence which determines the precedence in which hash values are assigned to keys. Keys are divided into subsets W_0, W_1,…, W_k.

The third step, searching, tries to extend the desired hash function h from the domain W_{i-1} to W_i. This is the only step of potentially exponential time complexity, since if the searching step encounters Wi for which h cannot be extended, it backtracks to earlier subsets, assigns different hash values to the keys of these subsets and tries again to recomputed hash values for subsequent subsets [89].

One of the most efficient and practical algorithms for generating order preserving minimal perfect hash functions, involves the generation of acyclic random graphs

G = (V, E), where |V| = cn and |E| = n [95], [96], [97].

6.2 Basic Concept

Consider S be a set of n distinct keys belonging to a finite universe U of keys. The keys in S are stored so that queries asking if key x ϵ U is in S can be answered. If the set of keys is static, then it is possible to compute a hash function h(x) to find any key in the table with no collisions and this function is called perfect hash function. [95].

The algorithm for selecting proper g values and setting mark (indirection) bits for vertices in G consists of the three steps: Mapping, Ordering, and Searching [89]. The Mapping step builds random tables the three functions h_0, h_1, and h_2 that map each key k into a unique triple ($h_0(k)$, $h_1(k)$, $h_2(k)$). The $h_0(k)$, $h_1(k)$, $h_2(k)$ triples are used to build a bipartite graph (called dependency graph). Let k_1, k_2, ...,k_n be the set of keys. The $h_0(k)$, $h_1(k)$, and $h_2(k)$ functions are selected as the result of building tables of random numbers. If triples are not distinct, new random tables are generated, defining new $h_0(k)$, $h_1(k)$, $h_2(k)$ functions.

For a given undirected graph G = (V, E), where |V| = cn and |E| = n, find a function g:V > {0,1,... , |V| -1} such that the function h : E > {0,1,..., n -1}, defined as h(e)= (g(a) + g(b)) mod n (1) is a bijection, where e = {a, b}.

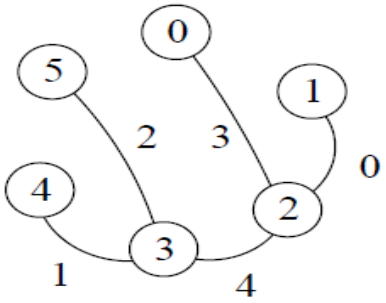
This means that we are looking for an assignment of values to vertices so that for each edge the sum of values associated with endpoints taken modulo the number of edges is a unique integer in the range [0, n -1]. The ordering and searching steps of the MOS approach are a very simple way of solving the perfect assignment problem. Czech, Havas and Majewski [95] showed that the perfect assignment problem can be solved in optimal time if G is acyclic. To generate an acyclic graph two vertices $h_1(x)$ and $h_2(x)$ are computed for each key x ϵ S. Thus, set S has a corresponding graph G, with V = {0,1,..., v} and E = {{$h_1(x)$, $h_2(x)$}: x ϵ S}.

We want to have acyclic graphs, so the algorithm repeatedly selects h_1 and h_2 until the corresponding graph is acyclic. In order to be useful the solution, we must have |S|= n and |V|= cn, for some constant c, such that acyclic graphs dominate the space of all random graphs. Havas et al. [98] proved that if |V|= cn holds with c > 2 the probability that G is acyclic is

$$p = e^{\frac{1}{c}} \sqrt{\frac{c-2}{c}}$$

For c = 2.09 the probability of a random graph being acyclic is p> 1/3. Consequently, for such c, the expected number of iterations to obtain an acyclic graph is lower than 3 and the g function needs 2.09n integer numbers to be stored, since its domain is the set V [95].

Given an acyclic graph G, for the ordering step we associate with each edge an unique number h(e) ϵ [0, n -1] in the order of the keys of S to obtain an order preserving function. Figure 55 illustrates the perfect assignment problem for an acyclic graph with six vertices and with the five table entries assigned to the edges.



v	$g(v)$
0	0
1	2
2	3
3	1
4	0
5	1

Figure 55: perfect assignment problem for a graph with six vertices and five edges [94]

6.3 The new Algorithm

Consider the following problem. We will try to describe an algorithm that will capture the movement of a node in an encrypted environment. Each node can move in the three-dimensional space (x, y, z). The main purpose of the project is the node to encryptly send the area in which it is located and the recipient to perceive, to where the node moved depending on the values received.

For a given undirected graph G = (V, E), where |V| = n and |E| = m, find a function g:V > {0,1,... , n-1} such that the function h : E > {0,..., m-1}, defined as

$$h(e = (u,v) \in E) = \big(g(u) + g(v)\big) mod\ m$$ is a bijection.

We are looking for an assignment of values to vertices so that for each edge the sum of values associated with its endpoints taken modulo the number of edges is a unique integer in the range [0,m-1].

If the graph is acyclic, a simple procedure can be used to find values for each vertex, as follows:

- Associate with each edge a unique number h(e) ϵ[0,m-1] in any order.
- For each connected component of G choose a vertex v
- For this vertex set g(v) to 0.
- Traverse the graph using a depth- first search beginning with vertex v.
- If vertex w is reached from vertex u, and the value associated with the edge
- e= (u,w) is h(e), set g(w) to (h(e)- g(u))mod m.
- Apply the above method to each component of G.

We are ready to present the new algorithm for generating a minimal perfect hash function. The mapping step generates a random undirected graph G taking S as input. The ordering step determines the order in which hash values are assigned to keys. The graph is derived to G_{crit} and $G_{ncrit.}$

```
procedure New Algorithm (S,g)

Mapping (S,G);

Ordering (G, Gcrit,  Gncrit,);

Searching ((G, Gcrit,  Gncrit ,g);
```

Figure 56: main steps of the new algorithm

6.3.1 Node movement

We assume that the motion of the node is divided into two steps and includes motion with respect to the axes (x, y) & (x, z), so as to show the motion to the right-left & up-down.

