SARMAL: A CRYPTOGRAPHIC HASH FUNCTION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

KEREM VARICI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
THE DEPARTMENT OF CRYPTOGRAPHY

AUGUST 2008

Approval of the Graduate School of Applied Mathematics

Prof. Dr. Ersan AKYILDIZ
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ferruh ÖZBUDAK
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ali DOĞANAKSOY
Supervisor

Examining Committee Members

Prof. Dr. Hurşit ÖNSIPER

Assoc. Prof. Dr. Ali DOĞANAKSOY

Assoc. Prof. Dr. Emrah Çakçak

Dr. Enis UNGAN

Dr. Meltem Sönmez TURAN

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: KEREM VARICI

Signature :

ABSTRACT

SARMAL: A CRYPTOGRAPHIC HASH FUNCTION

VARICI, Kerem
M.S., Department of THE DEPARTMENT OF CRYPTOGRAPHY
Supervisor : Assoc. Prof. Dr. Ali Doğanaksoy

AUGUST 2008, 49 pages

Recent years witnessed the continuous works on analysis of cryptographic hash functions which reveal that most of them are not as secure as claimed. Wang et al. presented the first full round collisions on $MD4$ and $RIPEMD$ using a new attack technique on hash functions which is based on differential cryptanalysis. Then, this attack is further developed and used in the analysis of other famous and widely used hash functions. As a result of these studies, National Institute of Standards and Technology (NIST) announced a public competition of designing a new hash function which will be chosen as the new hash function standard (Secure Hash Algorithm 3, ($SHA-3$)).

It is expected from new algorithm to provide security bounds for preimage, second-preimage and collision attacks, besides being resistant against all known attack methods. The new hash standard is expected to support variable hash sizes to be used for variable purposes. Moreover, the design should be efficient in both software and hardware implementations.

In this thesis, we present a new cryptographic hash function family, Sarmal, which is designed to satisfy all the properties above as a candidate for the $SHA-3$ competition. It uses the well known components from block cipher theory to satisfy both security/efficiency trade-off. On the other hand, HAIFA iterative hashing mode is used to prevent latest weaknesses

of standard Merkle-Damgård paradigm and provide flexible hash size. Moreover, software implementations reveal that Sarmal can be very efficient on multiple platforms.

Keywords: Sarmal, Design, Hash Function

ÖZ

SARMAL: KRIPTOGRAFIK ÖZET FONKSIYONU TASARIMI

VARICI, Kerem
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi : Assoc. Prof. Dr. Ali Doğanaksoy

Ağustos 2008, 49 sayfa

Son yıllarda kriptografik özet fonksiyonu analizinde süregelen çalışmalar, bir çoğunun belirtildiği kadar güvenli olmadığını göstermiştir. Wang vd. özet fonksiyonları için diferansiyel kriptanalize dayanan yeni bir atak tekniği kullanarak $MD4$ ve $RIPEMD$ fonksiyonlarina, tüm çevirimi kapsayan çakışmalar buldular. Daha sonra bu atak geliştirilerek herkes tarafından bilinen ve çoğu alanda kullanılan diğer özet fonksiyonlarının analizinde kullanıldı. Yapılan bu çalışmaların sonucunda "National Institute of Standards and Technology" (NIST), yeni özet fonksiyon standardı $SHA-3$ seçilmek üzere, herkesin katılımına açık bir tasarım yarışması başlattı.

Yeni algoritmanın ters görüntü kümesi, ikincil ters görüntü kümesi ve çakışma atakları için gerekli güvenlik sınırlarını sağlamasının yanı sıra, bilinen bütün atak yöntemlerine karşı da güvenli olması beklenilmektedir. Yeni özet fonksiyon standardının, çeşitli amaçlarda kullanılmak üzere değişik özet boylarını desteklemesi beklenmektedir. Ayrıca, tasarım yazılımsal ve donanımsal kodlamalar yönünden verimli olmalıdır.

Bu tezde, yukarda belirtilen bütün özellikleri sağlamak üzere tasarlanan ve yarışma adayı, yeni bir kriptografik özet fonksiyonu ailesi olan Sarmal anlatıldı. Tasarım, güvenlik ve verimlilik arasındaki ödünleşimi en iyi şekilde sağlamak için blok tipi algoritma tasarımında

sıklıkla kullanılan parçalardan oluşturulmuştur. Öte yandan, Merkle-Damgård standardındaki zayıflıkların önüne geçmek ve esnek özet boyu sağlamak için HAIFA kullanılmıştır. Ayrıca, yapılan yazılımsal kodlamalar Sarmal'ın bir çok platformda çok verimli çalışabileceğini göstermiştir.

Anahtar Kelimeler: Sarmal, Dizayn, Özet Fonksiyonu

To My Aunt and Family

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor, Ali DOĞANAKSOY, whose guide and support, added considerably to my graduate experience.

It is a great pleasure to thank my colleague Onur ÖZEN for his patience, endless support, and motivation. Our discussions ended up with many usefull ideas.

I want to thank Çelebi KOCAIR for his hospitality, and support on implementation issues, Meltem Sönmez TURAN for her instructive comments, Çağdaş ÇALIK for his collaboration on technical issues, and Deniz TOZ for her corrections and suggestions. I also would like to thank everyone at IAM..

I cannot end without thanking my family, on whose constant encouragement and love I have relied throughout my time at the University.

This study was supported by The Scientific and Technological Research Council of Turkey (TUBITAK) Grant No: 107T544.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

Introduction

Practically since humans began writing, they have been writing in code, and ciphers have decided the fate of empires throughout recorded history. *Cryptology* is derived from the Greek word *kryptos* meaning hidden. It is composed of two parts. First of them is *cryptography*, which is the science of keeping secrets secret. This is done by hiding the meaning of the message, not the message itself, by a process known as encryption. Each encryption method has a distinct algorithm, and a secret key is required to perform. The other is *cryptanalysis*, which is the science of finding the weaknesses in these algorithms and breaking into the message without the knowledge of the key. Over centuries, the history has witnessed the challenge between these two sciences.

Over 4000 years, cryptography has been used to conceal sensitive information, with an increasing importance throughout the centuries. Having contributed to the birth of modern computer, cryptanalysts began using technology to break all sorts of ciphers. Therefore, cryptographers began designing more complex ciphers for exploiting the power of the computers. In short, both cryptography and cryptanalysis have evolved in parallel by a considerable amount.

In the 1960s, when the computers became more powerful and cheap enough for the businesses, standardization became a must. Although companies had particular encryption systems for internal communication, they still needed a common system for communicating the other companies. To fulfill this need, Lucifer was developed by IBM.

In the 1970s, with the beginning of the information age, the exchange of digital information became an essential part of our society. By the end of 1990s, electronic mail became more popular than conventional mails, and nowadays billions of e-mails are sent each day. The internet has also provided the infrastructure for e-commerce, online banking, e-government applications, etc. However, the success of the information age depends on its ability to protect the information, hence, on the power of cryptography. Therefore, people needed cryptography in order to protect their privacy besides the national security.

In addition to privacy, it is equally important to provide confidentiality, authentication, non-repudiation and data integrity. Hash functions are fundamental components of many cryptographic applications such as digital signatures, random number generation, message integrity, authentication, e-cash etc. Employing hash functions for these applications both increase the security and improve the efficiency of these systems.

Hash functions are categorized into two groups; (i) *keyed* and (ii) *unkeyed* hash functions. Keyed hash functions input a fixed length key and a message of arbitrary finite length. Message Authentication Codes (MACs) are examples of keyed hash functions. Unkeyed hash functions only input message and they involve no secrecy. Modification Detection Codes (MDCs) are examples of unkeyed hash functions. They can further be divided into two groups as *One Way Hash Functions (OWHF)* and *Collision Resistant Hash Functions (CRHF)*. This classification is summarized in Figure 1.1.

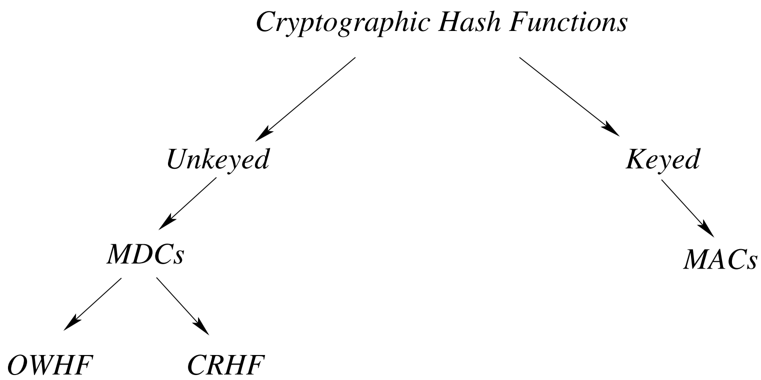

Figure 1.1: Categorization of Cryptographic Hash Functions

Most commonly used hash functions are MD5 (Message Digest) [2], SHA-1 (Secure Hash Algorithm) [3] and RIPEMD [4]. These algorithms are used in many applications such as SSL, PGP, S/MINE, SSH and SFTP. Comparison of commonly used hash functions are provided in Table 1.1.

Table 1.1: Comparison of Commonly Used Hash Functions

Hash Function	Hash Size	State Size	Block size Length	Max. Message Size	Collision
MD2 [5]	128	384	128	-	Almost
MD4 [6]	128	128	512	$2^{64}-1$	Yes
MD5 [2]	128	128	512	$2^{64}-1$	Yes
RIPEMD [7]	128	128	512	$2^{64}-1$	Yes
RIPEMD-128/256 [8]	128/256	128/256	512	$2^{64}-1$	No
RIPEMD-160/320 [8]	160/320	160/320	512	2^{64-1}	No
SHA-0 [3]	160	160	512	$2^{64}-1$	Yes
SHA-1 [3]	160	160	512	$2^{64}-1$	With flaws
SHA-256/224 [3]	256/224	256	512	$2^{64}-1$	With flaws
SHA-512/384 [3]	512/384	512	1024	$2^{128}-1$	No
Tiger [9]	192/160/128	192	512	$2^{64}-1$	No
PANAMA [10]	256	8736	256	-	With flaws
RadioGatún [11]	Arbitrary	58 words	3 words	-	No

NIST Competition

The design of the commonly used hash functions are based on MD4, as they iteratively use a compression function that inputs state variable and a fixed length block, and outputs another fixed length block. Recently, many attacks against hash functions having similar construction to MD4 are proposed [12, 13, 14, 15]. These recent studies motivated National Institute of Standards and Technology (NIST) to announce a public competition in 2007 to select a new cryptographic hash function to be used as the new standard [16]. Minimum requirements of the competition are given as;

- the algorithm must be publicly available,
- the algorithm must be implementable in a wide range of platforms and
- the algorithm must support 224, 256, 384 and 512 bit message digests and message length of at least $2^{64}-1$ bits.

The candidate algorithms will be compared based on their security, computational efficiency, memory requirements, hardware and software suitability, simplicity, flexibility and licensing requirements.

Overview of the Thesis

In this thesis, we aim to design a new hash function as a candidate of the NIST competition. In Chapter 2, basic properties, generic attack methods and construction methods are presented. In Chapter 3, our design, Sarmal is described. Chapter 4 is concluded the thesis.

CHAPTER 2

Cryptographic Hash Functions

A hash function takes a message as an input and produces output or digest which is called a hash value or hash. A hash function can be defined as $H : D \to R$, where input values are taken from a domain D and output values go to a range R. This function is always many-to-one and $|D| > |R|$. Thus, there has to be always collisions by pigeonhole principle and this can be seen in Figure 2.1(different input values with same output value). In a cryptographic hash function H, it is expected that each output values are seen equally likely to avoid finding collisions easily.

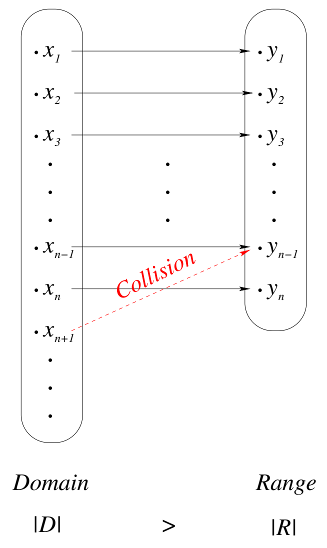

Figure 2.1: Pigeonhole Principle

2.1 Basic Properties

Hash functions take arbitrary length input and produce a fixed length output which is commonly called *fingerprint* or *message digest* of the input. Some desired structural and security-wise properties of cryptographic hash functions are given below.

Structural Properties

1. Algorithm of a hash function should be publicly known. There may not be any secret parameters.
2. For a given value x and a hash function H, it should be 'easy' to compute $H(x)$.

Security-Wise Properties

1. Preimage resistance: For a given hash value $H(x)$, it should be 'hard' to compute x.
2. Second preimage resistance: Given x and its hash value $H(x)$, it should be 'hard' to find $x^{'}$ such that $x \neq x^{'}$ and $H(x^{'}) = H(x)$.
3. Collision resistance: It should be 'hard' to find x and $x^{'}$ such that $x \neq x^{'}$ and $H(x) = H(x^{'})$.

This thesis is mainly focused on collision resistant hash functions which must satisfy all the conditions above.

2.2 Attack Methods against Hash Functions

Attacks on hash functions can be divided into two types [17]. First of them is *generic attacks* which are independent from the specification of the hash function and they mainly exploit the weaknesses of the hash functions in a general way. Second type attacks focus on structural weaknesses and exploits the weaknesses of the algorithm. In this section, these two attack types are going to be described.

