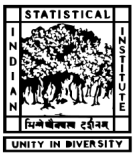

Indian Statistical Institute, Kolkata

Master's Thesis

Cryptanalysis of Iterated Hash and Its Variants

Author:
Ashwin Jha

Supervisor:
Dr. Mridul Nandi

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Technology

in

Computer Science

July 2015

Certificate of Authorship

This is to certify that the thesis entitled *Cryptanalysis of Iterated Hash and Its Variants*, submitted by *Ashwin Jha (roll number: CS1328)*, for the award of the degree of *Master of Technology in Computer Science*, is a record of an original research work carried out by him under my supervision and guidance. I confirm that:

- This work was done wholly or mainly while in candidature for a master's degree at this institution.
- Where the published work of others has been consulted, it has been clearly stated and duly attributed.
- The report fulfils all the requirements, as per the regulations of this institution.

Date:

Place:

Dr. Mridul Nandi
Applied Statistics Unit
ISI, Kolkata

Declaration of Authorship

I, *Ashwin Jha (roll number: CS1328)*, declare that this thesis titled, *Cryptanalysis of Iterated Hash and Its Variants* and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master's degree at this institution.
- Where I have consulted the published work of others, this is always clearly stated and duly attributed.
- I have acknowledged all main sources of help.

Date: Ashwin Jha

Place: CS1328

"It is not knowledge, but the act of learning, not possession but the act of getting there, which grants the greatest enjoyment."

Carl Friedrich Gauss

Abstract

Cryptanalysis of Iterated Hash and Its Variants

by Ashwin Jha

The subject of this thesis is the cryptanalysis of iterated hash function and its variants. In particular, we analyse these designs on three security properties of hash functions, namely, preimage resistance, second preimage resistance and herding attack resistance.

The iterated hash design is the most popular hash function design. Naturally, it is the most studied design as well. In this thesis, we try to do a comprehensive survey of attacks under random oracle model, on the aforementioned security goals of the iterated hash design and its variants.

Structures play a major role in constructing attacks on the iterated hash functions. Here, we are proposing three new structures, viz., *chain*, *multi-pipe expandable message set* and *rho structure*. The chain structure is used to reduce the complexity of herding attack on iterated hash from $O(2^{2n/3})$ to $O(2^{n/2})$. The multi-pipe expandable message set and rho structure are used in our analysis of the zipper hash. We also analyse the security of concatenated hash under cetain weakness assumptions on the underlying compression functions.

Acknowledgements

First and foremost, I would like to express my sincere gratitude towards Dr. Mridul Nandi for supervising the thesis. I would like to thank him for his constant encouragement, ideas and advice throughout the thesis period. I am grateful to him for working with me on all the problems considered in this thesis. Needless to say, the results presented in this thesis are the outcome of our joint work. I am deeply indebted to Dr. Nandi for all the academic and non-academic advice and support. I feel privileged to be in acquaintance with such a great intellect.

Second, I would like to thank the mid-term examiner of my thesis, Dr. Kishan Chand Gupta from our institution for his critique and advice that allowed me to proceed in the right direction. I would also like to thank the research scholars from ASU, particularly Avik da, who helped me in many ways. His intuitive argument on the construction of diamond structure helped me immensely. I had a great time with my friends from the M. Tech. batch at the institution. I cherish my discussions with Amit and Durgesh on various topics.

Last, but not the least, I would like to extend my heartiest gratitude towards my family for their constant support and encouragement. I would like to acknowledge the motivation provided by my mother Mrs. Bandana Jha, in pursuing my interests. I would also like to thank my brother Himanshu Jha, who always instilled confidence in me.

Contents

List of Figures

List of Tables

Dedicated to my family.

Chapter 1

Hash Functions - An Introduction

Historically, the term cryptography has been associated with the study and design of techniques for secure communication in the presence of third parties called adversaries or eavesdroppers. In 1976, a seminal paper titled *New Directions in Cryptography* [6] by Diffie and Hellman identified the requirement of data integrity, authentication and non-repudiation in cryptographic protocols. Modern cryptography has incorporated all these elements along with the traditional need for data confidentiality. Cryptographic hash function is an important primitive in modern cryptography which has a wide array of use in achieving information security goals like integrity, authentication, confidentiality etc.

In this thesis, we study the *cryptanalysis* (cryptographic analysis for weaknesses) of some cryptographic hash function constructions. Cryptographic hash function is an *efficiently* computable function that maps a variable length string (message) to a fixed length string (digest), with additional security features such as collision resistance (*hard* to find two different messages with same hash digest) and one-wayness (*hard* to invert the hash function on a given hash value). Some popular examples of cryptographic hash functions include MD4, MD5 [7], SHA-1 [8], SHA-2, SHA-3, Skein and BLAKE. In this thesis, whenever we use the term *hash function*, we implicitly mean a cryptographic hash function. Hash functions have important practical applications in information security verticals such as digital signatures [9], encryption [10], message authentication codes (MACs) [11] and several new cryptographic protocols like password hashing [12] and bitcoins [13]. They can also be employed as ordinary hash functions, to index data in hash tables, for digital fingerprinting, to detect duplicate data or uniquely identify files, construction of caches, efficient data structures (such as Bloom filters [14]) and as checksums to detect accidental data corruption.

Apart from collision resistance, cryptographic hash functions need certain additional features like one-wayness or preimage resistance and second preimage resistance. Different cryptographic protocols exploit different security properties of hash functions. One of the main application of hash functions is in domain extension which exploits the fact that a hash function can map arbitrary length strings to fixed length strings with low collision probability. So, designers of some other primitives do not need to consider variable length strings. They can simply use hash function as a preprocessor. Another important property is the preimage resistance, i.e., given a hash output it should be computationally *hard* for an adversary to compute the preimage. In other words, the hash function should be one way.

In section 1.1 we give an informal view of the security properties of a hash function that an adversary will try to attack and some known models of attack. In section 1.2 we have briefly discussed various applications of hash functions to emphasize the need for secure hash function constructions. In section 1.3 we discuss relevant works and give a motivation for the problem that we study in this thesis. Section 1.4 lay out the organization of the thesis.

1.1 Security Properties and Models of Attack

In this section, we will briefly discuss the security properties that are desired from a hash function. We will also review some known models of attack.

1.1.1 Security Properties

A cryptographic hash function requires certain additional features other than that in a normal hash function. Most of the security requirements depend upon the particular application of hash functions. However, three basic properties as described below are always desirable. For a hash function H mapping $\mathcal{D}$ to $\mathcal{R}$,

1. Collision Resistance: It should be *hard* to find two messages $M, M' \in \mathcal{D}$ such that $M \neq M'$ and $H(M) = H(M')$.
2. Preimage Resistance: Given a hash value $h \in \mathcal{R}$, it should be *hard* to find $M \in \mathcal{D}$ such that $H(M) = h$.
3. Second Preimage Resistance: Given a message $M \in \mathcal{D}$, it should be *hard* to find $M' \in \mathcal{D}$ such that $M' \neq M$ and $H(M') = H(M)$.

Apart from these basic properties several other security properties such as herding attack [2, 4] and trojan message attack [4] are defined in the literature. All these properties will be formally defined in a later chapter.

1.1.2 Models of Attack

The attacks on cryptographic hash functions can be broadly divided into two categories:

(a) Specific attacks, i.e., attacks that exploit some specific weakness in the building blocks of the hash function such as weaknesses in the underlying compression function, or use some specific tools, such as meet-in-the-middle or differential attack.

(b) Generic attacks, i.e., attacks that do not exploit the above mentioned weaknesses, and consider only the flaws in the general design.

1.1.2.1 Specific Attack Models

These attacks exploit certain weaknesses in the particular construction or use certain cryptographic tools applicable on the construction to break the security. Some examples of such attacks are MD4 [15], MD5 [16, 17], SHA-0 [18], SHA-1 [19, 20] and Streebog [21, 22]. Since, these attack models can vary based on the construction we are listing here three basic techniques.