Figure 57: three coordinate axes

6.3.2 The mapping step

The respective movement is also divided into two steps, where each step corresponds to a new field.

Figure 58: Axis (x, y)

Red→ starting point (point of immobility)

Blue→ Possible areas of the first step

Yellow → Possible areas of the second step

We consider that the backward movement presents the least probabilities, as the node will be in motion.

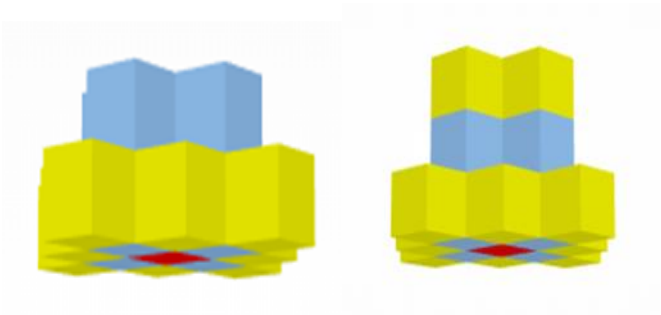


Figure 59: Axis (x, y)

Red→ starting point (point of immobility)

Blue→ Possible areas of the first step

Yellow → Possible areas of the second step

Field of motion

Axis (x, z) Axis (x, z)

First step → 5 fields First step → 3 (top-down-same level)

Second step → 5 fields Second step → 3 (top-down-same level)

Total 5^2 = 25 possible moves Total 3^2 = 9 possible moves

In total we have 25 * 9 = 225 fields

The mapping procedure (S,G) receives as input the set of keys from S and generates a random undirected graph G. To generate the Minimal Perfect Hash Function the number of critical edges in G must be$|E_{crit}| \leq \frac{1}{2} |E|$, where $|E_{crit}| \subseteq E$ be a set of critical edges which contains all edges from E connecting critical vertices. The reason is that the maximum value of h(e) assigned to an edge e E in this case is m-1. The random graph G is generated using two hash functions h_1 and h_2. These functions transform the keys from S to integers in [0, │V│-1], so the set of vertices V has │V│ vertices and each one of them is labeled with a distinct value from [0, │V│-1]. For each key x from S the edge {h_1(x), h_2(x)} is added to E. A self-loop occurs when h_1(x) = h_2(x). We want to avoid self-loops so we try to modify h_2(x) by adding a random number in the range [1, |V |−1]. When a multiple edge occurs we abort and start again a new iteration.

We expect that the number of iterations to obtain G is constant. Let p be the probability of generating a random graph G without self-loops and multiple edges. Let X be a random variable counting the number of iterations to generate G. Variable X is said to have the geometric distribution with P (X = i) = $p(1-p)^{i-1}$. So, the expected number of iterations to generate G is

$N_i(X) = \sum_{j=1}^{\infty} j\, P(X = j) = 1/p$, its variance is V (X) = (1 − p)/p^2.

Let d be the space of edges in G that may be generated by h_1 and h_2. The graphs generated in this step are undirected and the number of possible edges in d is given by $|d| = \binom{|V|}{2}$. The number of possible edges that might become a multiple edge when the j th edge is added to G is j −1, and the incremental construction of G implies that p (|V |) is:

$$p(|V| = \prod_{j=1}^{n} \frac{\binom{|V|}{2} - (j-1)}{\binom{|V|}{2}} = \prod_{j=0}^{n-1} \frac{\binom{|V|}{2} - j}{\binom{|V|}{2}}$$

As │V│= cn the probability is as follow:

$$p(n) = \prod_{j=0}^{n-1} 1 - \left(\frac{2j}{c^2n^2 - cn}\right)$$

Using an asymptotic estimate from Palmer [99], for two functions f_1: $\Re \rightarrow \Re$ and f_2: $\Re \rightarrow \Re$ defined by f_1(k) = 1 − k and f_2(k) = e−k, the inequality

$f_1(k) \leq f_2(k)$ is true $\forall\ k \in \Re$. Considering $k = \frac{2j}{c^2n^2-cn}$ we have

$$p(n) \leq \prod_{j=0}^{n-1} e^{-\left(\frac{2j}{c^2n^2-cn}\right)} = e^{-\left(\frac{n-1}{c^2n-c}\right)}$$

Thus, $\lim_{n\to\infty} p(n) \cong e^{\frac{1}{c^2}}$

As $N_i(X) = 1/p$ then $N_i(X) \simeq e^{\frac{1}{c^2}}$. After that, we empirically determine the c value to obtain a random graph G with $|E_{crit}| \leq \frac{1}{2}|E|$ the probability P|Ecrit| that |Ecrit| ≤ |E|, |E| = n, tends to 0 when c < 1.15 and n increases. However, it tends to 1 when c ≥ 1.15 and n increases. Thus, |V | = 1.15n is considered a threshold for generating a random graph G where |Ecrit| ≤ |E| with probability tending to 1 when n increases. Therefore, we use c = 1.15 in the new algorithm. The MPHF generated by the new algorithm needs 1.15n integer numbers to be stored, since |V | = 1.15n. Thus, the generated function is stored in 55% —1.15n/2.09n — of the space necessary to store the one generated by the CHM algorithm. As P|Ecrit| tends to 1 when n increases, we consider that the expected number of iterations to generate G is $N_i(X) \simeq e^{\frac{1}{c^2}}$. For c = 1.15, Ni(X) ≃ 2.13 on average, which is constant. So, the mapping step takes O(n) time. The rationale is that P|Ecrit| tends to 1 when n increases. However, if some addition g(u)+g(w) is greater than m in the searching step for {u, w} ∈ E then the mapping step is restarted.

6.3.3 Maximal value Assigned to An edge

For a random graph G with |Ecrit| = 0.5n and |V | = 1.15n, it is always possible to generate a MPHF because the maximal value A_{max} assigned to an edge e ∈ Ecrit is at most m − 1 (A_{max} corresponds to the maximal value generated by the assignment of values to critical vertices)

Theorem 1The number of back edges Nb edges of a random graph $G = G_{crit} \cup G_{ncrit}$ is given by: N_{bedges} = |Ecrit| − |Vcrit| + 1.