2.2.1 Attacks Independent of Hash Function

Birthday Attack: It is based on the generalized birthday problem. For a set of n elements, if two sample spaces are chosen as s_1 and s_2 in that set, then the probability of a match from these two spaces is approximately $1 - e^{-\frac{|s_1||s_2|}{2^n}}$. Moreover, if $|s_1| = |s_2| = n^{1/2}$, then the probability of the match closes to 0.63. Birthday problem is used to find collisions in the hash function. For a given hash function $H(x)$ with output length n, there exist 2^n different hash values and if one chooses $\sqrt{2^n} = 2^{n/2}$ different messages, a collision is expected with probability greater than $1/2$ according to the birthday problem.

Preimage Attack: For a given hash value, one chooses random messages and expects that the given hash value is going to be obtained. It is assumed that all output values are seen equally likely for the cryptographic hash functions. Therefore, if the output length of the hash function is given as n bits, then after 2^n trials, one message's output value is expected to satisfy the given hash value. This attack can be thought as exhaustive search to the hash functions.

Second Preimage Attack: For a given message and its hash value, one chooses random messages to obtain the same hash value. Again, due to the fact that all output values are seen equally likely, if the hash value is n bits, after 2^n message trials, one message value's hash is going to satisfy the given message's hash value. Again, this attack can be thought as exhaustive search to the hash functions.

The attacks on hash functions based on the generic attacks above can be described as follows:

- **Collision Attack:** Aim of this attack is to find two colliding pair of messages for the given hash function less than $2^{n/2}$ trials.
- **Near-Collision Attack:** In this attack scenerio, one tries to find two message pairs whose hash values are not same but difference between them is as small as possible. Moreover, message pairs must be found less than $2^{n/2}$ trials.
- **Pseudo-Collision Attack:** Most of the hash functions use Initial Values (IVs) which are fixed by the designers. Pseudo-collision attack is applied with choosing the IVs to

find collisions and exploit the weakness of the hash function. The total complexity of this attack is also same as other collision attacks and after $2^{n/2}$ trials, pseudo-collisions can be found.

- **Pseudo-Near-Collision Attack:** This attack is combination of 2 and 3. Attacker chooses the IV value and tries to find a near collisions after $2^{n/2}$ trials.

2.2.2 Attacks Dependent to Hash Function

Meet in the Middle Attack: Meet in the middle attack is adopted from the birthday attack. It enables to construct a message value whose hash is same with a given one. A hash function with an invertible compression function is required. Therefore, it is mostly applicable to the iterated hash functions. In this attack, chaining values will be compared rather than hash values. Attack works with going forward from initial value to an intermediate value with a sample space s_1 and going backward from hash value to the intermediate value with another sample space s_2. The probability of obtaining same intermediate chaining value from two different sample spaces equals to $1 - e^{-\frac{|s_1||s_2|}{2^n}}$ by birthday problem where n is the length of the hash and chaining values.

Fixed Point Attack: This attack is applied to hash functions that use a compression function. For the compression function $f(h_{i-1,}, m_i) = h_i$, a chaining value h is searched where $f(h, m_i) = h$. This means that the message value m_i does not affect the result of the hash value. Thus, m_i can be used to obtain second-preimage attacks. In other words, for a message value $m = m_1 m_2 \cdots m_t$, if a fix point found for its i^{th} element m_i ($f(h, m_i) = h$), then $m^* = m_1 m_2 \cdots m_{i-1} m_{i+1} \cdots m_t$ also gives the same hash value where $m \neq m^*$ [17].

Differential Attack: Differential attack to the hash functions is based on the Differential Cryptanalysis of block ciphers [18]. Mostly, it is applicable to block cipher based hash functions. The aim of the differential attack is to find a collision for a hash function (i.e. Zero difference is expected at the end). Moreover, differential attack is also used in an intermediate step to find internal collisions for hash functions and recent works on hash function cryptanalysis commonly use this type of attacks [19, 20, 21, 22].

2.3 Iterated Hash Functions

A common way to construct a hash function is to use iterations. Figure 2.2 gives a general method of constructing iterative hash functions. It was described by Merkle [23] and Damgård [24] in 1990 independently. The initial version of this iteration method has some weaknesses against long-message attack to obtain second-preimages. Thus, it was strengthened by addition of message length as a last message block.

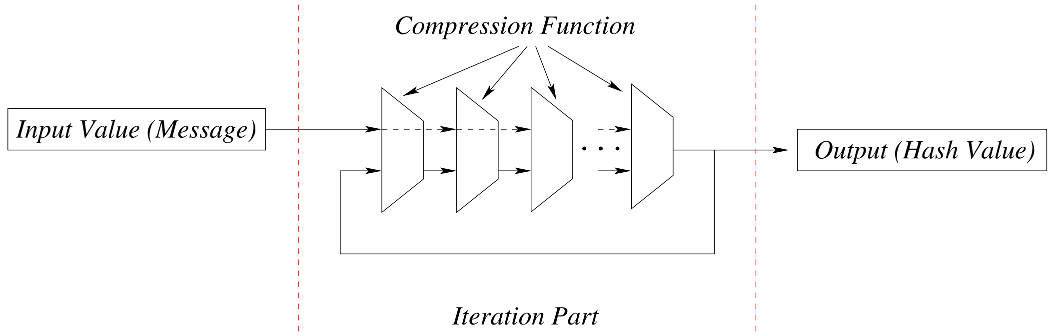

Figure 2.2: Iterated Hash Function

Merkle-Damgård - Strengthening is proceeded as follows: Firstly, a compression function $f : \{0,1\}^m \times \{0,1\}^n \rightarrow \{0,1\}^n$ is taken which uses messages of length m and *chaining variables* of length n and outputs the next *chaining variable* of length n. Thus, a given message M is padded to obtain a message of length multiple of n, then it is divided into $t-1$ equal pieces and message length of unpadded message is added as t^{th} piece. At each iteration, inputs m_i and h_{i-1} is used to derive output h_i. Output of i^{th} iteration h_i, is called the *chaining variable* or the *intermediate variable*. The first chaining variable is called the *Initial Value*. The iterative hash functions can be described as follows and the generalized scheme can be seen in Figure 2.3:

$$\begin{aligned} h_0 &= IV, \\ h_i &= f(m_i, h_{i-1}),\ i = 1, 2, \ldots, t \\ H(m) &= h_t \end{aligned}$$

Theorem 2.3.1 *If the given compression function $h(x)$ is collision-resistant, then the hash function $H(x)$ is collision resistant.*

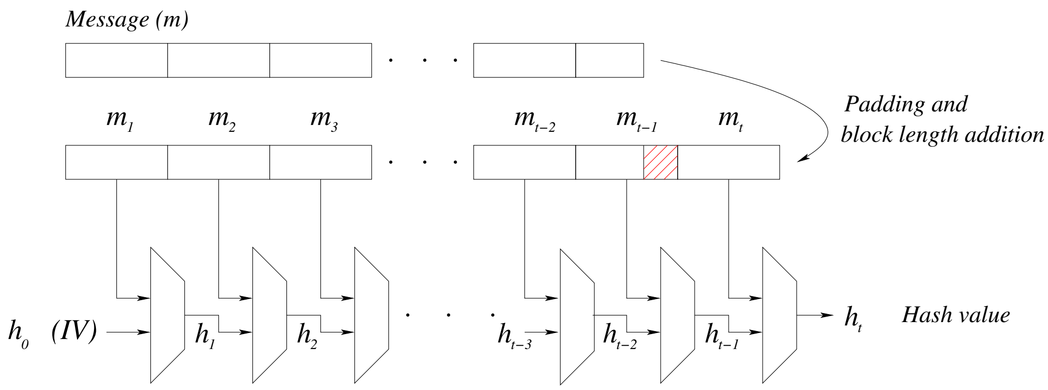

Figure 2.3: Merkle-Damgård Construction (Strengthening)

Proof. A proof of this theorem can be found in [25]. ■

As the result of this theorem, collision-resistant compression functions are become popular. Various constructions are published based on iteration. Some of them use block ciphers as compression function, some use stream ciphers etc. The hash functions based on block ciphers are going to be studied in next section detailed.

2.3.1 Attacks on Merkle-Damgård Strengthening

There exist many hash functions which utilize Merkle-Damgård construction today. Therefore, many articles related to security notions of this construction are published. Some of them are going to be summarized below.

Length Extension Attack: Let a long message $M = m_1 m_2 \cdots m_t$ be hashed with Merkle-Damgård construction without any Merkle-Damgård - Strengthening. A message M^* can be found such that $H(M) = H(M^*)$ by choosing a random message m^* and computing $f(IV, m^*)$ and checking for the collision between $f(IV, m^*)$ and the chaining variables of $H(M)$. If it is found, m^* can be concatenated to M rather than the messages that are computed up to that point where the same chaining variable is obtained. This yields M^* ($\neq M$) whose hash value is equal to hash of M. Therefore, M^* is called a second-preimage of M and this attack can be defined as second-preimage attack against Merkle-Damgård strengthening.

Multi-collision Attack: Joux [26] expressed that finding multi-collisions in the iterative hash functions is not harder than finding a collision. First, two single length messages are found such that $h_1 = f(h_0, m_1) = f(h_0, m_1^*)$ and this operation can be repeated until t-hash value is obtained such that $h_i = f(h_{i-1}, m_i) = f(h_{i-1}, m_i^*)$. Then, 2^t different multi-collisions can be obtained with a complexity $t \times 2^{n/2}$ rather than $2^{n(t-1)/t}$. For the case $t = 2$, the attack can be described as after finding $h_1 = f(h_0, m_1) = f(h_0, m_1^*)$ and $h_2 = f(h_1, m_2) = f(h_1, m_2^*)$, $2^2 = 4$ different collisions can be obtained as evaluating the hash values of concatenation of the messages $H(m_1 \| m_2) = H(m_1 \| m_2^*) = H(m_1^* \| m_2) = H(m_1^* \| m_2^*)$. Figure 2.4 illustrates the multi-collision attack.

Using multi-collisions, Joux also showed that concatenation of two different hash functions does not improve the collision resistance.

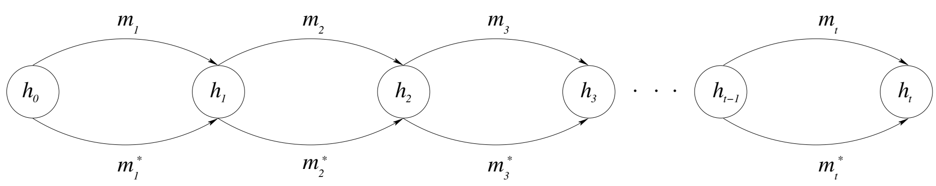

Figure 2.4: Multi-Collision Attack

Fixed Point and Second Preimage Attacks: It is stated by Dean [27] that for an iterative hash function, if the fix points of compression function can be calculated easily, then finding second-preimages is easier than expected. Davies-Meyer construction fits this condition well. Its compression function can be written as $h_i = E_{m_i}(h_{i-1}) \oplus h_{i-1}$ where E denotes the block cipher and subscripted value denotes the key value of block cipher. For a fixed point h, the equality will be $h = E_{m_i}(h) \oplus h$ and $E_{m_i}(h) = 0$ or $h = E_{m_i}^{-1}(0)$. For this kind of a compression function, it is easy to calculate fix points for randomly selected messages.

Either using Davies-Meyer construction or another one, if the fixed points can be calculated easily, then Dean's attack can be applicable and it works as follows:

1. Find $O(2^{n/2})$ fix points which is denoted by a set A and n is the length of hash value.
2. Compute the chaining values of $O(2^{n/2})$ single message blocks by taking $h_{i-1} = IV$ and call it set to B .

3. Search for the matches between the values in A and B. Due to the choice of number of elements in A and B, there must be a colliding pair in the sets A and B with the probability greater than $1/2$. After finding the match, concatenation of the message from A to the message from B that give the collision, one gets the m^* in the length extension attack. The obtained message length can be extended to the original message length by adding the fixed points required times. The sketch of the attack is given in Figure 2.5

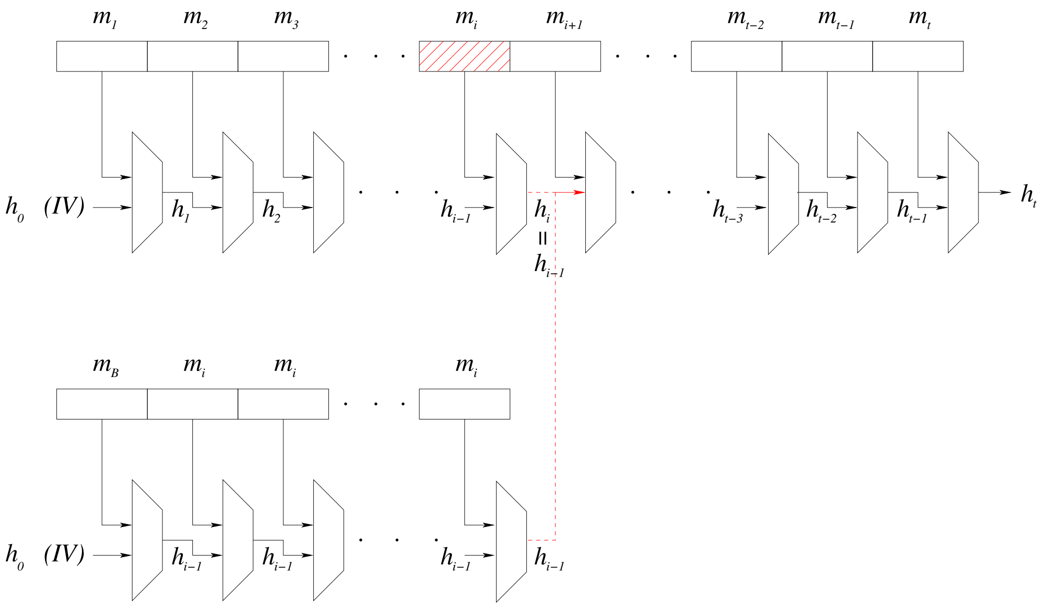

Figure 2.5: Dean's Fixed Point Attack

Kelsey and Schneier improved this attack to the case where it is not easy to find fixed points [28]. They used the multi-collision technique to by pass first two steps of Dean's attack and then third step is applied to find second-preimages.