- Meet-in-the-Middle Attack: This attack utilises the invertibility property of hash function to create a birthday attack on two lists. One of the list is constructed using forward hash queries and the other is constructed using backward queries. This enables the adversary to create a message for a given hash value. For further details, please refer [23].
- Fixed Point Property: A function f is said to have fixed point property if $f(h, m) = h$. Dean [24] used this property to construct a second preimage attack on iterated hash functions.
- Differential Attacks: Differential Cryptanalysis [25] is based on the relation between input and output differences. In this attack the adversary searches for input differences that are likely to cause a certain output difference.

1.1.2.2 Generic Attack Models

Generic attack models do not use the internal structure of a particular hash function to construct attacks. In these attacks, the hash function is considered as a black box or oracle and the complexity is analysed in terms of the number of calls made to the oracle. The black box or oracle output in these attacks are assumed to be following uniform distribution, i.e., each of the hash output has same number of preimages. For a hash function H from $\mathcal{D}$ to $\mathcal{R}$ consider the following attacks,

- Random Attack: If the hash function in consideration follows a uniform distribution, the probability that some random domain point maps to a given range point is $1/|\mathcal{R}|$. The adversary has to choose atleast $|\mathcal{R}|$ many random domain points to get a preimage of the given range point. So, a generic attack on preimage and second preimage security of a hash function should follow this bound.
- Birthday Attack: This attack is widely used for finding collisions in generic attacks. The basic idea behind this attack follows from the so called "Birthday Paradox" which says that if a room has at least 23 randomly sampled people, then the probability that two of them share the same birthday is at least $1/2$. Analogously in the hash function setting, one needs to compute $\sqrt{|\mathcal{R}|}$ random domain points to get a pair of domain points with same hash value.

In this thesis, we will be focusing on the generic attacks and attack structures used for different hash function designs.

1.2 Applications of Hash Functions

Famous cryptographer Schneier has rightly called hash functions "the workhorses" [26] of modern cryptography. Hash functions have applications in almost all the information security verticals. We are listing few of them to highlight the importance of having secure hash function definitions.

1. **Digital Signatures and the need for Domain Extension:** Diffie and Hellman first recognised [6] the need for a one way hash function as a building block of digital signatures. Signature schemes are generally defined over fixed length messages which makes it difficult to sign a message of arbitrary length. Hash functions can be handy in this situation where the message is first hashed to a fixed length digest and the signature algorithm is then applied to this hash digest to obtain the final signature output. This paradigm is commonly known as "Hash-then-Sign".

However, now the security of the overall signature scheme will depend upon the collision resistance of the underlying hash functions. A collision pair for the hash function will give same signature there by enabling forgery of signature scheme.

2. **Authentication Schemes:** In [6] Diffie and Hellman commented,

   Not only must a meddler be prevented from injecting totally new, authentic messages into a channel, but he must be prevented from creating apparently authentic messages by combining, or merely repeating, old messages which he has copied in the past. A cryptographic system intended to guarantee privacy will not, in general, prevent this latter form of mischief.

   This introduced the notion of authentication in cryptographic protocol which up until then was considered as a subproblem of data privacy. Hash functions (generally keyed) play a very important role in the design of message authentication codes or MACs. If the hash function in use is one way (preimage resistant) and collision resistant then it is hard for an adversary to create new or modify earlier authentication tags.

3. **Commitment Schemes:** A commitment scheme works in two phases, commit phase and the reveal phase. In commit phase, first party commits to a value while keeping it hidden from the second party and in reveal phase, the first party has to reveal the committed value. The commitment scheme has the following basic cryptographic requirements:

   (a) First party should be unable to change the committed value, once committed in the commit phase.

   (b) Second party should be unable to gain any information about the committed value before the reveal phase.

   Commitment scheme is a very important cryptographic protocol useful in biddings and future contracts. Hash functions can be easily used to construct commitment schemes. Suppose the first party has to commit M. A commitment to M is $H(M)$, where H is a secure hash function. Now, if H is preimage resistant then the second party will not gain any information about M from $H(M)$. Similarly, if H is collision resistant then the first party cannot change the committed value because that will imply that he has a valid collision pair.

4. **Data Integrity:** Hash functions have also been used for preserving data integrity, i.e., it helps in maintaining and assuring the consistency and accuracy of data. For example MD5, SHA-1 or SHA-2 hashes are sometimes posted along with the data on websites to allow verification of integrity [27]. If the hash function behaves as random oracle, then any inconsistencies or inaccuracies in the downloaded data will be reflected by the incorrect hash value.

1.3 Related Works and Motivation for this Work

Generally, a hash function is supposed to take variable length input, i.e., the hash function should be able to map variable length strings to a fixed length output string. Construction of such a function from scratch is very difficult. To overcome this difficulty a function of fixed input size (known as *compression function*) is used as per a set of rules (known as *combining rules* or *hash designs*) to extend the input size to any arbitrary length. Yet another use of combining rules is to increase the security of the hash function (at least, this is what the designer aims). Essentially, a hash function is a combination of one or more well defined compression functions and a combining rule. Some examples of compression function schemes are Davies-Meyer [28], Matyas-Meyer-Oseas [29] and Miyaguchi-Preneel [30]. Some examples of combining rules are Merkle-Damgård construction [31, 32], Concatenated Hash [4, 33], Hash Twice [4], HAIFA [34], Sponge [35] and Zipper Hash [4, 36]. Most of them use iterations of a compression function and hence are grouped as iterated hash and its variants.

In this thesis we survey some of the cryptanalytic results on popular iterated hash variants such as Hash Twice, Concatenated Hash and Zipper Hash. We will analyse the security of these constructions in the presence of various weaknesses and provide new techniques to attack these constructions. Our main focus will be the analysis of preimage, second preimage and herding resistance of these constructions.

The cryptanalytic study of iterated hash and its variant got a major boost with the seminal work [37] of Joux in which he found a very efficient way of finding multicollisions on iterated hash constructions. Kelsey and Schneier then expanded this idea to construct second preimages for long messages. Kelsey and Kohno proposed the CTFP attack [2] on Merkle-Damgård (or MD) constructions using a new data structure called *diamond structure*. This was later studied in more detail by Upadhyay in his thesis [38]. The diamond structure was also found to be suitable for other attacks [4, 38–40] where the known methods have failed. Another data structure called Kite was proposed in [38, 40] to construct an attack on dithered hash and hash twice. These works opened new avenues for research in hash function constructions. Since the vulnerability of simple iterated hash construction was evident due to various attacks, efforts were made to somehow amplify the security. Liskov proposed a construction [36] called Zipper Hash that makes two passes on the input message where the second pass uses the block-wise reverse of message. Hoch and Shamir [41] proposed a XOR construction and showed its robustness in collision property. Several other constructions [42, 43] were proposed that amplifies the security of the underlying compression functions.

Andreeva *et al.* [4] analysed the CTFP, second preimage and trojan message resistance of some MD variants like Concatenated Hash, Hash Twice and Zipper Hash. They showed that both concatenated hash and hash twice are susceptible to herding attacks and that the hash twice construction does not provide full second preimage resistance. In case of zipper hash they argued that conventional herding attack is as hard as finding a preimage and gave a modified herding attack for it. Recently, Leurent *et al.* [3] gave a surprising results for XOR combiner. They showed that the XOR combiner is weaker than the constituent hash functions in preimage resistance. They also proposed an innovative data structure called the switch and interchange structure. Chen *et al.* [5] gave a $2^{n/2}$ second preimage attack on zipper hash with invertible compression functions.