Theorem 2 The maximal value Amax assigned to an edge e ∈ Ecrit in the assignment of values to critical vertices is: Amax ≤ 2|Vcrit| − 3 + 2Nt.

6.4 Technical summaries for Privacy- preserving proximity- based security systems for location based services

6.4.1 A proximity- based authentication key generation strategy, without involving any trusted authority

We describe a proximity- based authentication key generation strategy, without involving any trusted authority, pre-shared secret or public key infrastructure. We assume that a radio client called Alice initiates the authentication and pairwise session key generation with clients I her proximity. A peer client called Bob responds to her request. Both clients monitor their ambient radio signals at the frequency band during the time specified by Alice. Bob informs Alice his public location tag, which incorporates the RSSIs, sequence numbers and media access control (MAC) addresses of the

packets. Bob builds and keep secret location tag, which consists of the packet arrival time sequence. Based on Bob's public location tag and her own measurements, Alice identifies their shared packets and uses their features to derive the proximity evidence of Bob for both authentication and session key generation. Meanwhile, Alice informs Bob the indices of their shared packets in his secret location tag and helps him to generate his copy of the session key [100].

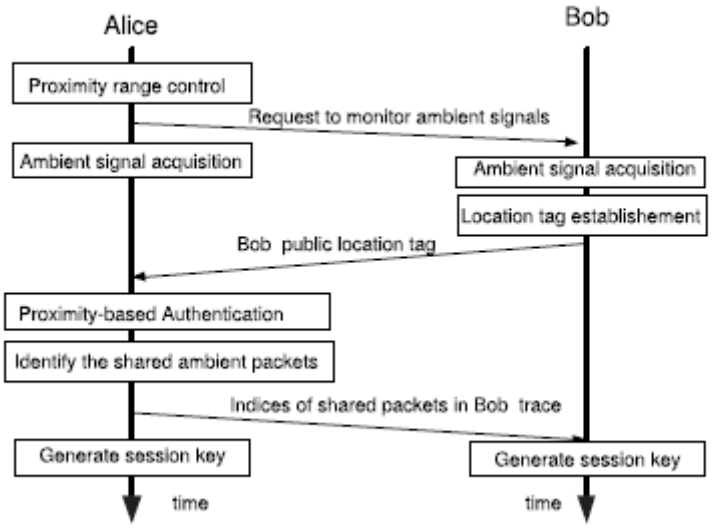


Figure 60: Flowchart of the proximity- based security system based on ambient radio signals

By integrating the authentication and key generation process, we build a proximity-based security protocol for mobile users in wireless networks.

As illustrated in figure 60 this protocol consists of the following steps:

1. Alice decides and broadcasts her proximity test policy.
2. Upon receiving Alice's request, Bob measures the features of the packets as Alice specified. Both clients extract and store the RSSIs, arrival time, MAC addresses and sequence number of their ambient packets.
3. Bob builds a location tag, sends Alice his public location tag, and keeps his secret location tag.
4. Alice authenticates Bob.
5. Alice compares Bob's public location tag with her trace to identify their shared packets. Following a key generation Algorithm, Alice builds a session key, K_A, and informs Bob the indices of their shared packets in his trace J.
6. Based on his secret location tag and the indices J, Bob generates his session key, K_B.

In the above handshake process, error connection coding can be applied to counteract the transmission errors due to channel fading and interference. In addition, because of the different ambient ratio environments and packet loss rates, clients usually take different time to obtain a given number of ambient packets. Due to this problem, the proposed key generation strategy relies on the same shared packets between Bob and Alice and thus provides a certain degree of robustness against packet loss.

6.4.2 A dynamic privacy- preserving key management scheme

We describe a proposed scheme in which, we first introduce a privacy- preserving authentication technique that not only provides the user's anonymous authentication but enables double-registration detection as well.

The location based services session key update procedures. Firstly the session of an LBS is divided into several time slots so that each time slot holds a different session key. Secondly, we integrate a novel dynamic threshold technique in traditional vehicle to vehicle and vehicle to infrastructure communications to achieve the session key's backward secrecy.

Performance evaluations via extensive simulations demonstrate the efficiency and effectiveness of the proposed scheme, in terms of low key update delay and fast key update ratio.

We consider a typical location- based service in vehicular ad hoc networks (VANET) which comprises an SP, some deployed RSUs affiliated to the SP, and a large number of vehicle users U' = { U_1, U_2,…} moving around the area, as shown in Figure 61.

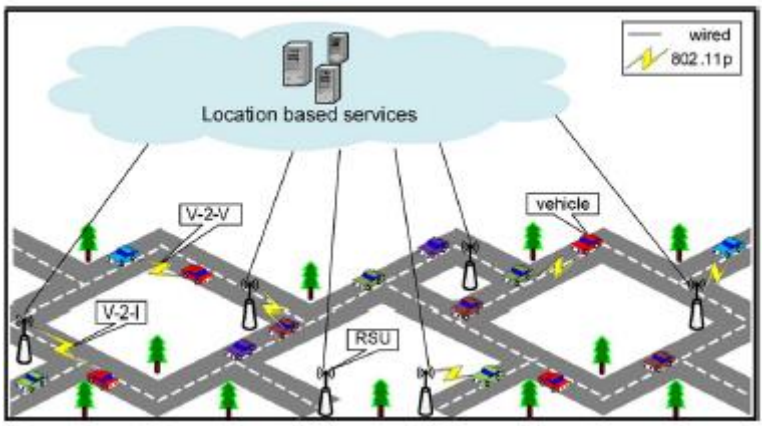


Figure 61: Network architecture for LBSs in VANETs.