Herding Attack: This attack type is based on time-memory trade off and was introduced by Kelsey and Kohno [29]. Attack has an offline and an online phase. In the offline phase of the attack, 2^t different chaining variables are chosen first, then with the aid of $O(2^{n/2-t/2})$ single message blocks, next chaining variables are computed and it is expected that some of these hash values are collided. This step is repeated until one chaining variable left and that value can be used as hash value. In the online phase, 2^{n-t} operations are performed with 2^{n-t} different messages to connect the prefix to the one of 2^t different chaining variable. (Length of the hash value is denoted as n.) The attack can be seen in Figure 2.6.

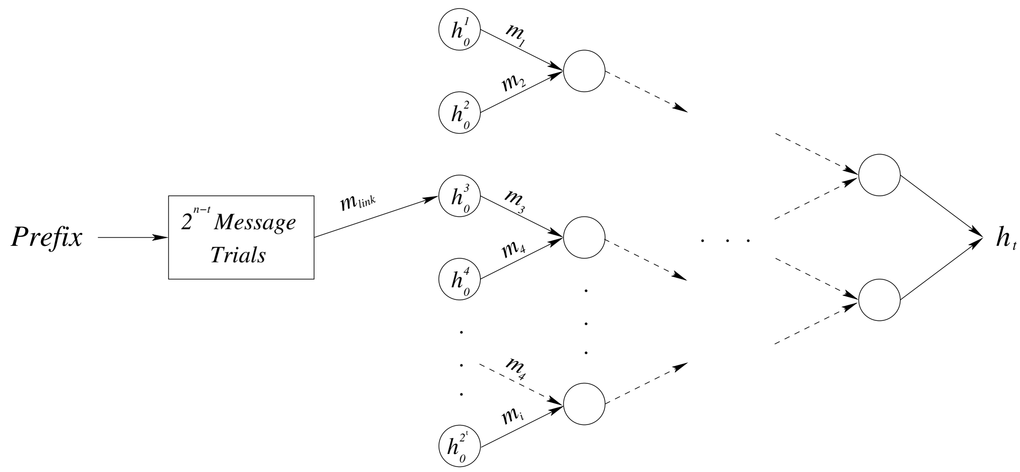

Figure 2.6: Herding Attack

2.4 Hash Iterative Framework (HAIFA):

In the previous section, drawbacks of Merkle-Damgård construction are given. HAsh Iterative FrAmework (HAIFA) is proposed by Biham and Dunkelman [30] to patch these problems and generalize the iterative hash function schemes. Figure 2.7 gives the general scheme of HAIFA. It is claimed that all the good properties of Merkle-Damgård construction is preserved and security is improved also variable hash size is enabled.

In the Merkle-Damgård construction, compression function uses a chaining value of size n and a message block of size n. In addition to these input values, a bit counter of size b and a salt of size s are used in HAIFA and it is defined as $f : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^b \times \{0,1\}^s \rightarrow \{0,1\}^n$ and $h_i = f(h_{i-1}, m_i, bh_i, s_i)$ where bh denotes the number of bits hashed so far and s denotes the salt. A message is hashed after three main and one optional steps which are :

1. Padding
2. Computation of IV
3. Iteration of compression function f

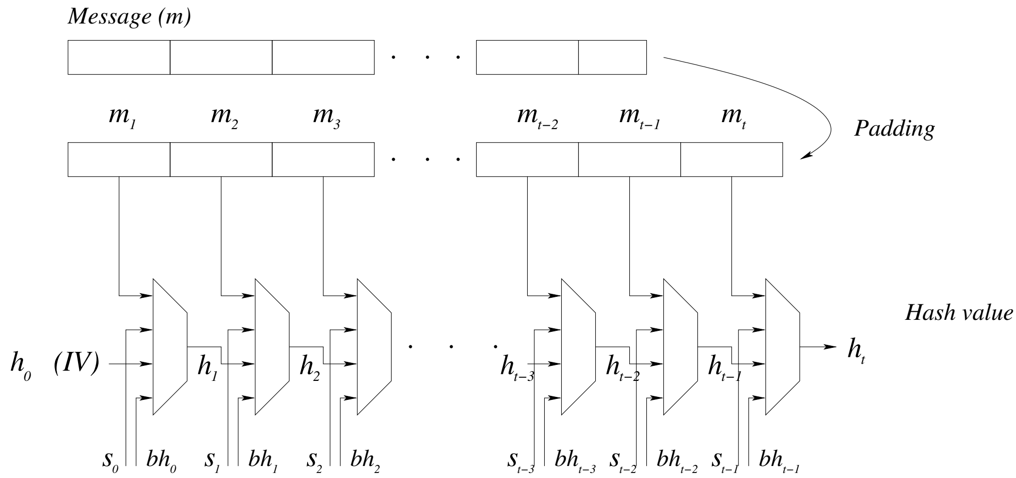

Figure 2.7: General Scheme of HAIFA

4. Truncation of the final chaining value (Optional)

Padding: The padding in HAIFA is very similar to the Merkle-Damgård Strengthening's padding. Moreover, the hash size is added as last *r*-bits of the message. The full padding can be described as:

1. Add a bit 1 to end of the message.
2. Add 0-bits that follows the bit 1. The required number of zeroes is decided by checking whether the length of padded message is a multiple of *n* after *t*-bits of message length and *r*-bits of hash size added.
3. Add the message length in *t*-bits.
4. Add the hash size in *r*-bits.

Addition of hash size is related to preventing the variable size hash outputs against the collision attacks. Also, the last block of the message can be identified by compression function.

Computation of IV: Initial value is computed with the operation $IV = f(IV_0, m_e, 0, 0)$ where IV_0 is a fixed value and m_e is the encoded version of the hashing message. m is first described in k-bits. Then, a single bit 1 and $n - k - 1$ zeroes is padded to the obtain encoded message m_e.

Iteration of compression function f: h_i is found by computing $f(h_{i-1}, m_i, bh, s)$ and this operation is repeated until message blocks ends in the iteration part. There does not exist any difference between iteration method of Merkle-Damgård and HAIFA. Only difference is between the compression functions.

Truncation of the final chaining value: This step is optional. If different hash sizes are required for the same hash function, truncating the last compression value can be applied. This process was previously used in the hash functions $SHA - 256$ and $SHA - 512$ to obtain $SHA - 224$ and $SHA - 384$ respectively. Except their initial values and constants, overall the structure is the same in this constructions. On the other hand, this can cause some problems. If the same chaining values in the first few blocks are obtained by applying collision attacks, then the remaining operations and the obtained hash values (up to truncation) will be the same. Therefore, the hash size of the message is added to the message in the padding part to protect the construction against this kind of attacks.

Security of HAIFA: As mentioned before, HAIFA can be considered like a generalized version of Merkle-Damgård construction. The proof methods of Merkle-Damgård construction can be applied to the HAIFA and collision resistance was proved by using the same arguments in the proof of Merkle-Damgård construction. Thus, it can be said that if the compression function is collision resistant, then HAIFA is also collision resistant [30]. Lately, it was also claimed that if the compression function of HAIFA is an ideal cipher or a random oracle, then second-preimages can be found with 2^n work which is optimal case. It was presented in a Workshop ("Hash functions in cryptology: theory and practice ") but not published yet.

It must be also showed that the attacks on Merkle-Damgård construction do not work on HAIFA. The additional variables in the construction mainly concentrate on preventing HAIFA against these attacks. In the proposal of the HAIFA [30], it is stated that the addition of number of bits hashed so far into the chaining variable provides resistance against fix point attacks. Multi-collisions cannot be be pre-computed without knowing the exact salt value.

In the second-preimage attack by Kelsey and Schneier, salt value must be known to produce the expandable message. If it is not known, an expandable message can be produced for all values of salt or there will be no offline phase in the attack. Herding attack is not feasible if the salt value is choosing at least 64-bits length in the HAIFA [30]. Table 2.1 shows the required works to apply the basic attacks to Ideal, Merkle-Damgård and HAIFA hash functions which is taken from [30].

Table 2.1: Complexities of Attacks on Ideal Hash Function, Merkle-Damgård and HAIFA (Compression functions of Merkle-Damgård and HAIFA are considered as ideal compression functions)

Type of Attack	Ideal Hash Function =	Merkle-Damgård ≥	HAIFA (fixed salt) ≥	HAIFA (distinct salt) ≥
Preimage	2^n	2^n	2^n	2^n
One-of-many Preimage ($k < 2^s$ messages)	$2^{n/k}$	$2^{n/k}$	$2^{n/k}$	2^n
Second-preimage (l-blocks)	2^n	$2^{n/l}$	2^n	2^n
One-of-many Second-preimage (l-blocks, $k < 2^s$ messages)	$2^{n/k}$	$2^{n/l}$	$2^{n/k}$	2^n
Collision	$2^{n/2}$	$2^{n/2}$	$2^{n/2}$	$2^{n/2}$
Multi-collision (t-collision)	$2^{n(t-1)/t}$	$\lceil log_2(t) \rceil 2^{n/2}$	$\lceil log_2(t) \rceil 2^{n/2}$	$\lceil log_2(t) \rceil 2^{n/2}$
Herding Online: Offline	-	2^{n-t} $2^{n/2+t/2}$	2^{n-t} $2^{n/2+t/2}$	2^{n-t} $2^{n/2+t/2+s}$

2.5 Construction of Hash Functions

In the previous sections, the basics of a hash function and some required information that helps to understand the following chapters are given. In this section, some important construction methods of the cryptographic hash functions are going to be described.

Up to now, many different construction methods are proposed to obtain a collision resistance hash function and new construction methods are also being developed. Some of them uses NP-complete mathematical problems, some uses well known cryptographic components like block ciphers and others are designed with totally new strategies. In this section the following construction methods are described:

1. Provably secure hash functions,
2. Block cipher based hash functions,
3. Sponge function based hash functions.

Each choice has some advantages and disadvantages and research is going on to find the best construction method. In the following sections, hashing methods that are given above are examined.

2.5.1 Provably Secure Hash Functions

The designs of this type of hashing are based on hard mathematical problems. Some of the problems can be reduced to NP-complete problems, which are also used in public key cryptography. On the other hand, the hardness of the some mathematical problems are also used to design a hash function. Thus, some schemes are badly broken after finding solutions to the defined problems.

The security bounds for a CRHF can be obtained and proved in this type of constructions easily. Due to the operations that are used for hashing, provably secure hash functions are slower than the others.

One of the first examples of this type hashing is given by Gibson [31] whose proposal is based on discrete logarithm problem. Three years after in 1994, Bellare et al.[32] designed a hash function which is also based on discrete logarithm problem. In 2006, Very Smooth Hash (VSH) [33] was presented in the Eurocrypt where the underlying number-theoretic problem can be reduced to finding non-trivial modular square root of very smooth number.

There also exists some hash functions based on expander graphs. These make use of the problems in the graphs and groups properties. LPS [34], ZT [35] and Prizer [34] hashes are the examples of expander graph based hashing. LPS hash is completely broken today. ZT and Prizer hashes are unbroken. The mathematical problems in these hashes could not be reduced to an NP-complete problem. Therefore, there exists always a possibility to deduce some weaknesses of the these hash functions.

There exist also some provably secure hash functions based on *Knapsack* [36, 37], *Lattice*

[38, 39, 40] and *Coding Theory* and *Fast Fourier Transforms (FFT)* [41].

2.5.2 Block Cipher Based Hash Functions

One of the main design types of hash functions is based on block ciphers where many ideas and theories have been developed in the last ten years related to this topic. The main problem of constructing a hash function from a block cipher is the bijectiveness (lack of one-wayness) of the block cipher. Thus, adopting the block ciphers to hash functions some extra operations are needed before and/or after encryption operations.

The block cipher based hashing is mainly preferred due to some reasons. First of all, more efficient hash functions can be constructed using block ciphers. Minimum requirements in hardware and better performances in hashing can be obtained. But there always exists a trade-off between the efficiency of the design and its security. The more efficient designs require very simple constructions and these are concluded with badly broken hashes like $MD-family$ which includes $MD-X$ ($MD2$, $MD4$, $MD5$ and $RIPEMD$) and $SHA-X$ ($SHA-0$, $SHA-1$ and $SHA-2$) hashes. The second reason why the block cipher based hashing is chosen that the block ciphers are well examined and exploited their weaknesses.

In block cipher based hashing, a block cipher is chosen as a compression function, then it is iterated for some rounds to produce hash values. After Merkle and Damgård showed iterative hash functions are collision resistant if the compression functions are collision resistant, block cipher based hashing became more popular. Most of the hashes, which are chosen as standard and used in many applications today, are also based on block ciphers. They use very simple encryption functions. Thus, they are more efficient than provably secure hash functions.

Choosing a block cipher did not provide the security margins in $1990s$ while the block length of the ciphers were 64-bit and it leads only 2^{32} complexity to get a collision. Therefore, more than one block cipher were used in the designs and various lengths of hash values are obtained. But, most designers concentrated on the following constructions:

1. Size of hash value equals to the block length of the block ciphers

2. Size of hash value equals to twice the block length of the block ciphers

Different lengths of hash values and different number of block ciphers in the new designs come up with one question: Which one is more efficient? The following definition enables to compare block cipher based hash functions.