The (second) preimage security of concatenated hash and zipper hash has for long been an open problem. Recent results have raised a valid doubt that these constructions may not be fully secure. Also, it will be interesting to analyse the security of these constructions in the presence of weaker compression functions. In this thesis, we will try to solve these problems. We will mainly deal with the zipper and concatenated hash functions. We aim to analyse the (second) preimage and herding attack resistance of zipper hash in random oracle model. For this we will introduce a new data structure called *Rho*. We will also analyse the (second) preimage resistance of concatenated hash with weaker compression functions. Apart from these, we will also introduce three new data structures, *chain structure*, *multi-pipe expandable message set*, and the *rho structure*. We will show a surprising application of chain structure in improving the complexity of herding attack on iterated hash.

1.4 Organisation of Thesis

The thesis has been organised in following chapters,

1. Chapter 2, covers the basics of hash functions and their security definitions. It also introduces the notations that we will be using throughout this thesis. We will also give some necessary definitions and results in graph theory and probability.

2. In chapter 3, we survey some popular variants of iterated hash function. We will also review the PGV compression function family.

3. In chapter 4, we briefly discuss various known data structures and techniques used for constructing attacks on hash functions. We will also summarise the known attacks on iterated hash variants like hash twice and concatenated hash.

4. In chapter 5, we will introduce two new structures, a chain structure and a multi-pipe expandable message set. We will also present our analysis of concatenated hash function in the presence of weak compression functions.

5. In chapter 6, we will present our analysis on zipper hash function. We will also introduce the new data structure *Rho* used in the analysis.

6. Chapter 7, concludes the thesis and gives a brief discussion on future course of research.

Chapter 2

Preliminaries

In this chapter, we will establish our notations, and formally define a hash function and its security goals. We will also mention some fundamental results in probability and graph theory, that will be useful in the later parts of this thesis.

2.1 Mathematical Background

2.1.1 Notations and Notes

We will denote the set of all integers by $\mathbb{Z}$, and the set of all natural numbers by $\mathbb{N}$. For any $n \in \mathbb{N}$, $\{0,1\}^n$ represents the set of all n-bit binary strings. $\{0,1\}^+$ denotes the set of all binary strings of length > 0, and $\{0,1\}^*$ denotes $\{0,1\}^+ \cup \{\phi\}$, where ϕ represents the empty string. For any $x \in \{0,1\}^n$, $|x|$ denotes the length (in no. of blocks) of x, and $\langle x \rangle_k$ denotes the k-bit binary representation of $|x|$. For $x, y \in \{0,1\}^n$, $x \| y$ represents the concatenation of x and y. We denote the padded version of a message $m \in \{0,1\}^*$ by $\overline{m} := (m_1, m_2, \cdots, m_l)$, where m_i denotes the individual message block for $1 \leq i \leq \ell$.

If $\mathcal{X}$ is a finite set, then $x \xleftarrow{\$} \mathcal{X}$ denotes the uniform random sampling of x from $\mathcal{X}$.

We will write $\lg n$ for the logarithm of n to the base 2 and $\ln n$ for the natural log of n.

Note 1. For $x \ll 1$, $e^x \approx 1 + x$. This can be easily deduced, using the first order approximation of e^x, in the Taylor series expansion of the exponential function.

Note 2. For $x \ll 1$, $(1+x)^n \approx 1 + nx$. This can be easily deduced, using the binomial expansion of $(1+x)^n$, and ignoring the higher degree terms in x (as $x \ll 1$).

2.1.2 Asymptotic Complexity

Most of the theoretical analysis of algorithms consider the asymptotic complexity of a particular algorithm. We will be using the following asymptotic notations in our complexity analysis:

1. $f(x) = O(g(x))$, iff there exists a constant $c > 0$ and a real number x_0, such that $|f(x)| \leq c|g(x)|$ for all $x > x_0$. This is generally used to give an idea of the upper bound.

2. $f(x) = \Omega(g(x))$, iff there exists a constant $c > 0$ and a real number x_0, such that $|f(x)| \geq c|g(x)|$ for all $x > x_0$. This is generally used to give an idea of the lower bound.

3. $f(x) = \Theta(g(x))$, iff there exist constants $c_1, c_2 > 0$ and a real number x_0, such that $c_1|g(x)| \leq |f(x)| \leq c_2|g(x)|$ for all $x > x_0$. This signifies that the function f (complexity function) grows in a similar fashion as g, and thus it can be approximated by g.

2.1.3 Probabilistic Results

Lemma 2.1. *From a set of N distinct elements q elements are randomly drawn with replacement. The probability p, that some of them will be equal to a given value x is,*

$$1 - (1 - \tfrac{1}{N})^q$$

Proof. Consider the probability of the complement event, i.e., the probability that none of the q elements is equal to x is,

$$(1 - \tfrac{1}{N})^q$$

The result follows. □

Remark 2.2. For $n \gg q$, $p \approx 1 - e^{-\frac{q}{N}}$ (using Note 1). Therefore, for $p \geq \frac{1}{2}$ we require a sample size, $q = O(N)$.

Lemma 2.3. *From a set of N distinct elements q elements are randomly drawn with replacement. The probability p, that at least two of the drawn elements are equal is,*

$$1 - (1 \cdot (1 - \tfrac{1}{N}) \cdot (1 - \tfrac{2}{N}) \cdots (1 - \tfrac{q-1}{N}))$$

Proof. The probability that all q elements of the sample are distinct is,

$$(1 \cdot (1 - \tfrac{1}{N}) \cdot (1 - \tfrac{2}{N}) \cdots (1 - \tfrac{q-1}{N}))$$

The result follows. □

Remark 2.4. For $N = 365$ and $q = 23$, $p \approx \frac{1}{2}$. This is the famous birthday problem or birthday paradox.

Remark 2.5. For $N \gg q$,

$$p = 1 - (1 \cdot (1 - \tfrac{1}{N}) \cdot (1 - \tfrac{2}{N}) \cdots (1 - \tfrac{q-1}{N})) \approx 1 - (e^{-\frac{1}{N}} e^{-\frac{2}{N}} \cdots e^{-\frac{q-1}{N}}) = 1 - e^{-\frac{q(q-1)}{2N}}$$

Therefore, for $p \geq \frac{1}{2}$ we require a sample size, $q = O(\sqrt{N})$.

Lemma 2.6. *Let P and R be two randomly chosen subsets of set S, such that $|P| = q_1$, $|R| = q_2$, $|S| = N$ and $q_1 + q_2 < N$. Then the probability p, that $P \cap R \neq \phi$ is greater than,*

$$1 - (1 - \tfrac{1}{N})^{q_1 q_2}$$

Proof. The probability that $P \cap R = \phi$ is,

$$\frac{\binom{N}{q_1}\binom{N-q_1}{q_2}}{\binom{N}{q_1}\binom{N}{q_2}} = \frac{\binom{N-q_1}{q_2}}{\binom{N}{q_2}} = \frac{(N-q_1)!}{(N-q_1-q_2)!}\frac{(N-q_2)!}{N!}$$
$$= (1 - \tfrac{q_1}{N})(1 - \tfrac{q_1}{N-1}) \cdots (1 - \tfrac{q_1}{N-q_2+1}) < (1 - \tfrac{q_1}{N})^{q_2} < (1 - \tfrac{1}{N})^{q_1 q_2}$$

The result follows. □

Remark 2.7. For $N \gg 1$, the above probability $p \approx (1 - e^{\frac{q_1 q_2}{N}})$. Therefore, for $p \geq \frac{1}{2}$ we require $q_1 q_2 = O(N)$.

Remark 2.8. The above lemma can be generalized to k sets of size $q_1, q_2, \cdots, q_k$ such that $q_1 + q_2 + \cdots + q_k < N$. In this case, for $p \geq \frac{1}{2}$ we require $q_1 q_2 \cdots q_k = O(N)$. This is called generalised birthday problem.

2.1.4 Graph Theoretic Definitions

Definition 2.9. Directed Graph: A directed graph or "digraph" G, is a 2-tuple (V, E) consisting of,

- Set of vertices V in G.