The SP in the area can provide various services, such as to provide some local traffic information or establish a virtual on- road community. Because the vehicle users move along the road, the SP cannot directly reach the vehicles. Therefore, after being connected with the SP by wire links or any other links with high bandwidth and low delay, the affiliated RSUs can help the SP to broadcast and/or relay messages to vehicle users via vehicular communications. The stationary RSUs are usually located at the road side and perform two main functions broadcasting and relaying. The broadcasting component is responsible for broadcasting service contents that originated from the SP to the vehicle users on the road, where the service contents can either directly reach the passing-by vehicles or reach other vehicles in a manner. The relaying component helps vehicle users with forwarding some requests to reduce the burdens at the SP. RSU is trustable and usually equipped with not only high- storage capacity but strong computational capability as well, which causes its high cost. Due to the high cost, it is impractical to erect RSUs to cover the whole area, particularly at the early deployment of LBSs in VANETs. In this network model, only a small number of RSUs are deployed at some spots [101].

Each vehicle U_i ϵ U' is equipped with an on board unit device, which allows it to communicate with other vehicles for sharing some information of common interest or communicate with the RSUs for accessing the LBSs and receiving service contents relayed by the RSUs. The OBU device in VANET has no power constrained issue and is equipped with powerful computational and communication capabilities [101].

6.5 Appendices (Description of Algorithm)

6.5.1 The Mapping Phase

Step Description of Algorithm

1. build random table for h_0, h_1 and h_2
2. for each v in (0 ...2r - 1] do vertex[v].firstedge = 0; vertex[v].degree = 0
3. for each i in [1...n] do

edge[i].ho = h_0 (k_i); edge(i).h_1 = h_1(k_i); edge[i].h_2 = h2(k_i)

edge[i].nextedge$_1$ = 0

add edge[i] to linked list with header vertex [h_1(k_i)].firstedge;

increment vertex[h 1(k_i)].degree

add edge[i) to linked list with header vertex(h_2(k_i)).firstedge;

increment vertex(h_2(k_i)).degree

4. for each v in [O ...r - 1) do

check that all edges in linked list vertex[v].firstedge

have distinct (h_0, h_1, h_2) triples.

5. if triples are not distinct then repeat from step(1).

6.5.2 The ordering phase

Step Description of Algorithm

```
1.  CId = 0 /* assign all vertices an ID 0. */
     for v in [0 ...2r - 1) do assign CId to v
    CId=  1
    for v in [0 ...2r - 1) do /*assign unique nonzero IDs to CCY s and AC s. */
        if v has nonzero degree and its component ID equals 0 then
            initialize(VSTACK)  /* process one component.  */
            push( v, VSTACK) /* save the first vertex of the component. */
           do
```

```
                v = pop(VSTACK) /* get an unassigned vertex from VSTACK. */
                assign CId to v /* assign the ID. */
                for each w adjacent to v do
                        /* if there are vertices unassigned, put them into VSTACK.  */
                    if component  ID of w is zero and not in VSTACK  then
                            push(w, VSTACK)
                  while VSTACK is not empty
             CId = CId +1 /* increase ID for next component.  */
```

2. initialize(VSTACK) /* get all one-degree vertices into VSTACK. * / .

```
for each nonzero degree v in [0 ...2r - 1) do
      if vertex[v].degree = 1 then
   push(v, VSTACK)
  decrement vertex[ v].degree
```

3. while VSTACK is not empty do /* visit and truncate all edges in Ecp. */ .

```
      v = pop(VSTACK)
      for each w adjacent to v do
          if degree of w > 0 then decrease vertex[w].degree
          if vertex[w].degree = 1 then push(w, VSTACK)
```

4. make all vertices not SELECTED /* obtain a VScc for all Vcc vertices. */ .

```
 i = 1;
for all nonzero degree and not  SELECTED v in  [0 ...2r - 1] do
     select vi = a vertex of maximum  degree > 0
     initialize(VHEAP);  insert  (v;, VHEAP)
     do
     vi = deletemax(VHEAP)
    mark vi SELECTED and put vi into VS
    for each w adjacent to vi do
          if w is not SELECTED and w is not in VHEAP then
              insert(w,  VHEAP)
         i=i+l
    while VHEAP is not empty
```

5. for i = 1 to t do /* assign indirection bit to all vertices in Vcc */ .

```
Let s= IK(vi)I and Wj be any MARKED vertex adjacent to vi
Let t be the number of not MARKED vertices adjacent to vi
If s = 0 then vertex[vi].bit  = 1
```

If s= 1 then
if vertex[w_j].bit = 0 then vertex[v_1].bit = 1
else vertex[v_1].bit = 0
If s > 1 then
if i= 0 and vertex[w_j].bit = 0 for all W_j then vertex[v_i].bit = 0
else
for all W_j do
if vertex[w_j] = 0 then vertex[w_j].bit = 1
vertex[v_i].bit = 1

6.5.3 The searching phase

Step Description of Algorithm

1. R = {}, S = {} /* S is the set of component IDs of those occupied trees. */
/* R records the root vertices of trees in S. */
/*Both sets are empty at first. */