Definition 2.5.1 *The rate R of a hash function is defined as hashed message block per encryption.*

Low rate hash functions with single length are mostly preferred rather than high rate hash functions due to the efficiency problems of high rate hash functions. Thus, hash functions with single block lengths examined in the following paragraphs. Then, some examples of double block length hash functions are going to be given and finalized with other examples of efficient designs which have neither single nor double block length.

Single Block Length Hash: All the designs with single block length have rate-1. Most known are given by Davies-Meyer [42], Matyas-Meyer-Oseas [43], Miyaguchi-Preneel [44, 45] which are also given in Figure 2.9 [1, 5, 6] respectively.

Preneel, Govaerts and VandeWalle (PGV) Constructions: PGV [45] defined single-length block hash with rate-1 in a general form. In the PGV construction, a block cipher is considered like Figure 2.8 where the block cipher takes two input values: Plaintext P and key k. It gives an output value which is XORed with an feedforward value FF. The values P, k, FF are thought as chosen from the set $\{m_i, h_{i-1}, m_i \oplus h_{i-1}, C\}$. Therefore, there exist $4^3 = 64$ possible construction methods and it was stated that only 12 of them (Figure 2.9) are seemed secure.

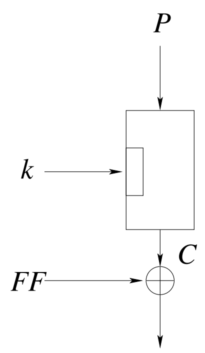

Figure 2.8: General Scheme (The key of the block cipher is shown as small box)

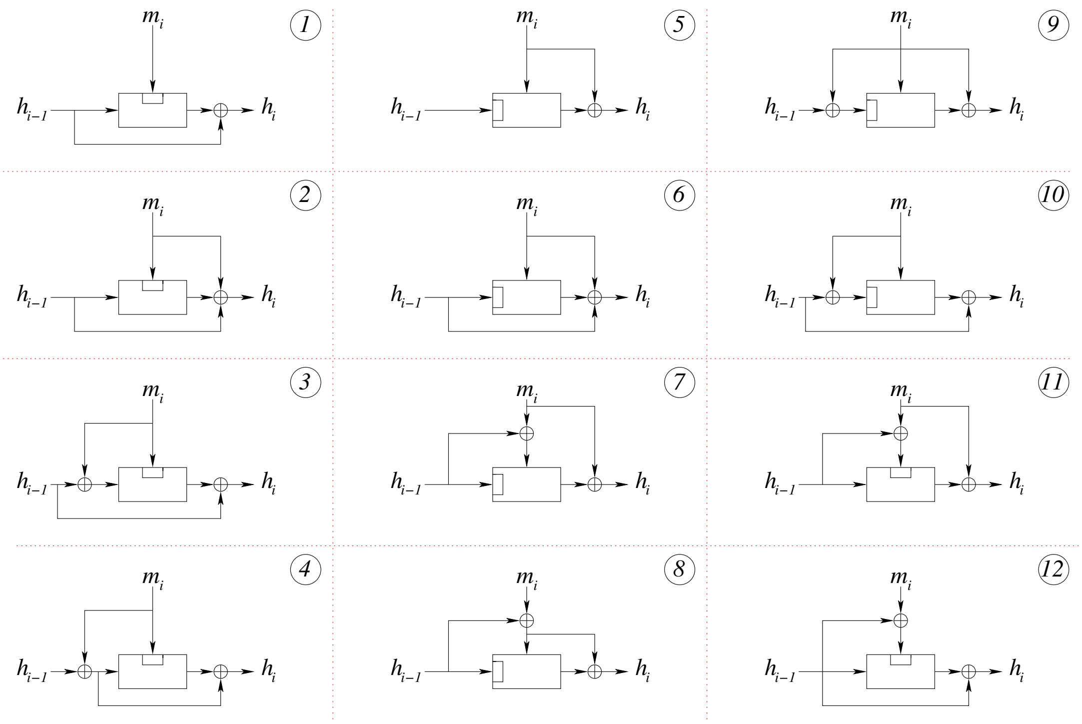

Figure 2.9: Twelve Secure Hash Constructions (The key of the block cipher is shown as small box)

In 2002, Black, Rogaway and Shrimpton [1] showed that collision resistance of the given constructions are close to the birthday bound and they also showed that 8 of them (Figure 2.10) are also collision resistant if they are iterated properly even though they do not have a secure compression functions. Stam [46] also shows the collision resistance of the constructions in a different way.

Double Block Length Hash: In the block cipher based hash functions if the output length is equal to the double block length of the cipher than it is called double block length hashing. Most known examples are $MDC-2$ [47] and $MDC-4$ [47]. The numbers at the end describes the required block cipher calls in the compression function. The design principle of these two hashes was to produce double length hashes using well known cipher Data Encryption Standart (DES) [48]. As it can can be seen from Figure 2.11 rate of these hash functions are $1/2$, $1/4$ respectively. Both use extended version of the Matyas-Meyer-Oseas scheme which is stated in Figure 2.9 as number 5.

Another example to double-length hashing is given by Nandi et al.[49] with rate $2/3$. The

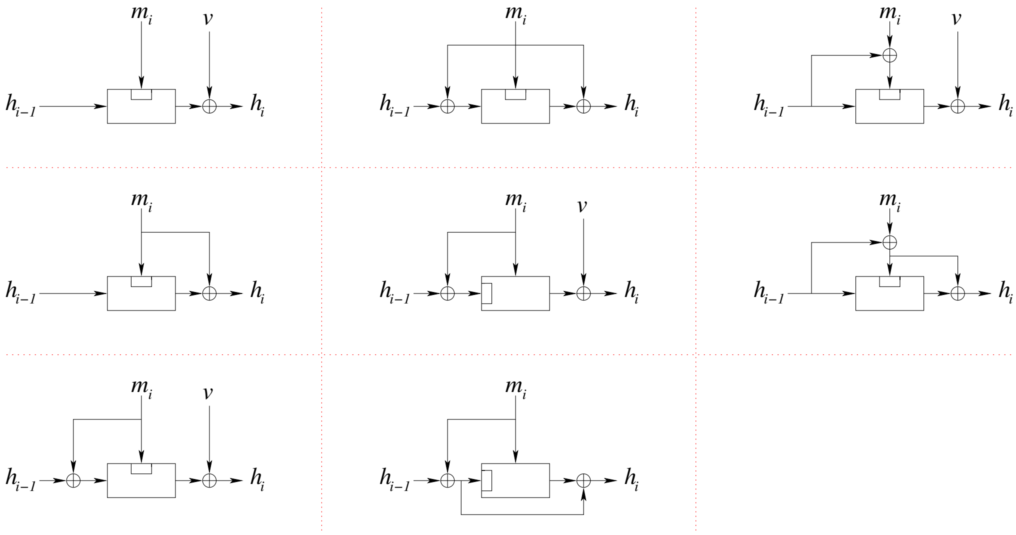

Figure 2.10: Eight Secure Hash Constructions Defined by Black et al. [1] (The key of the block cipher is shown as small box)

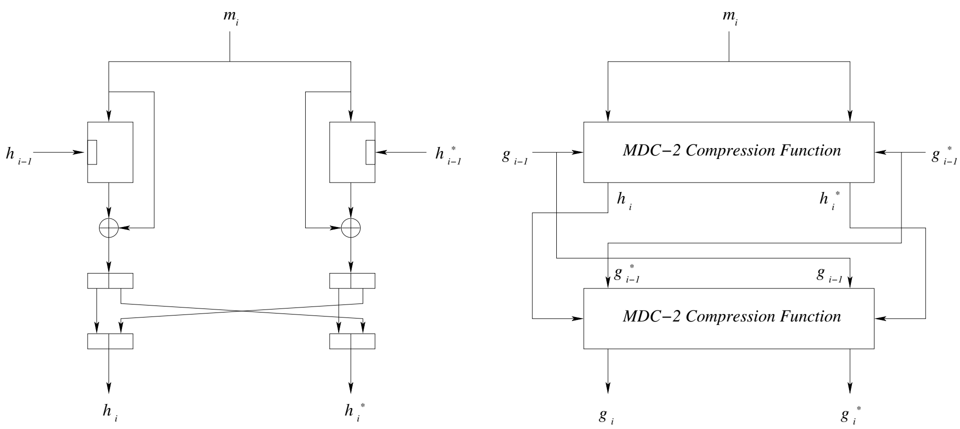

Figure 2.11: $MDC-2$ and $MDC-4$

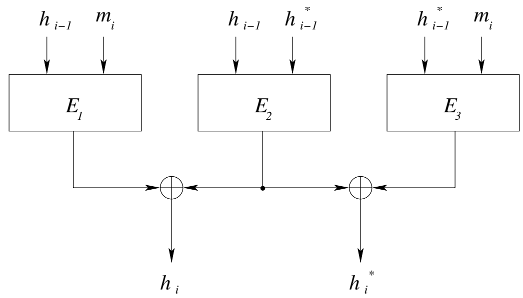

Figure 2.12: Nandi's Double-length Block Hash Design

hash function can be seen in Figure 2.12. In addition to this construction, there exist also Abreast-DM [50], Parallel-DM [51], Hirose family [52], Merkle's constructions based on *DES* and PBGV [53] constructions with double-length and different rates.

For all constructions mentioned above, information theoretic security bounds are discussed in the proposals and also in related articles. Knudsen, Lai and Preneel [54] investigated the security of double-length block hashing with rate-1. Rate $1/2$ constructions were studied by Hohl, Lai, Meier and Waldvogel [51]. Moreover, double-length block hashing with double-length key value was discussed by Satoh, Haga and Kurosawa [55] and Hattori, Hirose and Yoshida [56].

Larger than double block length hash functions are also introduced by Preneel and Knudsen [57]. They used block cipher based hash functions with Quaternary Codes.

2.6 Sponge Function Based Hash Functions

Sponge functions are defined by Bertoni et al. [58]. They are iteration of finite states. Using a sponge function it is possible to produce an infinite-length output from a variable-length input. Sponges can be used to construct both hash functions and stream ciphers.

Grindahl [59] is an example to constructing a hash function from a sponge function. It is designed by Thomsen et al. It supports 256 and 512 bits of output. It has some lacks in the collision resistance and that was showed by Peyrin [60]. Another example is Radiogatùn [11]. It is mainly a stream based hash function, but it can be also included into sponge function based hash construction.

It is expected from a cryptographic hash function that behaves like a random oracle and a random oracle does not have any weaknesses. When iterated hash functions are considered, there always exist inner collisions which can be defined as if two message pair m_1 and m_2 give the same chaining value, then concatenation of m_1 and m_2 with collide suffix m^* collide (i.e. $m_1||m^*$ and $m_2||m^*$ gives same hash value). In the sponge function construction, there also exist inner collisions and this is the only weaknesses of sponge functions so far. Gorski et al. [61] also showed that slide attacks can be applicable to sponge functions and gave two examples on MAC modes of Radiogatùn and Grindahl.

2.6.1 Stream Cipher Based Hash Functions

Some of the researches to find more efficient hashing come up with new hashing techniques. The construction method is based on neither a mathematical problem nor a block cipher. It is based on synchronous stream ciphers which are one of the important parts of the symmetric cryptosystems and they are suitable for applications where high speed is required. The hardware requirements are also lower then a block cipher constructions. Thus, some constructions were given in previous years.

Panama is the first stream based hash function which is designed by Daemen et al.[10] and badly broken by the designers [62, 63]. Then, strengthened version is proposed as Radiogatùn [11] in 2007. There is also a design which is based on the famous stream cipher RC4 [64] which is also broken by Indesteege and Preneel[65]. Performances of stream cipher based hash functions are better than the other constructions but there does not exist any mathematical or information theoretic proofs related to their security. The studies on the stream ciphers and their security can help to reach some results for the security results of stream based hashing.

CHAPTER 3

Sarmal: Cryptographic Hash Function Family

Most of the known hash functions and their updated versions were broken with last term attacks on hash functions [19, 20, 21, 22]. It is also showed that the commonly used construction method, Merkle-Damgård, is not secure as expected [26, 27, 28, 29]. NIST announced for a public competition to choose new hash function which will be a new hashing standard and called $SHA-3$ [16].

In the design of Sarmal, it is aimed to construct hash function, which satisfies the stated properties of new hash function. To achieve this, HAIFA is chosen as the construction method. Compression function of Sarmal consists of two linearly independent, identical branches which contains generalized Feistel networks. Whole construction is word oriented and Advance Encryption Standard(AES) type operations such as s-boxes and matrices, based on Maximum Distance Separable (MDS) codes, are used in the design. Message permutation is preferred rather than message expansion which are come up with extra implementation costs.

Sarmal is a hash function family which supports various size of hash digests(224, 256, 384 and 512 bit). The definition of Sarmal is given through 512-bit version. 384-bit version is defined in Section 3.1.3 and 224/256-bit versions are defined in Chapter B.

3.1 Description of Sarmal

3.1.1 Notation

Throughout this chapter, the following notation will be used. Each 512-bit block is composed of eight 64-bit *words* ($X[7], X[6], \ldots, X[0] = X[7-0]$). Note that the words and blocks are in

Table 3.1: Notation

$\oplus$	Bitwise logical exclusive OR (XOR)	s_{i-1}	128-bit salt value Â
$\boxplus$	Addition modulo 2^{64}	$s_{i-1}[j]$	j^{th} 64-bit word of 128-bit s_{i-1}
$\boxminus$	Substraction modulo 2^{64}	t_{i-1}	64-bit bit counter
X_i	512-bit intermediate value	c	256-bit constant value
$X_i[j]$	j^{th} 64-bit word of 512-bit X_i	$c[j]$	j^{th} 64-bit word of 256-bit c
h_{i-1}	512-bit chaining value	I	64-bit Input value
$h_{i-1}[j]$	j^{th} 64-bit word of 512-bit h_{i-1}	O	64-bit Output value
$S_i[.]$	8×8-bit S-box transformation	$M_{8\times 8}$	8×8 Maximum Distance Separable (MDS) Matrix

little-endian order (i.e. the least significant bit is the rightmost bit numbered 0, and the most significant bit is bit 63 for a 64-bit word.). The symbols that are used in the equations and figures are given in the Table 3.1.