- Set of arcs $E \subseteq V \times V$, such that an ordered pair $(u, v) \in E$ represents a directed arc from u to v in G.

A digraph $G' : (V', E')$, is called the *sub-graph* of G, if $V' \subseteq V$ and $E' \subseteq E$.

Definition 2.10. Walk: For a digraph $G : (V, E)$, a walk is defined as a sequence of vertices connected by arcs corresponding to the order of the vertices in the sequence.

A walk is considered *closed*, if the starting vertex is the same as the ending vertex, and *open* otherwise. A walk with no repeated vertices, is called a *path*. A closed walk where no other vertices are repeated, apart from the start vertex, is called a *cycle*. A closed walk with no repeated arcs, is called a *circuit*.

Definition 2.11. Directed Arc-Labeled Graph: A directed arc-labeled graph G is a 3-tuple (V, L, E) consisting of,

- Set of vertices V in G.
- Set of arc labels L.
- Set of arcs $E \subseteq V \times V \times L$, such that an ordered pair $(u, v, \ell) \in E$ represents a directed arc from u to v with label ℓ in G.

2.2 Basics of Hash Function

Definition 2.12. Hash Function: An n-bit hash function H can be defined as,

$$H : \mathcal{M} \to \mathcal{H},$$

where $\mathcal{M}$ and $\mathcal{H}$ are called the message space and digest space, respectively.

Generally, $\mathcal{H}$ is a finite set, and $\mathcal{M}$ is either infinite or $|\mathcal{M}| \gg |\mathcal{H}|$. The strings over an arbitrary finite alphabet can be encoded into strings over a binary alphabet. So, without loss of generality, we can assume that all our messages and digest values are over the binary alphabet $\{0, 1\}$. From now on, we will assume that the message space is $\{0, 1\}^*$, and the digest space is $\{0, 1\}^n$.

2.2.1 Random Oracle Model

The random oracle model represents the idea of an "ideal" hash function. In this model, the adversary is given only oracle access to the hash function, that is, the hash function is like a black box. This means the adversary does not has an access to the actual implementation of the hash function. The only way to compute the hash value is to query the oracle interface. An important property of the random oracle is that, if the adversary queries the same value multiple times, it gets the same output each time. A secure hash function is expected to offer the same level of resistance as offered by the random oracle.

In some cases, the adversary is given access to some additional interfaces, to formulate the model of weakness in hash designs. In this thesis, we will consider following additional interfaces:

1. *Fixed point interface*, which on input y, returns a random value x, such that $f(x, y) = x$.
2. *Weakly invertible interface*, which on input z, returns a random pair (x, y), such that $f(x, y) = z$.
3. *Strongly invertible interface*, which on input (y, z), returns a random value x, such that $f(x, y) = z$. This is referred as *Backward interface* [3]. Another variant returns y, on input (x, z), such that $f(x, y) = z$. This is referred as *Bridging interface* [3].

Observe that the strongly invertible interface gives more control to the adversary, as compared to a weakly invertible interface. The adversary gets to fix her choice of x or y. In other words, a strongly invertible interface can work as a weakly invertible interface, and hence it is a stronger notion of weakness.

2.2.2 Security Notions for Hash Function

There are several popular security notions for hash functions in literature. The most basic being collision resistance, preimage resistance and second preimage resistance. Rogaway *et al.* have coined seven different variants of these basic properties for a keyed hash function (see [44] for more details). Since, we are focussing on unkeyed hash functions, we are enumerating the three basic properties along with two additional properties, namely herding and trojan message resistance. For a hash function H consider,

1. **Collision Resistance:** H is said to be collision resistant, if it is *hard* for all *efficient* adversaries to find a $(M, M^{'})$ message pair, such that $H(M) = H(M^{'})$.

2. **Preimage Resistance:** H is said to be preimage resistant if given a hash value h, it is *hard* for all *efficient* adversaries to find a message M, such that $H(M) = h$.

3. **Second preimage Resistance:** H is said to be second preimage resistant if given a message M, it is *hard* for all *efficient* adversaries to find a message $M^{'} \neq M$, such that $H(M^{'}) = H(M)$.

4. **Chosen-Target Forced-Prefix (CTFP) or Herding Resistance:** In [2] Kelsey and Kohno proposed a security game that can be described as follows:

   (a) In the first phase, adversary $\mathcal{A}$ commits a hash value h. The hash value h is called *chosen target*.

   (b) In the second phase, the challenger $\mathcal{C}$ selects a prefix P and gives it to $\mathcal{A}$. The prefix P is called *forced prefix*.

   (c) In the third phase, $\mathcal{A}$ has to produce a suffix S, such that $H(P||S) = h$.

   A similar game (CTFS) can be constructed with forced suffix also. This notion has been employed in [4] to construct herding attack on the zipper hash. A hash function is said to be herding resistant, if it is *hard* for all *efficient* adversaries to produce suitable suffix(prefix) in CTFP(CTFS).

5. **Trojan-Message Resistance:** Trojan-message attack is a security game proposed by Andreeva *et al.* in [4]. For an adversary $\mathcal{A}$, it can be described as follows:

   (a) In the first phase, based on a constrained set of prefixes $\mathcal{P}$, the adversary $\mathcal{A}$ commits a string S, called the *trojan message*.

   (b) In the second phase, the challenger $\mathcal{C}$ selects a prefix P from $\mathcal{P}$. The challenger gives $P||S$ to $\mathcal{A}$.

   (c) In the third phase, $\mathcal{A}$ has to produce a second preimage T for $P||S$.

   A hash function is said to be trojan message resistant, if it is *hard* for all *efficient* adversaries to produce a suitable second preimage in the above security game.

2.2.3 Iterated Hash

A hash function is generally required to work for any arbitrary length input. Further, it is widely used as an input preprocessor in online fashion. To fulfil these requirements, a fixed length input function is used on the input (part of it at a time), according to a set of combining rules. The most popular of them is the iterated hash, in which the message is

read in from left to right, block-by-block and the hash value of previous iteration is used along with the current block, to produce the next hash value. A compression function is at the core of the iterated hash design.

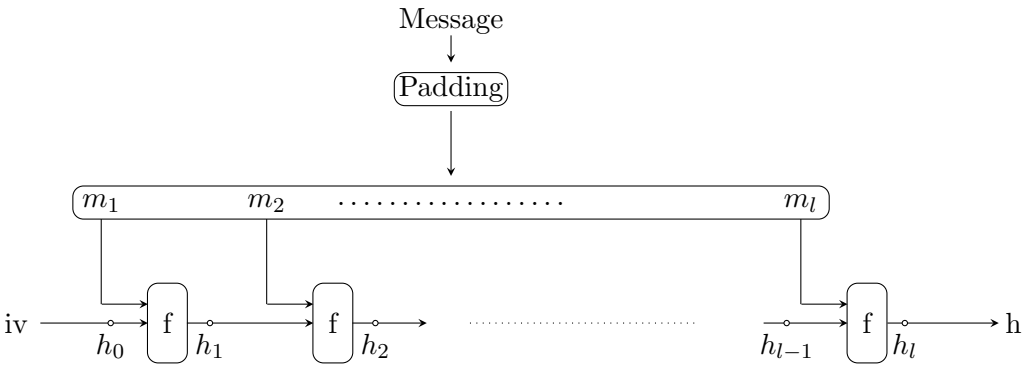

FIGURE 2.1: Construction of f^+ by iterated use of f function.

Definition 2.13. Compression Function: A compression function f of block size n' and length n, can be defined as,

$$f : \{0,1\}^n \times \{0,1\}^{n'} \to \{0,1\}^n$$

where $n', n \in \mathbb{N}$ and $n' > n$.

Note that, $n' > n$ has been chosen for simplicity, and it is not a necessary property for the analysis presented in this thesis. The adversary can simply increase the number of blocks, to make the overall size greater than n. Unless otherwise stated, lower-case letters f, g and their variants such as f', f_i where $i \in \mathbb{N}$, will denote compression functions.