for i = 1 to t do /* assign g values to V_{cc}s to have edges in Ecc indirectly hashed. */
mark v_i ASSIGNED /* select the next vertex in V Scc for g value assignment. */
establish a random probe sequence s_0, s_1, ..., S_{n-1} for [0 ...n -1]
/* prepare the order in which different g values will be tried. * /
j=0
do
Let W be the set of ASSIGNED vertices adjacent to v_i
Collision = false
if IK(v_i)I=0 then /* V_i is the first vertex of an un-assigned component. * /
vertex[v_i].g = s_{ji} /*assign v_i's g entry the value s_j. */
else
if IK(v_i)I = 1 AND vertex[v_i].mark ≠ vertex[w].mark then
/* if only one edge in the level and it is a direct edge, then assign the g value */
/* to vertex V_i such that h_{final} of the edge can be computed directly. */
let w be in W and k in K(v_i)
vertex[v_i].g = [edge[k].final - edge[k].h_0 - vertex[w].g] mod n
/* assign g value when k is direct */
if edge[k].final ≥ a then vertex[v_i].g = edge[k].final - a
else vertex[v_i].g = n - a + edge[k].final
else /* all the edges in the level have to be indirect. Need to find * /

```
        /* unoccupied zero-degree vertices or trees.  * /
if vi in [0 ...r - 1] then /* distinguish which side vi is on * /
      for each k in K(vi) do/* vi is on h1 side.  */
          h(k) = edge[k].ho + vertex[edge[k].h2] + (sj mod 2r)
                /* obtain the location of indirect-to vertex.  */
       if vertex[h(k)] is occupied OR vertex[h(k)).Cld  in S then
           collision = true /* the indirect-to vertex is occupied. */
else /* the vi is on h2 side.  */
        for each k in K(vi) do
h(k) = edge[k].h0 + vertex[edge[k].h1] + (sj mod 2r)
      if vertex[h(k)] is occupied OR vertex[h(k)].Cld  in S then
           collision = true
if not  collision  then
                /* if all indirect-to locations are not occupied, */
                /* set all of them occupied.  */
for each kin K(vi) do
       if vertex[h(k)] is a zero-degree vertex then
                set vertex[h(k)] occupied
      else
                S = S UNION {vertex[h(k)].CId}
                R =R UNION {vertex[h(k)]}
           vertex[ h(k)].g = edge[k].final  /"' set the g value of for indirect key  */
       i=i+1
   else /* if this sj  causes any collisions, try next one.  */
     j=j+1
    if j > n+1
        fail
While collision
    2.      Initialize (VSTACK)  /* process EAC  */ .
            for i = 0 to n -1 do
                if Vi is both cycle and tree vertex then
                                        /* identify starting vertices.  */
                    for all w not  ASSIGNED  in step 1 and  adjacent  to Vi  do
                        push (w, VSTACK)
               while VSTACK is not empty do v = pop(VSTACK)
```

/* directly hash all tree edges. */

mark v ASSIGNED

for w ASSIGNED and adjacent to v do

let k join v and w

vertex[v_i].g = [edge[k].final - edge[k].ho - vertex[w].g] mod n

for all w not ASSIGNED and adjacent to v and not in VSTACK do

push(w, VSTACK)

3. Repeat (2) for all vertices in R. Each vertex in R will act as v_i in (2).

4. repeat (2) for arbitrary root vertices in ACs that have not accepted any indirect edges. Each such vertex will act as V_i in (2)

7 CONCLUSION

In this thesis, we have shown how cryptographic hash functions gained its importance in the field of cryptography. We have made all the attempts to give a complete picture of cryptographic hashes, its design techniques and vulnerabilities. We discussed the most popular hash functions security properties and notions and showed how these requirements have influenced the design of hash functions.

In the second part of this thesis we provided a thorough discussion of the state of art of hash functions designs.

Finally, an algorithm for finding order preserving minimal perfect hash functions is described. The method is able to find OPMPHF for various sizes of key sets. Several probabilistic analysis results on the characteristics of the random graph G are given. They are useful in guiding a proper selection of various parameters and providing insights on the design of the three main steps of the algorithm.

More experiments with the algorithm are planned. One direction is the dynamic hashing. Other possible interests are concerned with the conjecture.

8 FUTURE WORK

Hashing algorithms can be pretty useful. However, IT is a really fast-changing industry and this entropy also extends to hashing algorithms. Hash algorithms through which devices in the IoT can securely send messages between them, can be proved very useful in the future, because the Internet of Things (IoT) promises to be the next big revolution of the World Wide Web. In order to ensure integrity, hash algorithm is used. It has a very wide range of applications, ranging from smart cities, smart homes, and a lot more. When nodes in wireless sensor networks are monitored through internet it becomes a part of Internet of Things. This brings in a lot of concerns related to security, privacy and standardization.

LIST OF NOTATIONS

List of Abbreviations

PKC Public Key Cryptography
AES Advanced Encryption Standard
DES Data Encryption Standard
DSA Digital Signature Algorithm
ECDSA Elliptic Curve Digital Signature Algorithm
ISO International Organization for Standardization
MAC Message Authentication Code
HMAC Hashing Message Authentication Codes
MDx Message Digest x
NESSIE New European Schemes for Signatures, Integrity and Encryption
NIST National Institute of Standards and Technology
RIPEMD RIPE Message Digest
RSA Rivest-Shamir-Adleman
SHA-1 Secure Hash Algorithm 1
SHA-2 Secure Hash Algorithm 2
SIMD Single instruction multiple - data
TIMESEC Digital Timestamping and the Evaluation of Security
TLS Transport Layer Security
SSL Secure Sockets Layer
TCP Transmission Control Protocol
PGP Pretty Good Privacy
RC4 Rivest Cipher 4
MOS Mapping- Ordering- Searching
ECC Elliptic Curve Cryptography
CAN Content Addressable Network
CRHF Collision Resistant Hash Function
OWHF One Way Hah Function
MDC Message Detection Code
NSA National Security Agency
HAIFA Hash Iterative Framework
CR Collision Resistance
Pre Pre- image Resistance

Sec 2nd Pre- image Resistance
PHF Perfect Hash Function
MPHF Minimal Perfect Hash Function
OPMPHF Order Preserving Minimal Perfect Hash Function
OPE Order Preserving encryption
ROPF Random Order Preserving Hash Function

List of Mathematical Symbols

f compression function
g output transformation
h hash function or MAC algorithm
E encryption algorithm
F round function of a block cipher
X input to a function
Y output from a function
K key for MAC or encryption algorithm
M message
P plaintext
C ciphertext
X_i input block
M_i message block
H_i chaining variable
IV initial value
W_j message word (32-bit or 64-bit)
n output length (of a hash function or MAC algorithm)
b block length (of a compression function or block cipher)
c chaining variable length
k key length
D domain of a function
R range of a function
K key space
M message space
|S| number of elements of the set S
O order
∈ Element of. . .
∞ infinity
exp exponential function
max maximum of. . .
mod modulo (remainder of integer division)
[Z] the smallest integer larger than or equal to Z

REFERENCES

[1] C. Paar and J. Pelzl, *Understanding Cryptography*. Springer Berlin Heidelberg, 2010.