Padding: A single bit 1 is concatenated to the message M, followed by zeros until the length of the message is 439 modulo 512. Afterwards, 100000000 is added and the 64-bit original message length is appended to the end.

Message Permutation: The compression function of Sarmal needs a 512-bit message in each iteration. The message is first divided into eight 64-bit words, then these words are permuted by $\sigma_k(m_i)$, which will be used in two consecutive rounds . Since, there are 16×2 rounds in the compression function, 16 permutations are needed. These permutations are given in Table 3.2. First eight permutations are used in left half and the remaining are used in the right half of the Sarmal.

Initial value: First 128 hexadecimal digits of π is taken as the initial value (i.e. IV or h_0) of Sarmal, and it is given in Table 3.3.

***s* and *t* values:** The s variable is used to show the salt value of the Sarmal where it is required in HAIFA construction to strengthen the structure. 128-bit s is used like a counter. It is initialized with an IV and incremented by one after each compression function calls and t shows the number of hashed bits so far. It is 64-bit and used in both left and right branches.

Table 3.2: Message Permutation

Left Part									Right Part								
$m_i[.]$	0	1	2	3	4	5	6	7	$m_i[.]$	0	1	2	3	4	5	6	7
$\sigma_1(m_i)$	0	1	2	3	4	5	6	7	$\sigma_9(m_i)$	3	4	0	2	5	7	6	1
$\sigma_2(m_i)$	6	0	7	1	2	3	4	5	$\sigma_{10}(m_i)$	0	7	6	3	4	2	1	5
$\sigma_3(m_i)$	5	2	6	4	0	1	7	3	$\sigma_{11}(m_i)$	1	0	5	4	3	6	7	2
$\sigma_4(m_i)$	1	7	3	2	5	4	0	6	$\sigma_{12}(m_i)$	7	2	1	5	6	0	3	4
$\sigma_5(m_i)$	4	3	5	7	1	6	2	0	$\sigma_{13}(m_i)$	6	1	3	7	2	5	4	0
$\sigma_6(m_i)$	3	4	1	6	7	0	5	2	$\sigma_{14}(m_i)$	5	3	4	0	7	1	2	6
$\sigma_7(m_i)$	2	6	0	5	3	7	1	4	$\sigma_{15}(m_i)$	2	6	7	1	0	4	5	3
$\sigma_8(m_i)$	7	5	4	0	6	2	3	1	$\sigma_{16}(m_i)$	4	5	2	6	1	3	0	7

Table 3.3: Initial Values of Sarmal

$h_0[0] = 243F6A8885A308D3_x$	$h_0[4] = 452821E638D01377_x$
$h_0[1] = 13198A2E03707344_x$	$h_0[5] = BE5466CF34E90C6C_x$
$h_0[2] = A4093822299F31D0_x$	$h_0[6] = C0AC29B7C97C50DD_x$
$h_0[3] = 082EFA98EC4E6C89_x$	$h_0[7] = 3F84D5B5B5470917_x$

Constants: Sarmal uses a 256-bit constant C which is divided into four 64-bit words. They are taken from the extension of square root of three. These values are given in Table 3.4.

Table 3.4: Constants of Sarmal

$C[0] = BB67AE8584CAA73B_x$	$C[2] = 25D834CC53DA4798_x$
$C[1] = 25742D7078B83B89_x$	$C[3] = C720A6486E45A6E2_x$

3.1.2 The Algorithm

Sarmal-512 accepts a message m of arbitrary length (no more than $(2^{64} - 1)$-bits) as input and outputs a 512-bit hash value $H(m)$. It uses a compression function $f(h_{i-1}, m_i, t_{i-1}, s_{i-1})$. In each iteration, the rightmost four words of h_{i-1} is concatenated with the rightmost word of the salt s_{i-1}, rightmost two words of the constant c, and t value. The 512-bit state is iterated 16 rounds in the right branch. $(X_0 = (X_0[7-0]) = (h_{i-1}[3-0], s_{i-1}[0], c[1-0], t))$

Similarly, the leftmost four words of h_{i-1} is concatenated with the leftmost word of the salt value s_{i-1}, leftmost two words of the constant c and t and the result is again processed for 16 rounds in the left branch. $(X_0 = (X_0[7-0]) = (h_{i-1}[7-4], s_{i-1}[1], c[3-2], t))$

The two outputs are XORed, and the resulting value is XORed with m_i. Figure 3.1 shows the general view of the compression function.

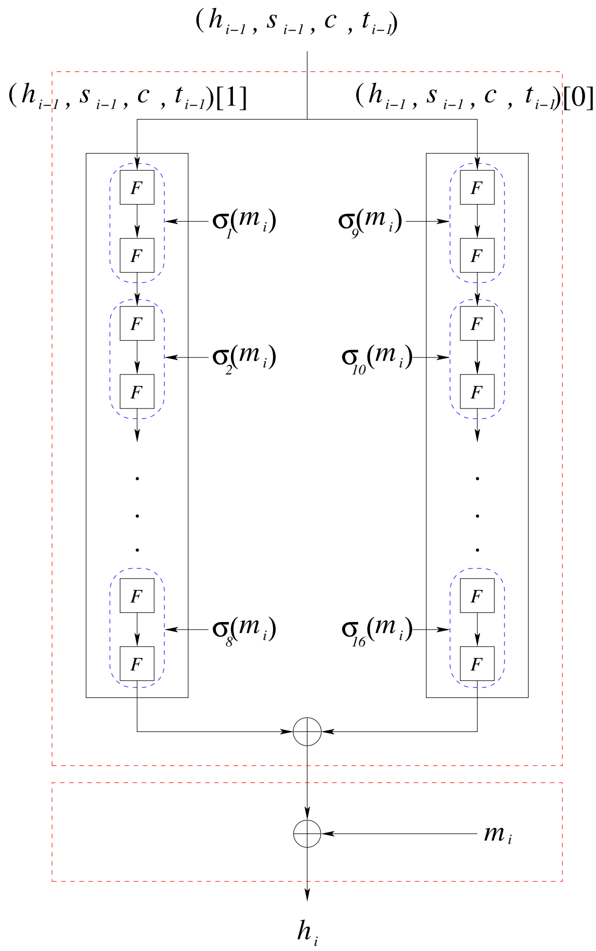

Figure 3.1: General View of Compression Function

Round Function: Let $F(x, w)$ denote the round function, where x and w are 512-bit and 256-bit inputs respectively, and w is obtained from m_i by a permutation σ_k $(k = 1, 2, \cdots, 16)$. For odd rounds w is the least significant four words of the given permutation, whereas for even rounds it corresponds to the most significant four words (i.e. $w = \sigma_k(m_i)[0-3]$ or $w = \sigma_k(m_i)[4-7]$). The F-function can be seen in the Figure 3.2 and may be formally described by the Algorithm 1:

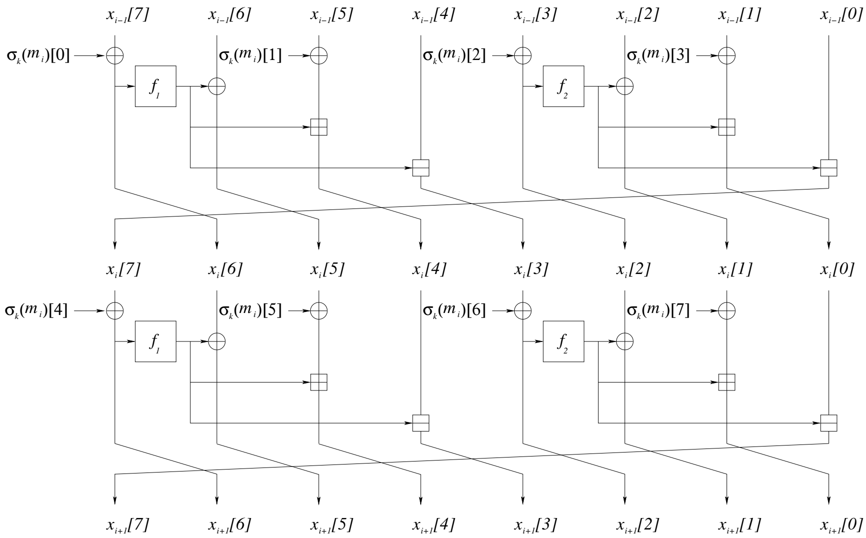

Figure 3.2: F-Function

Algorithm 1 F-function

Input: 512-bit state and 256-bit Message Value (m_i)

Output: 512-bit new state value

Calculate F(x,w) for each branch

for $1 \leq i \leq 16$ **do**

  if *(Round is Odd)* **then**

    $x_i[0] = (x_{i-1}[1] \oplus \sigma_k(m_i)[3]) \boxplus f_2(x_{i-1}[3] \oplus \sigma_k(m_i)[2])$ $x_i[1] = x_{i-1}[2] \oplus f_2(x_{i-1}[3] \oplus \sigma_k(m_i)[2])$ $x_i[2] = x_{i-1}[3] \oplus \sigma_k(m_i)[2]$ $x_i[3] = x_{i-1}[4] \boxminus f_1(x_{i-1}[7] \oplus \sigma_k(m_i)[0])$ $x_i[4] = (x_{i-1}[5] \oplus \sigma_k(m_i)[1]) \boxplus f_1(x_{i-1}[7] \oplus \sigma_k(m_i)[0])$ $x_i[5] = x_{i-1}[6] \oplus f_1(x_{i-1}[7] \oplus \sigma_k(m_i)[0])$ $x_i[6] = x_{i-1}[7] \oplus \sigma_k(m_i)[0]$ $x_i[7] = x_{i-1}[0] \boxminus f_2(x_{i-1}[3] \oplus \sigma_k(m_i)[2])$

  end

  if *(Round is Even)* **then**

    $x_i[0] = (x_{i-1}[1] \oplus \sigma_k(m_i)[7]) \boxplus f_2(x_{i-1}[3] \oplus \sigma_k(m_i)[6])$ $x_i[1] = x_{i-1}[2] \oplus f_2(x_{i-1}[3] \oplus \sigma_k(m_i)[6])$ $x_i[2] = x_{i-1}[3] \oplus \sigma_k(m_i)[6]$ $x_i[3] = x_{i-1}[4] \boxminus f_1(x_{i-1}[7] \oplus \sigma_k(m_i)[4])$ $x_i[4] = (x_{i-1}[5] \oplus \sigma_k(m_i)[5]) \boxplus f_1(x_{i-1}[7] \oplus \sigma_k(m_i)[4])$ $x_i[5] = x_{i-1}[6] \oplus f_1(x_{i-1}[7] \oplus \sigma_k(m_i)[4])$ $x_i[6] = x_{i-1}[7] \oplus \sigma_k(m_i)[4]$ $x_i[7] = x_{i-1}[0] \boxminus f_2(x_{i-1}[3] \oplus \sigma_k(m_i)[6])$

  end

end

---

f_1 and f_2 functions: f-functions can be considered as the main non-linear part of the compression function of Sarmal, since the complex operations are performed here, they are important for diffusion and confusion. Both f_1 and f_2 are typical examples of substitution-permutation network (SPN).

Let $f_1(I)$ and $f_2(I)$ denote the f-functions, where I is the 64-bit input, which can be seen as concatenation of 8-bytes $I = (I[7-0])$. It passes through 8 parallel 8×8-bit S-boxes and the output is multiplied with an MDS matrix M a nd obtained 64-bit output value O. (Details of S-boxes and Matrix M are in available Appendix A). The only difference of f_1 and f_2 is the selection of S-boxes. Figure 3.3 shows f_1 and f_2, respectively and they are described in the following equations:

f_1 function

$$\begin{aligned} I &= (I[7], I[6], \cdots, I[0]) \\ O_{8\times1} &= M_{8\times8} \cdot (S_0[I[7]], S_0[I[6]], \cdots, S_0[I[0]])^T \end{aligned}$$

f_2 function

$$\begin{aligned} I &= (I[7], I[6], \cdots, I[0]) \\ O_{8\times1} &= M_{8\times8} \cdot (S_1[I[7]], S_1[I[6]], \cdots, S_1[I[0]])^T \end{aligned}$$

*T represents the transpose operation

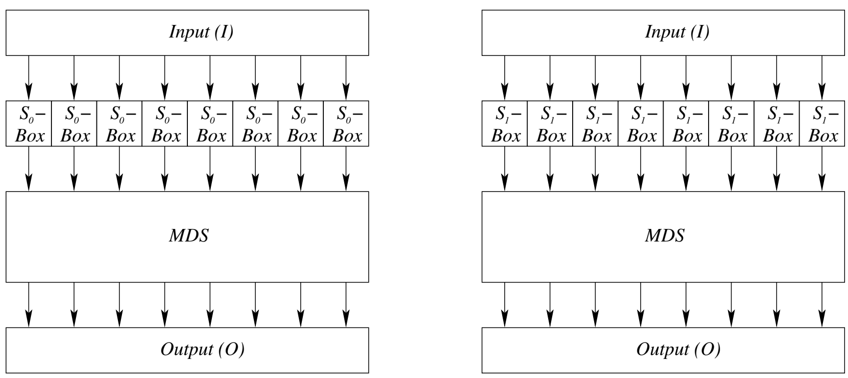

Figure 3.3: f_1 and f_2 functions

3.1.3 384-bit Version of Sarmal

The followings are the only differences between Sarmal-384 and Sarmal-512.