Definition 2.14. Iterated Hash Function: Using the compression function f in definition 2.13, and an n-bit constant iv, we can define the iterated hash function f^+ as follows,

$$f^+(\text{iv};\ m_1, m_2, \ldots, m_\ell) = h_\ell := f(\cdots f(f(\text{iv}, m_1), m_2) \cdots).$$

We can also write $h_i = f(h_{i-1}, m_i)$ where $h_0 = \text{iv}$, $1 \leq i \leq \ell$.

Iterated hash functions have a very nice representation using directed arc-labeled graphs. A directed arc-labeled graph G corresponding to a compression function f has the set of vertices $V = \{0,1\}^n$, the set of labels $L = \{0,1\}^{n'}$ and whenever $f(u,m) = v$ we have a label m for the arc (u,v). We write $u \xrightarrow{m} v$. Thus, the sub-graph in figure 2.2 represents the computation of the iterated function f^+.

We refer to such graphs as structures, and paths in such structures will be referred as *chains*. Note that we may have $h_i = h_j$ and so the above structure is a walk and

not necessarily a path (chain). The hash values on these structures will be referred as *chaining values*.

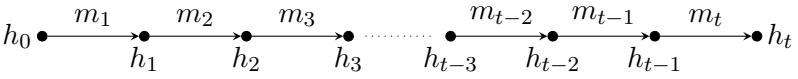

FIGURE 2.2: Graphical representation of iterated hash.

In this thesis, we also discuss attacks on some variants of the iterated hash. These designs may be two-pass, i.e., they may use the message blocks twice (in some order) and the underlying compression function for each pass may be different. We differentiate between the computations on different compression functions, by using thick edges for the second pass compression function, and denote the junction point between the two passes by an unfilled node (i.e. $\circ$).

In practice, messages may not have length that is divisible by n'. So, it is necessary to do some preprocessing for length correction. This is called *padding*, which generally employs an injective function P, such that for an arbitrary message M, $|P(M)|$ is divisible by n'. Two popular padding rules are,

1. P_{ozp}, defined as $P_{ozp}(M) = M \| 10^d$, and
2. P_{sMD}, defined as $P_{sMD}(M) = M \| \langle M \rangle_{64} \| 10^d$,

where d is the smallest non-negative integer for which length of the padded message is divisible by n'.

With $P = P_{sMD}$, the composition $f^+ \circ P$ is called the *MD hash function*, proposed independently by Merkle [31] and Damgård [32]. The MD hash function is collision resistant, if the underlying compression function is collision resistant [31, 32].

2.2.4 Attack Algorithms and their Complexity

In this thesis, we are considering generic attack algorithms on hash designs. We will assume that the adversary knows the hash design, but the underlying compression function is like a black box for the adversary. In other words, the adversary knows the combining rule to use while working with a compression function f, but the concrete definition of f is not known to the adversary. This setting is called the random oracle model (refer 2.2.1). In certain cases, we will consider some weaker notions of random oracle model, where some additional interfaces will be available to the adversary. But, even in these settings the actual definition of f will not be known to the adversary.

Most of the attacks discussed in this thesis can be analysed by analysing the underlying structures, techniques and the relationship between them. For example, suppose an attack uses internally the Joux multicollision (complexity $O(2^{n/2})$), then it is inferred that it requires at least $O(2^{n/2})$ computations. We will be using similar analysis, and moreover we will focus on the structural elaboration of the attack algorithms.

All the attacks discussed in this thesis are constructed as (a sequence of) distinct statistical experiments. The (sequence of) experiment(s) is modelled as independent Bernoulli trials, which is repeated until the required event $\mathcal{W}$ happens. Suppose the probability of the event $\mathcal{W}$ is p and let X be the number of repetitions needed. Then X is a random variable that follows the geometric distribution, and the expected value $E(X)$ of X satisfies $E(X) = \frac{1}{p}$.

From the discussion in the preceding paragraph and the fact that f is a random oracle, it is clear that the adversary cannot perform successful preimage attack in any other way than through random search. The same is expected for second preimage attacks against messages with just a single message block. Using remark 2.2, we can conlcude that finding a preimage or second preimgae should take $O(2^n)$ oracle queries. Similar argument can be given to say that if f is modelled as random oracle, then the hash function will be collision resistant upto birthday bound, i.e., $O(2^{n/2})$ oracle queries (refer remark 2.5). So, any hash function that claims to be secure in the random oracle model should offer similar query bounds. Note that these bounds can be significantly lowered by considering specific implementations of hash function and their underlying compression functions. This is evident from the attacks by Wang *et al.* [15, 16, 18, 20], Biham *et al.* [19], Klima [17] and Joux *et al.* [45] on hash functions based on the MD4 design, such as MD5, RIPEMD, SHA-0 and SHA-1.

2.2.4.1 On Computational Complexity

In the random oracle model, the adversary can only make compression function calls to compute hash values (or compute preimage for hash value in the weaker model). So, it seems reasonable to consider the expected number of compression function calls as a measure for efficiency of an attack algorithm. Indeed, this has been the case with most of the works on generic attack[1, 2, 37]. We will follow a similar approach in this thesis. For example, if a compression function outputs n bits, then we will state that it has $O(2^{n/2})$ collision security and $O(2^n)$ preimage and second preimage security.

Suppose the adversary is constructing collision attack on the above function, then it will require $t \cdot 2^{n/2}$ expected no. of oracle calls, where $t = O(n^k), k \in \mathbb{N}$. We can ignore the

factor t for sufficiently large values of n. In other words, we will ignore the logarithmic and polynomial factors whenever appropriate.

2.2.4.2 On Memory Complexity

Memory is another constraint for an attack algorithm. Suppose the adversary is constructing a collision attack, where she is supposed to store $2^{n/2}$ hash values for a computational complexity of $O(2^{n/2})$. But, if the required memory is not available, then the computational cost will increase. So, the adversary has to find an equilibrium between space and time. This is called *time-memory trade-off*.

In hash function analysis, many attack algorithms work in two phases, an *offline* phase and an *online* phase. In the offline phase, the adversary generally precomputes certain structures, which can reduce the computational complexity in the online phase. This increases the memory complexity as the adversary has to store these structures for online use. The general principle is to make offline phase of higher computational cost to decrease the online cost.

We will consider two parameters while analysing the memory complexity of an algorithm. First, the online storage of intermediate hash values and message blocks, and second, the size of offline created structures.

2.2.4.3 On Message Complexity

In this thesis, we will also analyse the number of random message blocks sampled for constructing an attack. In most of the cases it closely resembles the computational complexity. But, for certain attacks and structures, it reveals a very peculiar property; the adversary's control over the message content is inversely proportional to the number of random message blocks sampled. We will term this analysis as the message complexity of an attack algorithm.

Chapter 3

Iterated Hash Variants and PGV Function Family

In this chapter, we will survey the variants of iterated hash such as concatenated hash, hash twice, XOR combiner and the zipper hash. Though these designs have many improvements over the iterated hash designs, yet they inherit many cryptanalytic flaws from iterated hash. We will also review the PGV compression function family based on block ciphers. These schemes are very popular and many concrete hash function definitions such as MD5, SHA-1 etc. are based on these functions.

3.1 Iterated Hash Variants

In order to improve over the security offered by iterated hash many variants were proposed. These efforts were generally concentrated in two directions. First, combining two or more independent hash functions into one hash function. The aim was to amplify the security of the combined hash function by using the security of its constituent hash functions. Combiners like concatenated hash and XOR combiner fall under this category. Second, multiple passes over the message blocks. The aim was to increase the dependency of hash output on the given message blocks. Hash functions in this category are practically inefficient due to their multi-pass nature. Designs like the zipper hash and hash twice fall under this category.