[2] W. Stallings, *Cryptography and Network Security: Principles and Practices*. 2005.

[3] C. Paar and J. Pelzl, *Understanding Cryptography*. 2010.

[4] B. Guttman and E. A. Roback, “NIST SP800-12 - An Introduction to Computer Security: The NIST Handbook (1995),” 1995.

[5] J. F. Kurose and K. W. Ross, *COMPUTER NETWORKING A Top-Down Approach*. .

[6] W. Diffie, W. Diffie, and M. E. Hellman, “New Directions in Cryptography,” *IEEE Trans. Inf. Theory*, 1976, doi: 10.1109/TIT.1976.1055638.

[7] K. H. Rosen, *CryptographyTheoryandpractice(3ed).pdf*. 2006.

[8] “No Title.” https://searchsecurity.techtarget.com/definition/asymmetric-cryptography.

[9] H. Tiwari and K. Asawa, “Cryptographic hash function: An elevated view,” *Eur. J. Sci. Res.*, 2010.

[10] “Recent Contribution to Cryptographic Hash Functions.,” [Online]. Available: https://www.drdobbs.com/security/recent-contributions-to-cryptographic-ha/219500573.

[11] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby, “Collisions of SHA-0 and Reduced SHA-1.”

[12] X. Wang, Y. L. Yin, and H. Yu, “Finding collisions in the full SHA-1,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3621 LNCS, no. 90304009, pp. 17–36, 2006, doi: 10.1007/11535218_2.

[13] X. Wang, D. Feng, X. Lai, and H. Yu, “Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD,” vol. 5, pp. 5–8, 2004.

[14] “NIST.” https://www.federalregister.gov/documents/2007/11/02/E7-21581/announcing-request-for-candidate-algorithm-nominations-for-a-new-cryptographic-hash-algorithm-sha-3.

[15] “Hash Function Competition.” https://en.wikipedia.org/wiki/NIST_hash_function_competition, last accessed 30/5/2019.

[16] B. Preneel, “The first 30 years of cryptographic hash functions and the NIST SHA-3 competition,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5985 LNCS, pp. 1–14, 2010, doi: 10.1007/978-3-642-11925-5_1.

[17] “Recent Contribution to Cryptographic Hash Functions.”

[18] T. Dierks and C. Allen, “RFC 2246: The TLS Protocol,” *Network Working Group*. 1999.

[19] G. Kim and E. Spafford, “Experiences with tripwire: Using integrity checkers for intrusion detection,” no. March 1995, 1994, [Online]. Available: http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=2114&context=cstech.

[20] G. H. Kim and E. H. Spafford, “Writing, Supporting, and Evaluating Tripwire: A Publically Available Security Tool,” *Proc. USENIX Unix Appl. Dev. Symp.*, pp. 89–107, 1994.

[21] S. Notions, S. Al-kuwari, J. H. Davenport, and R. J. Bradford, “Cryptographic Hash Functions : Recent Design Trends and,” *Eprint.Iacr.Org*, 2011.

[22] M. Bellare and T. Kohno, “Hash function balance and its impact on birthday attacks,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 2004.

[23] R. C. Merkle, “A certified digital signature,” pp. 218–238, 1990.

[24] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. 1996.

[25] S. Goldwasser and M. Bellare, “Lecture notes on cryptography,” *… “Cryptography Comput. Secur. …*, no. July, pp. 1–289, 2008, [Online]. Available: http://sreekavitha.in/Cryptography/gb.pdf.

[26] M. das G. Rua, “The MD5 Message Digest Algorithm,” *Japanese Soc. Biofeedback Res.*, vol. 19, pp. 709–715, 1992, doi: 10.20595/jjbf.19.0_3.

[27] “ARCHIVED PUBLICATION.” [Online]. Available: http://csrc.nist.gov/publications/PubsFIPS.html#fips180-4.

[28] J.-P. A. M. C.-W. P. Henzen, “The Hash Function BLAKE.”

[29] M. Bellare and T. Ristenpart, “Hash functions in the dedicated-key setting: Design choices and MPP transforms,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4596 LNCS, pp. 399–410, 2007, doi: 10.1007/978-3-540-73420-8_36.

[30] R. Rivest, “Abelian square-free dithering for iterated hash functions,” *ECrypt Hash Funct. Work. June*, 2005, [Online]. Available: http://people.csail.mit.edu/rivest/Rivest-AbelianSquareFreeDitheringForIteratedHashFunctions.pdf.

[31] E. Andreeva *et al.*, “Second preimage attacks on dithered hash functions,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4965 LNCS, pp. 270–288, 2008, doi: 10.1007/978-3-540-78967-3_16.

[32] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “Sponge Functions,” 2014. [Online]. Available: http://keyak.noekeon.org/.

[33] S. Lucks, “Design Principles for Iterated Hash Functions,” *E-Print*, no. September, pp. 1–22, 2004, [Online]. Available: http://eprint.iacr.org/2004/253.

[34] I. Dinur, O. Dunkelman, and A. Shamir, “New attacks on Keccak-224 and Keccak-256,” 2012, doi: 10.1007/978-3-642-34047-5_25.

[35] I. Dinur, O. Dunkelman, and A. Shamir, “Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials,” 2014, doi: 10.1007/978-3-662-43933-3_12.

[36] D. C. Kim, D. Hong, J. K. Lee, W. H. Kim, and D. Kwon, “LSH: A new fast secure hash function family,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, vol. 8949, pp. 286–313, doi: 10.1007/978-

3-319-15943-0_18.