1. The initial value (h_0) and the constant c values are changed in Sarmal-384.
2. 011000000 string is added rather than 100000000 in the padding part.
3. Hash value is obtained by truncating the final hash value to 384-bits in the Sarmal-384. The left-most 384-bits of final hash value is taken hash value of Sarmal-384.(i.e. $H(m) = (h_t[7-2])$).

Initial value of Sarmal-384: Table 3.5 shows the initial value of Sarmal-384. Extension of Golden ratio is used in the IV.

Table 3.5: Initial Value of Sarmal-384

$h_0[0] = 9E3779B97F4A7C15_x$	$h_0[4] = 2767F0B153D27B7F_x$
$h_0[1] = F39CC0605CEDC834_x$	$h_0[5] = 0347045B5BF1827F_x$
$h_0[2] = 1082276BF3A27251_x$	$h_0[6] = 01886F0928403002_x$
$h_0[3] = F86C6A11D0C18E95_x$	$h_0[7] = C1D64BA40F335E36_x$

Constants of Sarmal-384: Constants are taken from the extension of square root of five. These constant values are given in Table 3.6.

Table 3.6: Constants of Sarmal-384

$C[0] = 3C6EF372FE94F82B_x$	$C[2] = 21044ED7E744E4A3_x$
$C[1] = E73980C0B9DB9068_x$	$C[3] = F0D8D423A1831D2A_x$

3.2 Design Rationale

The following goals are taken into consideration while designing the hash function family $Sarmal$.

1. The new design should ensure better security results than $SHA-2$ family.

2. The new design should be analyzed easily.

3. Attacks on MD and SHA family should not work on the new design.

4. The software and hardware performance should be good.

5. The new design should suggest various hash sizes.

The following design rationale is used to achieve these goals.

3.2.1 Structure

Recent attacks show that using Merkle-Damgård construction does not guarantee as much security as expected. New iteration method HAIFA is proposed to patch the weaknesses of MD, and to improve MD construction. Thus, we use HAIFA in our design.

Two independent branches are used in *Sarmal*. $RIPEMD$ [7], $RIPEMD-128/160$ [8] and $FORK$ [66] use similar type of construction. There exist attacks on $RIPEMD$ and $FORK$ due to the flaws in their design. $RIPEMD$ used the same message type in both branches and the weakness was exploited in [19]. It was shown in [67] that the compression function of $FORK$ caused some weaknesses. On the other hand, there is no known attacks to $RIPEMD-128/160$, even though it uses the $MD4$ structure which is badly broken today. Thus, using more than one branch can be secure if it is used correctly.

Therefore, it can be deduced that, using more than one branch can only provide enough security if it is used correctly.

3.2.2 F-function

It is required to handle 512-bit data in the hashing process. Generalized Feistel type block cipher is used in $Sarmal$ to handle that much of data and the required non-linear part of the hash function is reduced to 64-bit f-functions. Addition and subtraction modulo 2^{64} are also used as non-linear parts to diffuse the output of f-function to the branches differently.

3.2.3 f-functions (f_1 and f_2)

A simple SPN is chosen for the design of f-functions. Constructing f-functions from 8 parallel s-boxes and the matrix M, based on a $[16, 8, 9]$ MDS code, enables us to express whole structure with look-up tables and to define a lower bound for the branch number of f-functions.

Intel's new processor (Nehalem) is going to support operations on AES and enable faster implementations of AES-like structures which is an advantage for our design.

Diffusion Layer An 8×8 MDS matrix, which provides good diffusion properties, is used for diffusion. It guarantees that to achieve branch number of at least 9 for the branch number which is described in Definition 3.3.1. This property helps to give the security margins for the hash function Sarmal.

Message Permutation The message permutations are considered as an 8×8 matrix M_p for each branch(In Table 3.2, input values form this defined matrix). Some restriction are put on this matrix. Firs, four conditions are used in order to avoid local collisions. Let $\alpha_{i,j}$ denote the value in i^{th} row and j^{th} column. Then, these conditions are $\alpha_{i,1} \neq \alpha_{i+1,2}$, $\alpha_{i,3} \neq \alpha_{i+1,0}$, $\alpha_{i,5} \neq \alpha_{i+1,6}$, $\alpha_{i,7} \neq \alpha_{i+1,4}$. The reason, is that the same message cancels itself after two rounds, which can be seen in Figure 3.4.

Decided permutations also satisfy that if two of the 64-bit messages are same, then at most two of the above conditions can be satisfied for different i's and j's. Therefore, these two messages keep propagating through each branch.

3.3 Security Analysis

We need some definitions, before discussing the security analysis of Sarmal.

Branch number of a transformation is a helpful tool while calculating the number of active s-boxes of a structure which gives lower attack complexity bound of the cipher against differential cryptanalysis.

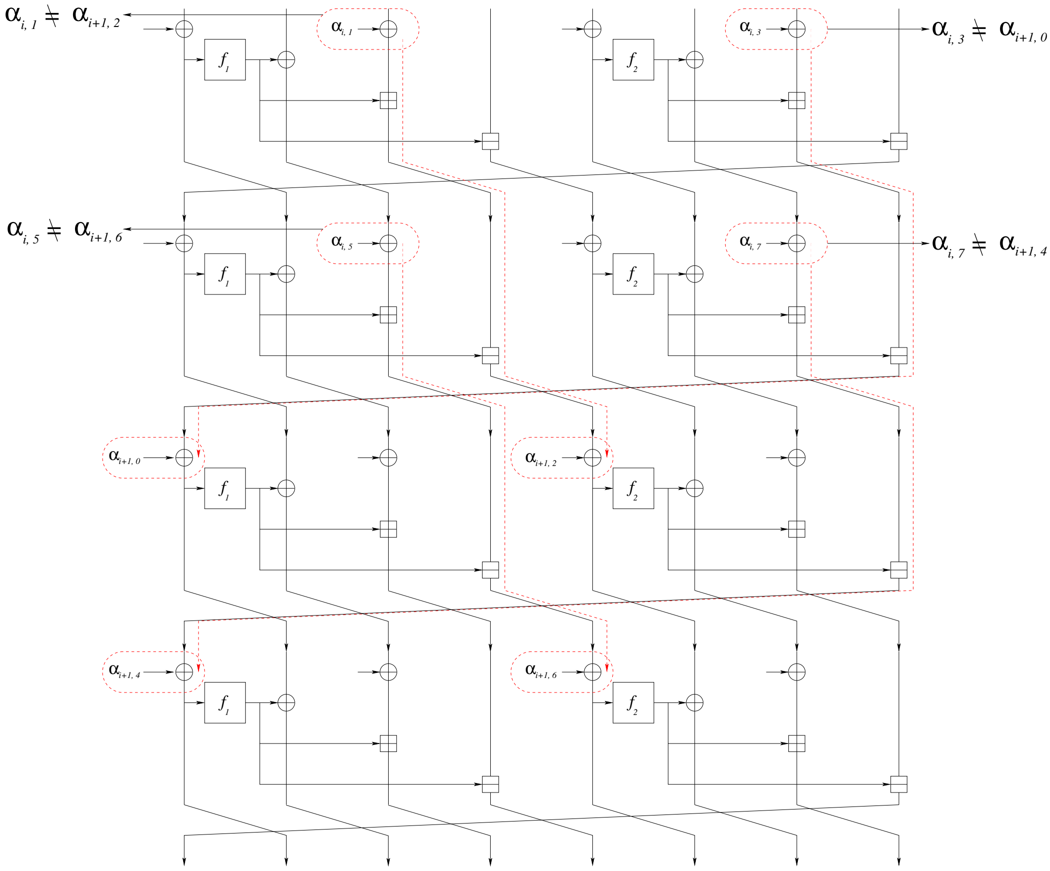

Figure 3.4: Conditions on Message Permutation

Definition 3.3.1 (Branch Number[68]) *Let F be a linear transformation operating on bytes and let W(.) be the byte weight of an input value (i.e. counts the non-zero bytes of the given value). Then, the branch number of F is defined as* $min_{a \neq 0}\{W(a) + W(F(a)\}$..

For a block cipher or a block cipher based construction, one of the important parts is the non-linear layer where s-boxes are mainly used. In each construction, they are used in parallel and more than once. Finding the paths, with the minimum number of s-box passes, until the end of cipher gains an important role, and a lower bound can be given for the attack complexity. This notion is named as active s-box number (ASB).

3.3.1 Collision Resistance

The complexity of a collision attack must be less than $2^{n/2}$ for a n-bit hash function. To show Sarmal's collision resistance, number of active s-boxes were calculated for the worst scenario.

Choosing the worst case scenario, mentioned below, is eased the calculation of active s-boxes and if the results are greater than the expected bound, then the last term attacks do not work on Sarmal. Since, a good differential path with minimum number of active s-boxes is required for the recent attacks.

- Addition and subtraction operations modulo 2^{64} are converted to the XOR operation which eases the active s-box computation. Since, both addition and subtraction operations are non-linear, the original design's result is not going to be worse than the modified version's result, and actually it is expected to see more active s-boxes.
- All of the eight words, entering the round function F, are considered as numbers ranging from 0 to 8 which corresponds to the number of non-zero bytes.
- f-functions gives output ranging from 0 to 8 that depends on the input value. f-functions use MDS matrices which guarantee that the total number of non-zero bytes for the input and output values are greater than 9.
- Two differences cancels each other in XOR operation only if they are same. Therefore, if the two values entering the XOR operation are the same, then the resulting value is taken as zero regardless of the actual values and positions of the bytes, in order to create the worst case scenario.

- If the two values entering the XOR operation are different, then the resulting value is taken as the difference between them which means that the non-zero bytes of two values are in the same positions with the same byte-values, in order to form the worst case scenario.

- A non-zero input difference is given to only one of the eight messages or one of the eight branches in the F or these are given at the same time. A software program is developed and results showed that the required attack complexity exceeds before reaching the 16^{th} round of Sarmal. The best results are given in Table 3.7.

Table 3.7: Active S-box Number

Round Number	ASB (Left)	ASB (Right)	ASB (Total)	Non-zero values
15	48	60	108	$M[1] = 6$
14	54	57	111	$M[0] = 6$
14	57	63	120	$M[4] = 6$
14	48	66	114	$M[5] = 6$
14	57	57	114	$M[7] = 6$
12	61	59	120	$M[0] = 8$ $X[4] = 7$
11	63	63	126	$X[2] = 7$

The maximum value of the negative log_2 in the XOR-table is 2^{-5}. The required attack complexity for differential cryptanalysis is $2^{(5 \times ASB)}$ which is greater than 2^{512}. Thus, the time complexity for differential attacks on full round Sarmal exceeds exhaustive search.

3.3.2 Resistance Against Known Attacks

Recent attacks on Merkle-Damgård construction is by passed by HAIFA. Length extension attacks (See Section 2.3.1) are prevented by choosing t-value which includes information about the number of bits hashed so far. Second preimage attack based on fix points or multi-collisions are prevented by using t-value which was stated in the [28] as adding block index into the compression function. This idea was used more practically and number of bits is kept instead of block index. More data storage is going to be needed to perform the herding attack due to existence of salt value (if salt is not fixed).

3.4 Implementation Issues

Performance of Sarmal-512 is going to be compared with $SHA-512$ in this section. Two of the fastest $SHA-512$ implementations, implemented by Dai [69] and Gladman [70], are taken into consideration and compared with optimized Sarmal-512 code. Gladman's code is an optimized implementation of $SHA-512$ without using any assembly or Streaming SIMD Extensions 2 ($SSE2$) structures. Other one uses these structures in the implementation of $SHA-512$. An optimized code is also used with assembly language in the implementation of Sarmal-512. It can be improved by using $SSE2$ structures. The tests are performed in the system:

Computer	:	Intel Core 2 Duo (2.00 GHz, 4 MB L2 Cache),
	:	2GB DDR2 667 MHz RAM
Operating System	:	Ubuntu 8.04.1 64-bit
Compiler	:	GNU C Compiler (GCC) 4.2.3

Table 3.8: Performance of Sarmal-512 and $SHA-512$

Algorithm	Cycles/Byte
Sarmal-512	14.4
$SHA-512$ [69]	12.1
$SHA-512$ [70]	31.1

The number of cycles is measured and small numbers represents faster implementations in the Table 3.8. The results show that using assembly and $SSE2$ structures increase the performances of the hash functions. Therefore, we expect better results for Sarmal after combining the implementations of these two structures.

CHAPTER 4

Conclusion

In this thesis, starting from the very definition of cryptographic hash functions, the design methods together with their weaknesses and strengths are described. All the design and analysis methods that have been speeding up since a couple of years are considered and based on the output of those scientific effort, a new family of hash function is introduced.

Our aim was to construct a cryptographic hash function which is more secure and faster than $SHA-2$, as it serves as a model after the continuous attacks to the well known hash functions. To achieve these goals, the previous experiences on block ciphers are used and a block cipher based construction which is suitable to our constraints is figured as Sarmal. In Sarmal, we use HAIFA construction as we believe it is more suitable for our purposes especially in terms of security and flexibility.

We plan to submit this design to the NIST's competition on designing a new cryptographic hash function which will be standardized and called $SHA-3$ as a new generation hashing standard.

As a future work, we are planning to do the followings for our design:

- The design of a new and more hardware efficient s-boxes .
- Efficient implementation of our design by bit slice implementation and SSE2 techniques.
- The hardware optimization and design of Sarmal which is resistant against Side Channel Attacks
- The security evaluations based on the latest attacks which will emerge before submis-

sion

- Efficient software implementations of Sarmal which are suitable for various architectures.