3.1.1 Concatenated Hash

Concatenated hash function uses two independent hash functions H and G internally. The message M is hashed on each of the hash functions, and outputs are concatenated

to produce the output of the concatenated hash function (see figure 3.1), i.e, formally for hash functions H and G, the concatenated hash $\mathcal{C}_{HG}$ is defined as,

$$\forall m \in \{0,1\}^*, \mathcal{C}_{HG}(m) := H(m)||G(m)$$

Ideally, the hash function $\mathcal{C}_{HG}$ should provide 2^n collision resistance and 2^{2n} preimage resistance. But, Joux [37] showed that if one of the component hash functions is based on MD hash function, then the collision and preimage security reduces to $2^{n/2}$ and 2^n respectively. In particular, the concatenated hash does not amplifies the security. But it is a robust construction, i.e., it offers at least same level of security as offered by its constituent hash functions. Later, Hoch *et al.* [41] analysed the security of concatenated hash with weak compression functions and proved that it offers $\frac{n}{2}$-bits security for collision and preimage resistance. In this thesis we will show that even if one of the compression function is weak, then a second preimage can be computed in less than 2^n computations.

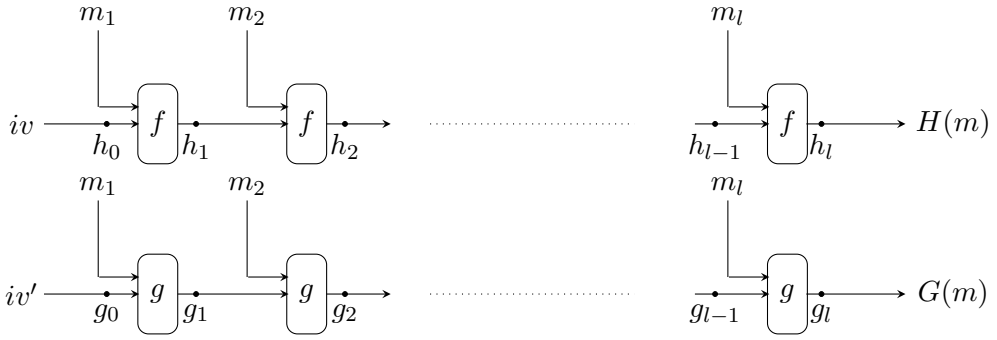

FIGURE 3.1: Schematic of Concatenated Hash $\mathcal{C}_{HG}$.

3.1.2 XOR Combiner

The XOR combiner is similar to the concatenated hash in its use of two independent hash functions H and G. But instead of concatenating $H(\cdot)$ and $G(\cdot)$ outputs it XOR them, i.e, formally for hash functions H and G, the XOR combiner $\mathcal{X}_{HG}$ is defined as,

$$\forall m \in \{0,1\}^*, \mathcal{X}_{HG}(m) := H(m) \oplus G(m)$$

Recently, Laurent *et al.* [3] have shown that the XOR combiner is not robust in preimage resistance, i.e., it offers less than 2^n preimage security.

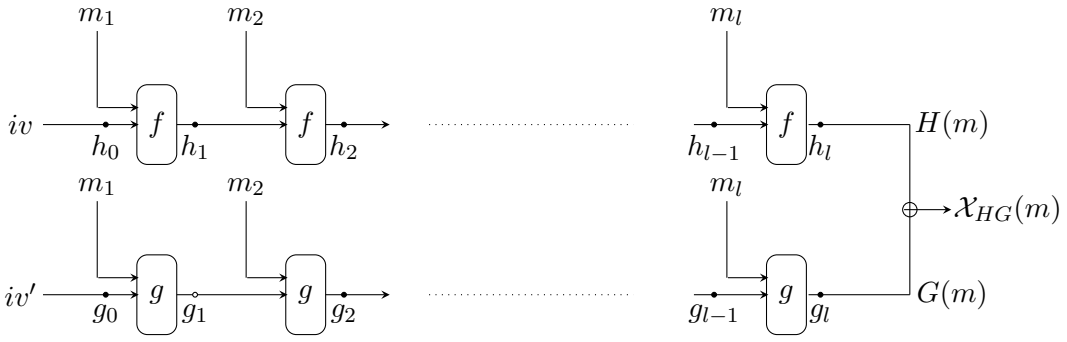

FIGURE 3.2: Schematic of XOR Combiner $\mathcal{X}_{HG}$.

3.1.3 Zipper Hash

At SAC 2006, Liskov proposed the zipper hash [36] and argued that it is indistinguishable [46, 47] with random oracle even when the underlying compression functions have some weaknesses. Conceptually, the zipper hash [36] is a two-pass hash function which uses a padded message $\overline{m}$ in the first pass and its block-wise reverse in the second pass. Formally, an n-bit zipper hash function $\mathcal{Z}$ based on two independent compression functions f_1 and f_2 (and a fixed initial value iv) can be defined as

$$\mathcal{Z}(\overline{m}) := f_2^+(f_1^+(\text{iv}; \overline{m}); \overline{m}^{\text{rev}})$$

where $\overline{m} = (m_1, \ldots, m_\ell)$ and $\overline{m}^{\text{rev}} = (m_\ell, \ldots, m_1)$ is the block-wise reverse of $\overline{m}$.

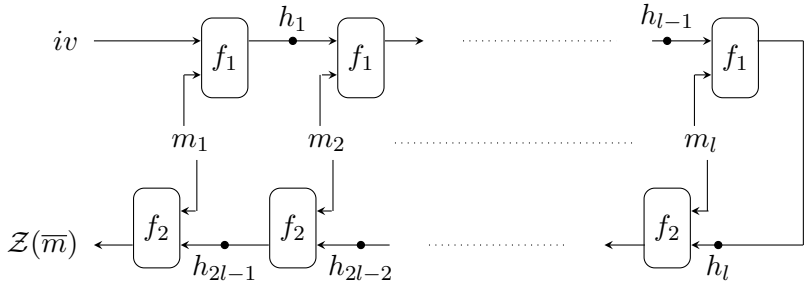

FIGURE 3.3: Schematic of Zipper Hash $\mathcal{Z}$.

Though zipper hash was originally defined to create secure hash functions from weak compression functions, however further study [48] showed that assumed weaknesses in compression functions create fatal flaws in the design. We will consider the zipper hash in many different settings, such as f_1, f_2 are random oracles, f_1, f_2 have some form of weakness (see chapter 2), and $f_1 = f_2$. The zipper hash with $f_1 = f_2$ will be referred as the *relaxed zipper hash*. Zipper hash is practically inefficient due to its two pass nature. But, theoretically it has been an intriguing design as it has resisted several

known attacks [1, 2, 39, 40]. The first significant attack on the zipper hash is given by Andreeva *et al.* [4], which is concerned with the herding attack. Recently, Chen *et al.* [5] have shown a second preimage attack for zipper hash with weak compression functions.

3.1.4 Hash Twice

The hash twice construction is a multi-pass scheme similar to the zipper hash. Here, two consecutive copies of the message are hashed one after another (see figure 3.4). Formally, an n-bit hash twice function $\mathcal{T}$ based on two independent compression function f_1 and f_2 is defined as,

$$\mathcal{T} := f_2^+(f_1^+(iv; \overline{m}); \overline{m})$$

Hash twice is practically insignificant due to its multi-pass over message blocks. And as it turns out, the hash twice design only marginally increases the security of a plain iterated hash construction. Andreeva *et al.* [4] have proposed second preimage and herding attacks on hash twice, which show that this design is not secure.