[37] A. Joux, “Multicollisions in iterated hash functions. Application to cascaded constructions,” vol. 1, no. 1, pp. 6–8, 2003, doi: 10.16309/j.cnki.issn.1007-1776.2003.03.004.

[38] M. Nandi and S. Paul, “Speeding up the wide-pipe: Secure and fast hashing,” 2010, doi: 10.1007/978-3-642-17401-8_12.

[39] N. Ferguson *et al.*, “The Skein Hash Function Family.”

[40] R. L. Rivest *et al.*, “The MD6 hash function: A proposal to NIST for SHA-3,” *Preprint*, 2008.

[41] P. S. L. M. Barreto, V. Rijmen, and S. T. S. A, “The W HIRLPOOL Hashing Function,” no. January 2003, pp. 1–20, 2014.

[42] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, “Cryptanalytic attacks on pseudorandom number generators,” 1998, doi: 10.1007/3-540-69710-1_12.

[43] “Hash Function JH; designed by Hongjun Wu.” https://www3.ntu.edu.sg/home/wuhj/research/jh/ (accessed Aug. 01, 2020).

[44] J.-P. Aumasson, L. Henzen, W. Meier, and R. C. W. Phan, “SHA-3 proposal BLAKE,” *SHA3 Compet.*, pp. 1–79, 2010, [Online]. Available: http://www.131002.net/blake.

[45] M. Maqableh, a Samsudin, and M. Alia, “New Hash function based on chaos theory (CHA-1),” *Int. J. …*, vol. 8, no. 2, pp. 20–26, 2008, [Online]. Available: http://paper.ijcsns.org/07_book/200802/20080203.pdf.

[46] J. Kelsey, “The New SHA3 Hash Functions.”

[47] J. Kelsey, S. Chang, and R. Perlner, “SHA-3 Derived Functions : cSHAKE, KMAC, TupleHash and ParallelHash,” *NIST Spec. Publ.*, vol. 800, p. 185, 2016, doi: 10.6028/NIST.SP.800-185.

[48] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, “CAESAR submission: Ketje v1,” 2014. [Online]. Available: http://ketje.noekeon.org/.

[49] R. Gost, “Algebraic Aspects of the Russian Hash Standard,” pp. 1–18, 2012.

[50] J. Guo, J. Jean, G. Leurent, T. Peyrin, and L. Wang, “The usage of counter revisited: Second-preimage attack on new Russian standardized hash function,” 2014, doi: 10.1007/978-3-319-13051-4_12.

[51] Z. Wang, H. Yu, and X. Wang, “Cryptanalysis of GOST R hash function,” *Inf. Process. Lett.*, 2014, doi: 10.1016/j.ipl.2014.07.007.

[52] B. Ma, B. Li, R. Hao, and X. Li, “Improved cryptanalysis on reduced-round GOST and Whirlpool hash function,” 2014, doi: 10.1007/978-3-319-07536-5_18.

[53] “Keccak Team.” https://keccak.team/index.html (accessed Aug. 01, 2020).

[54] D. Aspinall, “Outline Cryptography V : Digital Signatures Handwritten versus Digital Signatures Signature mechanism Digital signatures with a TTP Digital signatures from PK encryption Attacks

on signature schemes [HAC] Existential forgery Signatures with redundancy S.”

[55] A. Kirkby, “Cryptography And E-Commerce: A Wiley Tech Brief,” *Netw. Secur.*, vol. 2001, no. 4, p. 9, Apr. 2001, doi: 10.1016/s1353-4858(01)00416-0.

[56] S. Karforma and S. Banerjee, “OBJECT ORIENTED MODELING OF ELGAMAL DIGITAL SIGNATURE FOR AUTHENTICATION OF STUDY MATERIAL IN E-LEARNING SYSTEM,” vol. 8354, no. 4, 2015.

[57] F. Publication, “Archived Publication,” vol. 3, no. June 2009, 2013, [Online]. Available: http://csrc.nist.gov/publications/PubsFIPS.html#fips180-4.

[58] J. Tom, B. K. Alese, A. Thompson, P. Nlerum, and B. State, “Performance and Security of Group Signature in Wireless Networks,” vol. 4523, no. May, pp. 82–98, 2018.

[59] . K. R., “Elliptic Curve ElGamal Encryption and Signature Schemes,” *Inf. Technol. J.*, 2005, doi: 10.3923/itj.2005.299.306.

[60] C. November, “Lecture 12 Motivation : Domain Extension of MACs Collision-Resistant Hash Functions Extending the Domain of CRHF,” pp. 1–18, 2008.

[61] C. Crypto, L. Notes, and C. S. Vol, “Mihir Bellare Phillip Rogaway,” *Lect. Notes Comput. Sci.*, vol. 1294, pp. 1–32, 1997.

[62] A. Bala and I. Chana, “Fault Tolerance-Challenges, Techniques and Implementation in Cloud Computing,” *Int. J. Comput. Sci. Issues*, vol. 9, no. 1, pp. 288–293, 2012.

[63] I. B. Damgård, “A design principle for hash functions,” 1990, doi: 10.1007/0-387-34805-0_39.

[64] P. Pal and P. Sarkar, “PARSHA-256 - A new parallelizable hash function and a multithreaded implementation,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 2003, doi: 10.1007/978-3-540-39887-5_25.

[65] M. Naor and M. Yung, “Universal one-way hash functions and their cryptographic applications,” 1989, doi: 10.1145/73007.73011.

[66] M. Bellare and M. Daniele, “A New Paradigm for Collision- free Hashing: Incrementality at Reduced Cost,” pp. 285–299, 1996.

[67] A. Joux, “Multicollisions in iterated hash functions. application to cascaded constructions,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 2004, doi: 10.1007/978-3-540-28628-8_19.

[68] J. Kelsey and T. Kohno, “Herding hash functions and the nostradamus attack,” 2006, doi: 10.1007/11761679_12.

[69] G. Leurent, T. Peyrin, and L. Wang, “New generic attacks against hash-based MACs,” 2013, doi: 10.1007/978-3-642-42045-0_1.