REFERENCES

[1] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.

[2] Ronald L. Rivest. The MD5 message-digest algorithm, 1992. URL: `http://theory.lcs.mit.edu/ rivest/rfc1321.txt`. Note: Request For Comments 1321.

[3] *Secure Hash Standard*. National Institute of Standards and Technology, Washington, 2002. URL: `http://csrc.nist.gov/publications/fips/`. Note: Federal Information Processing Standard 180-2.

[4] Burt Kaliski. The MD2 Message-Digest Algorithm. In *RFC 1319*, 1992.

[5] Burt Kaliski. The MD2 Message-Digest Algorithm. In *RFC 1319*, 1992.

[6] Ronald L. Rivest. The MD4 Message-digest Algorithm, 1991.

[7] Research and Development in Advanced Communication Technologies in Europe. RIPE Integrity Primitives: Final Report of RACE Integriy Primitives Evaluation. In *RACE*, 1992.

[8] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A Strengthened Version of RIPEMD. In Gollmann [71], pages 71–82.

[9] Ross J. Anderson and Eli Biham. TIGER: A Fast New Hash Function. In Gollmann [71], pages 89–97.

[10] Joan Daemen and Craig S. K. Clapp. Fast Hashing and Stream Encryption with PANAMA. In Serge Vaudenay, editor, *FSE*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 1998.

[11] Guido Bertoni and Joan Daemen and Gilles VanAssche and Michaël Peeters. Radio-Gatún, A Belt and Mill Hash Function, 2007.

[12] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions MD4, MD5, HAVAL–128 and RIPEMD, 2004. URL: `http://eprint.iacr.org/2004/199/`.

[13] Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, pages 56–71, London, UK, 1998. Springer-Verlag.

[14] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Cramer [72], pages 19–35.

[15] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In Cramer [72], pages 36–57.

[16] National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA3) Family. 2007. http://csrc.nist.gov/groups/ST/hash/index.html.

[17] Bart Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Electrical Engineering Department, Katholieke Universiteit Leuven, 1993.

[18] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *CRYPTO*, pages 2–21, 1990.

[19] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer [72], pages 1–18.

[20] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In *CRYPTO*, pages 1–16, 2005.

[21] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In *CRYPTO*, pages 17–36, 2005.

[22] Yu Sasaki, Lei Wang, Kazuo Ohta, and Noboru Kunihiro. New Message Difference for MD4. In Biryukov [73], pages 329–348.

[23] Ralph C. Merkle. One Way Hash Functions and DES . In Brassard [74], pages 428–446.

[24] Ivan Damgård. A Design Principle for Hash Functions. In Brassard [74], pages 416–427.

[25] Magnus Daum. *Cryptanalysis of Hash Functions of the MD4-Family*. PhD thesis, Ruhr Universität Bochum, 2005.

[26] Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.

[27] Richared D. Dean. *Formal Aspects of Mobile Code Security.* PhD thesis, Princeton University, 1999.

[28] John Kelsey and Bruce Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In Cramer [72], pages 474–490.

[29] John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In Vaudenay [75], pages 183–200.

[30] Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. http://eprint.iacr.org.

[31] J.K. Gibson. Discrete logarithm hash function that is collision free and one way. *Computers and Digital Techniques, IEE Proceedings*, 138(6):407–410, Nov 1991.

[32] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental Cryptography: The Case of Hashing and Signing. In Desmedt [76], pages 216–233.

[33] Scott Contini, Arjen K. Lenstra, and Ron Steinfeld. VSH, an Efficient and Provable Collision-Resistant Hash Function. In Vaudenay [75], pages 165–182.

[34] Denis X. Charles, Kristin E. Lauter, and Eyal Z. Goren. Cryptographic Hash Functions from Expander Graphs. In *Journal of Cryptology*, 2007.

[35] Jean-Pierre Tillich and Gilles Zémor. Group-theoretic hash functions. In Gérard D. Cohen, Simon Litsyn, Antoine Lobstein, and Gilles Zémor, editors, *Algebraic Coding*, volume 781 of *Lecture Notes in Computer Science*, pages 90–110. Springer, 1993.

[36] Gilles Zémor. Hash Functions and Cayley Graphs. *Des. Codes Cryptography*, 4(4):381–394, 1994.

[37] Jean-Pierre Tillich and Gilles Zémor. Hashing with SL_2. In Desmedt [76], pages 40–49.

[38] Miklós Ajtai. Generating Hard Instances of the Short Basis Problem. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP*, volume 1644 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 1999.

[39] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Collision-Free Hashing from Lattice Problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(42), 1996.

[40] Daniele Micciancio. Improved cryptographic hash functions with worst-case/average-case connection. In *STOC*, pages 609–618, 2002.

[41] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. Provably Secure FFT Hashing. NIST 2nd Cryptogaphic Hash Workshop, 2006. Available on-line at URL `http://www.csrc.nist.gov/pki/HashWorkshop/2006/program_2006.htm`.

[42] D. Davies and W.L. Price. Digital signatures, an update. In *5th International Conference on Computer Communication*, pages 845–849, 1994.

[43] S.M. Matyas, C.H. Meyer, and J. Oseas. Generating strong one-way functions with crypographic algorithm. In *IBM Technical Disclosure Bulletin, 27(10A):5658-5659, 1985*, 1985.

[44] S. Miyaguchi, M. Iwata, and K. Ohta. New 128-bit hash function. In *Proceeding of 4th International Joint Workshop on Computer Communications*, pages 279–288, 1989.

[45] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In Stinson [77], pages 368–378.

[46] Martijn Stam. Another Glance At Blockcipher Based Hashing. Cryptology ePrint Archive, Report 2008/071, 2008. http://eprint.iacr.org/.

[47] B. Brachtl, D. Coppersmith, M. Hyden, S. Matayas, C. Meyer, J. Oseas, S. Pilpel, and M. Schiling. Data Authentication Using Modification Detection Codes Based on a Public One Way Encryption Function. In *U. S. Patent Number 4,908,861*, 1990.

[48] National Institute of Standards and Technology. Data Encrypticon Standart (DES). In *Federal Information Processing Standards*, 1999.

[49] Mridul Nandi, Wonil Lee, Kouichi Sakurai, and Sangjin Lee. Security Analysis of a 2/3-Rate Double Length Compression Function in the Black-Box Model. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 243–254. Springer, 2005.

[50] Xuejia Lai and James L. Massey. Hash Function Based on Block Ciphers. In *EUROCRYPT*, pages 55–70, 1992.

[51] Walter Hohl, Xuejia Lai, Thomas Meier, and Christian Waldvogel. Security of Iterated Hash Functions Based on Block Ciphers. In Stinson [77], pages 379–390.

[52] Shoichi Hirose. Provably Secure Double-Block-Length Hash Functions in a Black-Box Model. In Choonsik Park and Seongtaek Chee, editors, *ICISC*, volume 3506 of *Lecture Notes in Computer Science*, pages 330–342. Springer, 2004.

[53] Bart Preneel and Antoon Bosselaers and René Govaerts and Joos Vandewalle. Collision-free Hash Functions Based on Block Cipher Algorithms. In *International Carnahan Conference on Security Technology*, pages 203–210, 1989.

[54] Lars R. Knudsen, Xuejia Lai, and Bart Preneel. Attacks on Fast Double Block Length Hash Functions. *J. Cryptology*, 11(1):59–72, 1998.

[55] T. Satoh, M. Haga, and K. Kurosawa. Towards secure and fast hash functions. In *IEICE Transactions on Fundementals*, pages 55–62, 1999.

[56] Mitsuhiro Hattori, Shoichi Hirose, and Susumu Yoshida. Analysis of Double Block Length Hash Functions. In Kenneth G. Paterson, editor, *IMA Int. Conf.*, volume 2898 of *Lecture Notes in Computer Science*, pages 290–302. Springer, 2003.

[57] Kwangjo Kim and Tsutomu Matsumoto, editors. *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*, volume 1163 of *Lecture Notes in Computer Science*. Springer, 1996.

[58] Guido Bertoni and Joan Daemen and Michaël Peeters and Gilles Van Assche. Sponge Functions. ECRYPT Hash Worksop, 2007.

[59] Lars R. Knudsen, Christian Rechberger, and Søren S. Thomsen. The Grindahl Hash Functions. In Biryukov [73], pages 39–57.

[60] Thomas Peyrin. Cryptanalysis of Grindahl. In *ASIACRYPT*, pages 551–567, 2007.

[61] Michael Gorski, Stefan Lucks, and Thomas Peyrin. Slide Attacks on Hash Functions. Cryptology ePrint Archive, Report 2008/263, 2008. http://eprint.iacr.org/.

[62] Vincent Rijmen, Bart Van Rompay, Bart Preneel, and Joos Vandewalle. Producing Collisions for PANAMA. In Mitsuru Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2001.

[63] Joan Daemen and Gilles Van Assche. Producing Collisions for Panama, Instantaneously. In Biryukov [73], pages 1–18.

[64] Donghoon Chang, Kishan Chand Gupta, and Mridul Nandi. RC4-Hash: A New Hash Function Based on RC4. In *INDOCRYPT*, pages 80–94, 2006.

[65] Sebastiaan Indesteege and Bart Preneel. Collisions for RC4-Hash. ISC, 2008.

[66] Deukjo Hong, Donghoon Chang, Jaechul Sung, Sangjin Lee, Seokhie Hong, Jaesang Lee, Dukjae Moon, and Sungtaek Chee. A New Dedicated 256-Bit Hash Function: FORK-256. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 2006.

[67] Krystian Matusiewicz, Thomas Peyrin, Olivier Billet, Scott Contini, and Josef Pieprzyk. Cryptanalysis of fork-256. In Biryukov [73], pages 19–38.

[68] J. Daemen and V. Rijmen. The Design of Rijndael: AES - The Advanced Encryption Standard. In *Springer*, 2002.

[69] Wei Dai. Crypto++ Library. http://www.cryptopp.com.

[70] Brian Gladman. http://fp.gladman.plus.com/cryptography_technology/sha/index.htm.

[71] Dieter Gollmann, editor. *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*. Springer, 1996.

[72] Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

[73] Alex Biryukov, editor. *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*. Springer, 2007.

[74] Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.

[75] Serge Vaudenay, editor. *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*. Springer, 2006.

[76] Yvo Desmedt, editor. *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*. Springer, 1994.

[77] Douglas R. Stinson, editor. *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*. Springer, 1994.

[78] Joan Daemen and Vincent Rijmen. Rijndael/AES. In Henk C. A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*. Springer, 2005.

[79] Paulo S. L. M. Barreto and Vincent Rijmen. The Whirlpool Hashing Function. In *First open NESSIE Workshop*, 2000.

APPENDIX A

S-boxes and MDS Matrices

A.1 S-Boxes

The S-boxes of Sarmal are taken from AES[78] and Whirlpool[79]. Both have nice cryptographic properties. The largest values in XOR table and Linear Approximation Table (LAT) are minimized. Hardware compatibility is also considered in the designs of S-boxes.

Table A.1: S_0-Box

	00_x	01_x	02_x	03_x	04_x	05_x	06_x	07_x	08_x	09_x	$0A_x$	$0B_x$	$0C_x$	$0D_x$	$0E_x$	$0F_x$
00_x	18_x	23_x	$C6_x$	$E8_x$	87_x	$B8_x$	01_x	$4F_x$	36_x	$A6_x$	$D2_x$	$F5_x$	79_x	$6F_x$	91_x	52_x
10_x	60_x	BC_x	$9B_x$	$8E_x$	$A3_x$	$0C_x$	$7B_x$	35_x	$1D_x$	$E0_x$	$D7_x$	$C2_x$	$2E_x$	$4B_x$	FE_x	57_x
20_x	15_x	77_x	37_x	$E5_x$	$9F_x$	$F0_x$	$4A_x$	DA_x	58_x	$C9_x$	29_x	$0A_x$	$B1_x$	$A0_x$	$6B_x$	85_x
30_x	BD_x	$5D_x$	10_x	$F4_x$	CB_x	$3E_x$	05_x	67_x	$E4_x$	27_x	41_x	$8B_x$	$A7_x$	$7D_x$	95_x	$D8_x$
40_x	FB_x	EE_x	$7C_x$	66_x	DD_x	17_x	47_x	$9E_x$	CA_x	$2D_x$	BF_x	07_x	AD_x	$5A_x$	83_x	33_x
50_x	63_x	02_x	AA_x	71_x	$C8_x$	19_x	49_x	$D9_x$	$F2_x$	$E3_x$	$5B_x$	88_x	$9A_x$	26_x	32_x	$B0_x$
60_x	$E9_x$	$0F_x$	$D5_x$	80_x	BE_x	CD_x	34_x	48_x	FF_x	$7A_x$	90_x	$5F_x$	20_x	68_x	$1A_x$	AE_x
70_x	$B4_x$	54_x	93_x	22_x	64_x	$F1_x$	73_x	12_x	40_x	08_x	$C3_x$	EC_x	DB_x	$A1_x$	$8D_x$	$3D_x$
80_x	97_x	00_x	CF_x	$2B_x$	76_x	82_x	$D6_x$	$1B_x$	$B5_x$	AF_x	$6A_x$	50_x	45_x	$F3_x$	30_x	EF_x
90_x	$3F_x$	55_x	$A2_x$	EA_x	65_x	BA_x	$2F_x$	$C0_x$	DE_x	$1C_x$	FD_x	$4D_x$	92_x	75_x	06_x	$8A_x$
$A0_x$	$B2_x$	$E6_x$	$0E_x$	$1F_x$	62_x	$D4_x$	$A8_x$	96_x	$F9_x$	$C5_x$	25_x	59_x	84_x	72_x	39_x	$4C_x$
$B0_x$	$5E_x$	78_x	38_x	$8C_x$	$D1_x$	$A5_x$	$E2_x$	61_x	$B3_x$	21_x	$9C_x$	$1E_x$	43_x	$C7_x$	FC_x	04_x
$C0_x$	51_x	99_x	$6D_x$	$0D_x$	FA_x	DF_x	$7E_x$	24_x	$3B_x$	AB_x	CE_x	11_x	$8F_x$	$4E_x$	$B7_x$	EB_x
$D0_x$	$3C_x$	81_x	94_x	$F7_x$	$B9_x$	13_x	$2C_x$	$D3_x$	$E7_x$	$6E_x$	$C4_x$	03_x	56_x	44_x	$7F_x$	$A9_x$
$E0_x$	$2A_x$	BB_x	$C1_x$	53_x	DC_x	$0B_x$	$9D_x$	$6C_x$	31_x	74_x	$F6_x$	46_x	AC_x	89_x	14_x	$E1_x$
$F0_x$	16_x	$3A_x$	69_x	09_x	70_x	$B6_x$	$D0_x$	ED_x	CC_x	42_x	98_x	$A4_x$	28_x	$5C_x$	$F8_x$	86_x