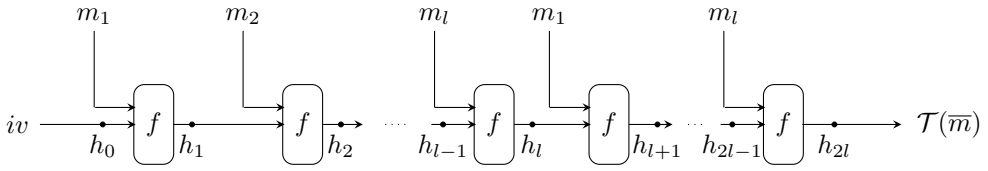

FIGURE 3.4: Schematic of Hash Twice $\mathcal{T}$.

3.2 PGV Compression Functions

Preneel, Govaerts, and Vandewalle [49] considered 64 most basic ways to construct a compression function from a block cipher $E : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}^n$. They regarded 12 of these 64 schemes as secure and the remaining 52 schemes were shown to be vulnerable to various attacks (see [49] for more information). Black, Rogaway and Shrimpton [50] gave more formal treatment and security proofs to these basic schemes. They categorised the 64 schemes into three groups,

1. *Group 1*: They grouped 12 constructions in the first group. The schemes in this group were found to be both collision and preimage resistant.

2. *Group 2*: They grouped another 8 constructions in the second group. The schemes in this group provide the same level of collision security as in group 1. But, their preimage resistance is lower than the group 1 schemes.

3. *Group 3*: The remaining 44 constructions were found to be neither collision nor preimage resistant. These were grouped in the third group.

So, effectively only the first two groups are useful for hash function constructions. Some of the popular block cipher based constructions that belong to the PGV groups are described in the following subsections.

3.2.1 Davies-Meyer

The Davies-Meyer compression function uses each block of the message (m_i) as the key to a block cipher. It feeds the previous chaining value (h_{i-1}) as the plaintext to be encrypted. The output ciphertext is then XORed with the previous chaining value (h_{i-1}) to produce the next chaining value (h_i). Formally, if E represents a block cipher then, the Davies-Meyer compression function can be described by the relation, $h_i = E_{m_i}(h_{i-1}) \oplus h_{i-1}$.

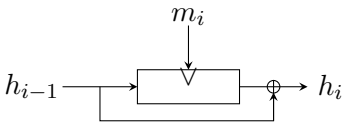

FIGURE 3.5: Schematic of Davies-Meyer construction.

The Davies-Meyer function is a *group* 1 PGV scheme (f_5 in [50]). This scheme has been employed in the SHA-1 hash function. A notable property of the Davies-Meyer construction is that it is possible to compute fixed points for this function, i.e., for any m, one can find a value of h such that $E_m(h) \oplus h = h$ (by setting $h = E_m^{-1}(0)$).

3.2.2 Matyas-Meyer-Oseas

The Matyas-Meyer-Oseas (MMO) compression function can be considered as a dual of the Davies-Meyer function with respect to its treatment of message block and the previous chaining value. It uses the previous chaining value h_{i-1} as the key and feeds the message block as the plaintext to be encrypted. The output ciphertext is then XORed with the message block to produce the next chaining value.

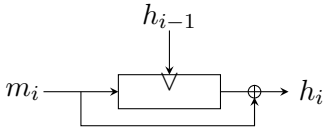

FIGURE 3.6: Schematic of Matyas-Meyer-Oseas construction.

Formally, the MMO compression function can be described by the relation, $h_i = E_{h_{i-1}}(m_i) \oplus m_i$. Similar to Davies-Meyer, the MMO function is a *group* 1 PGV scheme (f_1 in [50]). But, unlike Davies-Meyer, fixed point computation is not easy for this scheme.

3.2.3 Miyaguchi-Preneel

The Miyaguchi-Preneel compression function can be viewed as an extension of MMO function. In this scheme, the output ciphertext is XORed with both the message block and the previous chaining value. Formally, the Miyaguchi-Preneel compression function can be described by the relation, $h_i = E_{h_{i-1}}(m_i) \oplus m_i \oplus h_{i-1}$.

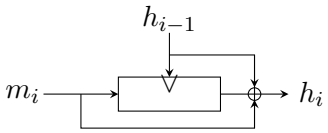

FIGURE 3.7: Schematic of Miyaguchi-Preneel construction.

This is a *group* 1 PGV scheme (f_3 in [50]) and offers similar security as offered by MMO. This scheme is used in the Whirlpool [51] hash function.

3.2.4 Notable Weaknesses in Some PGV functions

Among the 64 PGV schemes, only 20 schemes were found to be collision resistant (see [50] for details). Further, 8 of these schemes (*group* 2) have lower preimage resistance ($O(2^{n/2})$). We will briefly discuss various security gaps that an adversary can exploit in the 20 collision resistant PGV hash schemes.

1. *Weakly invertible compression functions:* Four of the eight *group* 2 PGV schemes (f_{13}, f_{14}, f_{16}, f_{18}) are weakly invertible compression function, i.e., adversary has the power to compute random preimage values for these schemes.

2. *Strongly invertible compression functions:* Four of the eight *group* 2 PGV schemes (f_{15}, f_{17}, f_{19}, f_{20}) have this weakness (see [50] for structural information).

3. *Fixed point weakness:* Four of the twelve *group* 1 PGV schemes have fixed point weakness property. Further, for two *group* 1 PGV schemes (f_{10} and f_{11}) random fixed points can be easily computed. Surprisingly, none of the *group* 2 PGV schemes have fixed point weakness.

Chapter 4

Existing Structures, Techniques And Attacks

In this chapter, we will review some of the existing structures and techniques for constructing attacks on the iterated hash and its variant designs. We will also review some of the known attacks on hash twice, concatenated hash and zipper hash.

4.1 Techniques

Here we are reviewing some of the attack techniques and strategies used in this thesis.

4.1.1 A Note on Collisions Between Lists

Given two lists L_1 and L_2 of size 2^k and 2^{n-k} respectively, with high probability we can find a collision between these lists (see lemma 2.6 in chapter 2). Note that one of the lists can be generated run-time while finding collision. This technique is called hitting and is very useful in finding a linking string[1] from a random chaining value to a chain. Meet-in-the-middle is a special case of the two list collision in which, one of the list is created by making forward queries and the other list by backward queries. This technique is extensively used for constructing attacks or structures. We denote these three techniques by algorithms LISTCOLL(L_1, L_2), MEET-IN-MID$_{f,g}(h, h')$ and HITTING$_f(h, L)$.

[1] A string is a concatenation of one or more message blocks.

4.1.2 A Note on Time-Memory Tradeoff Attacks

One-wayness of a function is the most fundamental problem in cryptography [52]. Hellman [53] first showed a *preimage attack* or in other words, broke the one-wayness of any function H (i.e., *given h, to find M such that $H(M) = h$*). In his attack, the online time T and memory M satisfy the relation $TM^2 = 2^{2n}$. Consequently, the attack is known as a time/memory trade-off (TMTO) algorithm. Note that in the TMTO set-up, the precomputation time is not considered, since this is a one-time off-line activity which can be performed at the cryptanalyst's leisure. Several efforts have been made to improve the time/memory trade-off curve [54–58]. However, most of the improvement either assumes multiple targets or works for a specific structure of a one-way function, e.g., stream ciphers [54, 57].

4.2 Structures

As stated in chapter 2, the iterated hash can be represented in terms of graphs or *structures*. These structures form an important basis for constructing attacks on the iterated hash schemes and its variants. Here we are reviewing some of the popular structures.

4.2.1 Joux's Multicollision

For an iterated hash based on the compression function f with n-bits output,

> $\mathcal{M} = \{M_i : 0 \leq i \leq r\}$ is said to be an r-multicollision set, if $\forall\, M, M^{'} \in \mathcal{M}$ and $M \neq M^{'}$, $f^{+}(h, M) = f^{+}(h, M')$ for some hash value $h \in \{0,1\}^n$.

It was a common assertion that the complexity of finding an r-multicollision in a random oracle is $\Theta(2^{n(r-1)/r})$. Nandi *et al.* [59] later observed that the true complexity is actually $\Theta(r \cdot 2^{n(r-1)/r})$. At Crypto 2004, Joux presented a seminal idea [37] to find multicollisions in an iterated hash function at a much lower computational complexity.