[70] I. Dinur and G. Leurent, “Improved generic attacks against hash-based MACs and HAIFA,” 2014, doi: 10.1007/978-3-662-44371-2_9.

[71] G. Bertoni, “Keccak sponge function family main document,” *Nist*, no. January 2009, pp. 1–121, 2009.

[72] T. Peyrin, Y. Sasaki, and L. Wang, “Generic related-key attacks for HMAC,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7658 LNCS, pp. 580–597, 2012, doi: 10.1007/978-3-642-34961-4_35.

[73] B. den Boer and A. Bosselaers, “Collisions for the compression function of MD5,” 1994, doi: 10.1007/3-540-48285-7_26.

[74] I. Mironov, “Hash functions : Theory, attacks, and applications Theory of hash functions,” *Microsoft Res. Silicon Val. Campus*, 2005.

[75] X. Wang and H. Yu, “How to break MD5 and other hash functions,” 2005, doi: 10.1007/11426639_2.

[76] T. Xie, F. Liu, and D. Feng, “Fast collision attack on MD5,” *IACR ePrint Arch. Rep.*, 2006, doi: 10.1.1.301.4421.

[77] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, “The first collision for full SHA-1.” [Online]. Available: https://shattered.io.

[78] M. Lamberger and F. Mendel, “Higher-Order Differential Attack on Reduced SHA-256,” *IACR Cryptol. ePrint Arch.*, 2011.

[79] “Difficulty - Bitcoin Wiki.” https://en.bitcoin.it/wiki/Difficulty (accessed Aug. 01, 2020).

[80] J. Zhong and X. Lai, “Improved Preimage Attack on One-block MD4,” vol. 2.

[81] “Lessons From The History Of Attacks On Secure Hash Functions.” https://electriccoin.co/blog/lessons-from-the-history-of-attacks-on-secure-hash-functions/.

[82] “RadioGatún.” http://radiogatun.noekeon.org/ (accessed Aug. 01, 2020).

[83] N. Ferguson, “The Skein Hash Function Family,” *Argument*, vol. 30, no. 4, p. 79, 2010, [Online]. Available: http://www.schneier.com/skein.html.

[84] P. Gauravaram *et al.*, “Grøstl – a SHA-3 candidate,” vol. 0, no. 2, pp. 1–42, 2011.

[85] H. Dobbertin, A. Bosselaers, and B. Preneel, “RIPEMD-160: A strengthened version of RIPEMD,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1996, vol. 1039, pp. 71–82, doi: 10.1007/3-540-60865-6_44.

[86] R. Anderson and E. Biham, “Tiger: A fast new hash function,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1996, vol. 1039, pp. 89–97, doi: 10.1007/3-540-60865-6_46.

[87] J. Daemen and C. Clapp, “Fast hashing and stream encryption with PANAMA,” in *Lecture Notes in Computer Science (including* subseries *Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1998, vol. 1372, pp. 60–74, doi: 10.1007/3-540-69710-1_5.

[88] F. Mendel, N. Pramstaller, C. Rechberger, M. Kontak, and J. Szmidt, “Cryptanalysis of the GOST Hash Function,” doi: 10.1007/978-3-540-85174-5.

[89] E. A. Fox, Q. fan chen, and L. S. Heath, “Faster algorithm for constructing minimal perfect hash functions,” *SIGIR Forum (ACM Spec. Interes. Gr. Inf. Retrieval)*, pp. 266–273, 1992, doi: 10.1145/133160.133209.

[90] A. Boldyreva, N. Chenette, and A. O’Neill, “Order-preserving encryption revisited: Improved security analysis and alternative solutions,” 2011, doi: 10.1007/978-3-642-22792-9_33.

[91] R. J. Cichelli, “Minimal Perfect Hash Functions Made Simple,” *Commun. ACM*, vol. 23, no. 1, pp. 17–19, 1980, doi: 10.1145/358808.358813.

[92] M. Antoine and F. Huet, “Multiple order-preserving hash functions for load balancing in P2P networks,” 2018, doi: 10.1504/IJCNDS.2018.088501.

[93] E. A. Fox, Q. F. Chen, A. M. Daoud, and L. S. Heath, “Order preserving minimal perfect hash functions and information retrieval,” 1989, doi: 10.1145/96749.98233.

[94] F. C. Botelho, D. Menoti, and N. Ziviani, “A new algorithm for constructing minimal perfect hash functions,” no. May, 2004.

[95] Z. J. Czech, G. Havas, and B. S. Majewski, “Perfect hashing,” *Theor. Comput. Sci.*, vol. 182, no. 1–2, pp. 1–143, 1997, doi: 10.1016/S0304-3975(96)00146-6.

[96] B. S. Majewski, N. C. Wormald, G. Havas, and Z. J. Czech, “A family of perfect hashing methods,” *Comput. J.*, vol. 39, no. 6, 1996.

[97] B. Bollobas, *Random Graphs*. London, Orlando, San Diego, NewYork, Toronto, Montreal, Sydney, Tokyo: Academic Press, Inc., 1985.

[98] G. Havas, “Graphs , Hypergraphs and Hashing,” no. March, 2016, doi: 10.1007/3-540-57899-4.

[99] Palmer E.M., *Graphical Evolution*. New York: John Wiley & Sons, 185AD.

[100] L. Xiao, Q. Yan, W. Lou, G. Chen, and Y. T. Hou, “Proximity-based security techniques for mobile users in wireless networks,” *IEEE Trans. Inf. Forensics Secur.*, vol. 8, no. 12, pp. 2089–2100, 2013, doi: 10.1109/TIFS.2013.2286269.

[101] R. Lu, X. Lin, X. Liang, S. Member, and X. S. Shen, “A Dynamic Privacy-Preserving Key Management Scheme for Location-Based Services in VANETs,” *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 1, pp. 127–139, 2012, doi: 10.1109/TITS.2011.2164068.