Table A.2: S_1-Box

	00_x	01_x	02_x	03_x	04_x	05_x	06_x	07_x	08_x	09_x	$0A_x$	$0B_x$	$0C_x$	$0D_x$	$0E_x$	$0F_x$
00_x	63_x	$7C_x$	77_x	$7B_x$	$F2_x$	$6B_x$	$6F_x$	$C5_x$	30_x	01_x	67_x	$2B_x$	FE_x	$D7_x$	AB_x	76_x
10_x	CA_x	82_x	$C9_x$	$7D_x$	FA_x	59_x	47_x	$F0_x$	AD_x	$D4_x$	$A2_x$	AF_x	$9C_x$	$A4_x$	72_x	$C0_x$
20_x	$B7_x$	FD_x	93_x	26_x	36_x	$3F_x$	$F7_x$	CC_x	34_x	$A5_x$	$E5_x$	$F1_x$	71_x	$D8_x$	31_x	15_x
30_x	04_x	$C7_x$	23_x	$C3_x$	18_x	96_x	05_x	$9A_x$	07_x	12_x	80_x	$E2_x$	EB_x	27_x	$B2_x$	75_x
40_x	09_x	83_x	$2C_x$	$1A_x$	$1B_x$	$6E_x$	$5A_x$	$A0_x$	52_x	$3B_x$	$D6_x$	$B3_x$	29_x	$E3_x$	$2F_x$	84_x
50_x	53_x	$D1_x$	00_x	ED_x	20_x	FC_x	$B1_x$	$5B_x$	$6A_x$	CB_x	BE_x	39_x	$4A_x$	$4C_x$	58_x	CF_x
60_x	$D0_x$	EF_x	AA_x	FB_x	43_x	$4D_x$	33_x	85_x	45_x	$F9_x$	02_x	$7F_x$	50_x	$3C_x$	$9F_x$	$A8_x$
70_x	51_x	$A3_x$	40_x	$8F_x$	92_x	$9D_x$	38_x	$F5_x$	BC_x	$B6_x$	DA_x	21_x	10_x	FF_x	$F3_x$	$D2_x$
80_x	CD_x	$0C_x$	13_x	EC_x	$5F_x$	97_x	44_x	17_x	$C4_x$	$A7_x$	$7E_x$	$3D_x$	64_x	$5D_x$	19_x	73_x
90_x	60_x	81_x	$4F_x$	DC_x	22_x	$2A_x$	90_x	88_x	46_x	EE_x	$B8_x$	14_x	DE_x	$5E_x$	$0B_x$	DB_x
$A0_x$	$E0_x$	32_x	$3A_x$	$0A_x$	49_x	06_x	24_x	$5C_x$	$C2_x$	$D3_x$	AC_x	62_x	91_x	95_x	$E4_x$	79_x
$B0_x$	$E7_x$	$C8_x$	37_x	$6D_x$	$8D_x$	$D5_x$	$4E_x$	$A9_x$	$6C_x$	56_x	$F4_x$	EA_x	65_x	$7A_x$	AE_x	08_x
$C0_x$	BA_x	78_x	25_x	$2E_x$	$1C_x$	$A6_x$	$B4_x$	$C6_x$	$E8_x$	DD_x	74_x	$1F_x$	$4B_x$	BD_x	$8B_x$	$8A_x$
$D0_x$	70_x	$3E_x$	$B5_x$	66_x	48_x	03_x	$F6_x$	$0E_x$	61_x	35_x	57_x	$B9_x$	86_x	$C1_x$	$1D_x$	$9E_x$
$E0_x$	$E1_x$	$F8_x$	98_x	11_x	69_x	$D9_x$	$8E_x$	94_x	$9B_x$	$1E_x$	87_x	$E9_x$	CE_x	55_x	28_x	DF_x
$F0_x$	$8C_x$	$A1_x$	89_x	$0D_x$	BF_x	$E6_x$	42_x	68_x	41_x	99_x	$2D_x$	$0F_x$	$B0_x$	54_x	BB_x	16_x

A.2 MDS Matrices

A.2.1 Sarmal-384/512

The matrix M, used in f-functions, is a linear mapping based on a $[16, 8, 9]$ MDS code and defined from $GF(2^8)$ to $GF(2^8)$. The field $GF(2^8)$ is given as $GF(2)[x]/p(x)$ where $p(x) = x^8 + x^4 + x^3 + x^2 + 1$ and $p(x)$ is a primitive polynomial. Each row in matrix M and input value I are considered as polynomials in $GF(2^8)$ and multiplied to find output value which is described as $O_{8\times1} = M_{8\times8} \cdot I_{8\times1}$.

$$M = \begin{bmatrix} 01_x & 09_x & 02_x & 05_x & 08_x & 01_x & 04_x & 01_x \\ 01_x & 01_x & 09_x & 02_x & 05_x & 08_x & 01_x & 04_x \\ 04_x & 01_x & 01_x & 09_x & 02_x & 05_x & 08_x & 01_x \\ 01_x & 04_x & 01_x & 01_x & 09_x & 02_x & 05_x & 08_x \\ 08_x & 01_x & 04_x & 01_x & 01_x & 09_x & 02_x & 05_x \\ 05_x & 08_x & 01_x & 04_x & 01_x & 01_x & 09_x & 02_x \\ 02_x & 05_x & 08_x & 01_x & 04_x & 01_x & 01_x & 09_x \\ 09_x & 02_x & 05_x & 08_x & 01_x & 04_x & 01_x & 01_x \end{bmatrix}$$

A.2.2 Sarmal-224/256

The linear mapping, used in Sarmal-224/256, is based on a $[8, 4, 5]$ MDS code.and taken from AES. It is defined from $GF(2^8)$ to $GF(2^8)$. The field $GF(2^8)$ is given as $GF(2)[x]/p(x)$ where $p(x) = x^8 + x^4 + x^3 + x^2 + 1$ and $p(x)$ is a primitive polynomial. Each row in matrix M and input value I are considered as polynomials in $GF(2^8)$ and multiplied to find output value which is described as $O_{4\times1} = M_{4\times4} \cdot I_{4\times1}$.

$$M = \begin{bmatrix} 02_x & 03_x & 01_x & 01_x \\ 01_x & 02_x & 03_x & 01_x \\ 01_x & 01_x & 02_x & 03_x \\ 03_x & 01_x & 01_x & 02_x \end{bmatrix}$$

APPENDIX B

Sarmal-256/224

B.1 Description of **Sarmal**-256

B.1.1 Notation

Throughout this chapter, the following notation will be used. Each 256-bit *block* is composed of eight 32-bit *words* $(X[8], X[7], \ldots, X[0] = X[7-0])$. Note that the words and blocks are in little-endian order (i.e. the least significant bit is the rightmost bit numbered 0, and the most significant bit is bit 31 for a 32-bit word.). The symbols that are used in the equations and figures are given in the Table B.1.

Padding: A single bit 1 is concatenated to the message M, followed by zeros until the length of the message is 183 modulo 256. Afterwards, 010000000 is added at the end and the 64-bit original message length is appended to the end.

Table B.1: Notation

$\oplus$	Bitwise logical exclusive OR (XOR)	s_{i-1}	64-bit salt value Â
$\boxplus$	Addition modulo 2^{32}	$s_{i-1}[j]$	j^{th} 32-bit word of 64-bit s_{i-1}
$\boxminus$	Substraction modulo 2^{32}	t_{i-1}	32-bit bit counter
X_i	256-bit intermediate value	c	128-bit constant value
$X_i[j]$	j^{th} 32-bit word of 256-bit X_i	$c[j]$	j^{th} 32-bit word of 128-bit c
h_{i-1}	256-bit chaining value	I	32-bit Input value
$h_{i-1}[j]$	j^{th} 32-bit word of 256-bit h_{i-1}	O	32-bit Output value
$S_i[.]$	8×8-bit S-box transformation	$M_{8\times8}$	4×4 MDS Matrix

Initial value: Between the 129^{th} hexadecimal digit to 192^{th} hexadecimal digit of π is used as the initial value (i.e. IV or h_0) of Sarmal-256, and it is given in Table B.2.

Table B.2: Initial Value of Sarmal-256

$h_0[0] = 9216D5D9_x$	$h_0[4] = 2FFD72DB_x$
$h_0[1] = 8979FB1B_x$	$h_0[5] = D01ADFB7_x$
$h_0[2] = D1310BA6_x$	$h_0[6] = B8E1AFED_x$
$h_0[3] = 98DFB5AC_x$	$h_0[7] = 6A267E96_x$

s **and** *t* **values:** 64-bit s is used like a counter. It is initialized with an IV and incremented by one after each compression function calls and t shows the number of hashed bits so far. It is 64-bit and $t[0]$ (rightmost part) used in the right branch and $t[1]$ used in the left branch.

Constants: Sarmal uses a 128-bit constant, C, which is divided into four 32-bit words. They are taken from the extension of square root of three from 65^{th} hexadecimal digit to 96^{th} hexadecimal digit. These values are given in Table B.3.

Table B.3: Constants of Sarmal-256

$C[0] = 490BCFD9_x$	$C[2] = A9930AAE_x$
$C[1] = 5EF15DBD_x$	$C[3] = 12228F87_x$

B.1.2 Sarmal-256 Algorithm

The structure of Sarmal is preserved in the construction of Sarmal-224/256. On the other hand, the size of the variables are halved. Sarmal accepts again a message m of arbitrary length (no more than $2^{64} - 1$ bits) as input. But outputs a 256-bit hash value $H(m)$. It uses 256-bit compression function $f(h_{i-1}, m_i, t_{i-1}, s_{i-1})$. In each iteration, the rightmost four words of h_{i-1} is concatenated with the rightmost word of the salt s_{i-1}, rightmost two words of the constant c, and rightmost word of the t value. The 256-bit state is iterated 16 rounds in the right branch. $(X_0 = (X_0[7-0]) = (h_{i-1}[3-0], s_{i-1}[0], c[1-0], t[1]))$

Similarly, the leftmost four words of h_{i-1} is concatenated with the leftmost word of the salt value s_{i-1}, leftmost two words of the constant c, and and leftmost word of the t and the result is again processed for 16 rounds in the left branch. ($X_0 = (X_0[7-0]) = (h_{i-1}[7-4], s_{i-1}[1], c[3-2], t[0])$)

The two outputs are XORed, and the resulting value is XORed with m_i.

Round Function: The round function $F(x, w)$ uses 256-bit x and 128-bit w inputs this time, and w is obtained from m_i by a permutation σ_k. For odd rounds w is the least significant four words of the given permutation, whereas for even rounds it is the most significant four words (i.e. $w = \sigma_k(m_i)[0-3]$ or $w = \sigma_k(m_i)[4-7]$).

f_1 and f_2 functions: $f_1(I)$ and $f_2(I)$ take 32-bit input I , which can be seen as concatenation of 8-bytes $I = (I[3-0])$. It passes through 4 parallel 8×8 S-boxes and the output is multiplied with an MDS matrix M. (The same s-boxes are used in Sarmal-224/256 but the matrix M is changed to 4×4 MDS matrix and details are available in Appendix A).

B.1.3 224-bit Version of Sarmal

The following are the only differences between Sarmal-224 and Sarmal-256.

1. The initial value (h_0) and the constant c are changed in Sarmal-384.
2. 001100000 string is added rather than 010000000 in the padding part.
3. Hash value is obtained by truncating the final hash value to 224-bits in the Sarmal-224, i.e., the left-most 224-bits of final hash value is taken hash value of Sarmal-224 ($H(m) = (h_t[7-2])$).

Initial Value of Sarmal-224: Table B.4 shows the initial value of Sarmal-224. Extension of Golden ratio from 129^{th} hexadecimal digit to 192^{th} hexadecimal digit is used in the IV.

Table B.4: Initial Value of Sarmal-224

$h_0[0] = 85839D6E_x$	$h_0[4] = CADD0CCC_x$
$h_0[1] = FFBD7DC6_x$	$h_0[5] = FDFFBBE1_x$
$h_0[2] = 64D325D1_x$	$h_0[6] = 626E33B8_x$
$h_0[3] = C5371682_x$	$h_0[7] = D04B4331_x$

Constants of Sarmal-224: Constants are taken from the extension of square root of five from 65^{th} hexadecimal digit to 96^{th} hexadecimal digit . These constant values are given in Table B.5.

Table B.5: Constants of Sarmal-224

$C[0] = 4ECFE162_x$	$C[2] = 068E08B6_x$
$C[1] = A7A4F6FE_x$	$C[3] = B7E304FE_x$