The basic idea of Joux's attack (illustrated in figure 4.1) is to find $\lg r$ successive collisions (by applying birthday attack) for the compression function, starting from a random hash value h_0. Suppose this process ends at h_k, then we have total $2^{\lg r} = r$ paths from h_0 to h_k (2 for each collision cycle), which gives us the required r-multicollision set. We will denote this algorithm by $\text{COLL}_{f(h_0,k)}$, which returns a 2^k-multicollision set.

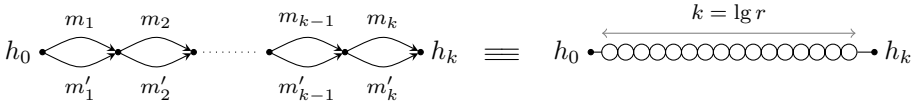

FIGURE 4.1: Joux multicollision structure and its shorthand representation in R.H.S.

Complexity Analysis. Each of the $\lg r$ birthday attacks cost $O(2^{n/2})$ computations, therefore, total computational complexity is $O(\lg r \cdot 2^{n/2})$. Note that, the algorithm does not need to describe all r messages, as it can be simply described by only $2 \lg r$ message blocks. Hence the memory complexity is $O(n \lg r)$. The message complexity is $O(2^{n/2})$ as it requires sampling of $O(2^{n/2})$ message blocks.

4.2.1.1 Application in Collision and Preimage Attacks on Concatenated Hash

Joux [37] showed that if one of the constituent hash function of the concatenated hash is following iterated hash design, then the collision and (second) preimage security of concatenated hash can be reduced to $2^{n/2}$ and 2^n from 2^n and 2^{2n} respectively[2].

Suppose H and G are the constituent hash functions and H is based on iterated hash. In the collision attack, the adversary first finds a $2^{n/2}$-multicollision set in H and then uses these $2^{n/2}$ messages to construct a birthday attack on G. This will give a collision pair in both H and G. The preimage attack requires a 2^n preimage set in H. This can be constructed by first finding a 2^n multicollision set and then linking the last hash value with the target hash value. The 2^n preimage set is then used to construct a random attack on G. This will give a preimage in both H and G.

Complexity Analysis. The computational complexity for the collision attack is $O(\frac{n}{2} \cdot 2^{n/2}) + O(2^{n/2}) \approx O(2^{n/2})$. Similarly, for the preimage attack it is $O(n \cdot 2^{n/2}) + O(2^n) \approx O(2^n)$. The memory complexity for the collision and preimage attacks is $O(n^2)$. The message complexity is $O(2^{n/2})$ in both the attacks.

4.2.2 Kelsey-Schneier Expandable Message

For a n-bit hash function H,

[2]For simplicity, we have assumed that both the constituent hash functions have n-bits hash output.

$\mathcal{M} = \{M_i : 0 \leq i \leq 2^k - 1\}$ is said to be a $(k, 2^k - 1)$-expandable message set if $\mathcal{M}$ is a 2^k-multicollision set and $\forall \ell \in [k, 2^k + k - 1], \exists M \in \mathcal{M}$ such that number of blocks in M is ℓ.

Kelsey and Shneier extended Joux's idea to get an expandable message set [1]. The basic idea is to exponentially (in power of 2) increase the no. of blocks in the lower arc of each iteration of a 2^k-Joux multicollision structure (see figure 4.2). This gives a set of 2^k colliding messages with lengths ranging from k to $2^k + k - 1$ blocks. We denote this algorithm as EXPMSG$_{f(h_0,k)}$ which returns a $[k\,, 2^k + k - 1]$-expandable message set.

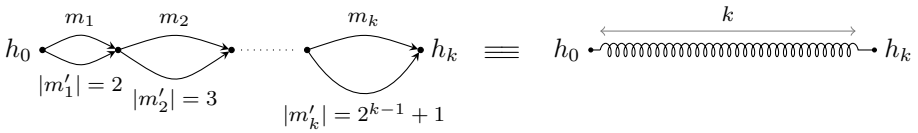

FIGURE 4.2: Kelsey Schneier expandable message and its shorthand representation in R.H.S.

Complexity Analysis. Computationally, the expandable message algorithm makes 2^k more f computations then the regular Joux multicollision algorithm. Hence, the computational complexity will be $O(2^k + k \cdot 2^{n/2})$. Note that a single random message block can be used to extend the arcs in the above algorithm. Hence, the memory complexity is $O(kn)$, as in Joux multicollision. The message complexity is $O(2^{n/2})$.

4.2.2.1 Application in Long Message Second Preimage Attack

Dean [24], in his thesis showed that the fixed points of a compression function can be used to construct a long message second preimage attack. But, this method was specific to constructions with fixed point property. Kelsey and Schneier [1] used the expandable message structure in place of the fixed point property to construct a long message second preimage attack on the iterated hash. Suppose $\overline{M}$ is a 2^l-blocks padded message and ω denotes the walk generated during the hash computation of $\overline{M}$. The basic idea is to hit any of the intermediate chaining values on ω from a random walk (starting from iv) ω'. This is done by trying 2^{n-l} random single message blocks after the walk ω'. By birthday attack, we can conclude that there will be a match with high probability. Say the match happens for message block x and the matching is at hash value h_i, then messages from $\omega' \| $the random message block $x \|$message blocks on ω after h_i gives the required second preimage. This algorithm gives correct second preimage if there is no length padding. But in case of length padding it may give incorrect results, as the length $|\omega'| + 1$ may not be equal to $|\omega_{1\ldots i}|$. To counter this the random walk ω' is constructed

using an expandable message set which helps in adjusting the number of blocks in ω' to accommodate length padding.

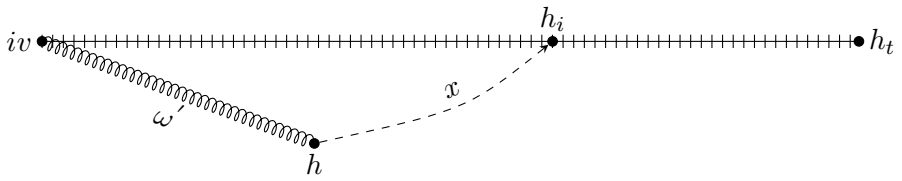

FIGURE 4.3: Kelsey Schneier long message second preimage attack [1].

Complexity Analysis. The computational complexity of the above algorithm can be analysed in three parts,

- First, note that the adversary has to compute the ω walk which will take $O(2^\ell)$ computations.
- Second, to get a length comparable to 2^ℓ a $[\ell, 2^\ell + \ell - 1]$-expandable message set will be required. This will take $O(2^\ell + \ell \cdot 2^{n/2})$ computations.
- Third, to hit the walk ω we need $2^{n-\ell+\lg \ell}$ computations.

Therefore, total computational complexity will be $O(2^\ell + \ell \cdot 2^{n/2} + 2^{n-\ell+\lg \ell})$. The message complexity in this algorithm will be $O(2^{n/2} + 2^{n-\ell+\lg \ell})$ and the memory complexity will be $O(2^\ell)$. So, we get an optimal attack for $\ell = \frac{n}{2}$.

4.2.3 Diamond Structure

A diamond structure [2] gives a different kind of multicollision set where each message starts from a different chaining value. A 2^k Diamond $\mathcal{D}$ (illustrated in figure 4.4) is a complete binary tree with 2^k leaf nodes, where each node represents a chaining value and the directed arc (h, h') with label m represents the transition $f(h, m) = h'$. The set of leaves and the root node of $\mathcal{D}$ will be denoted by $\mathcal{L}_\mathcal{D}$ and $h_\mathcal{D}$, respectively. The storage requirement for a 2^k diamond structure is $O(2^k)$.

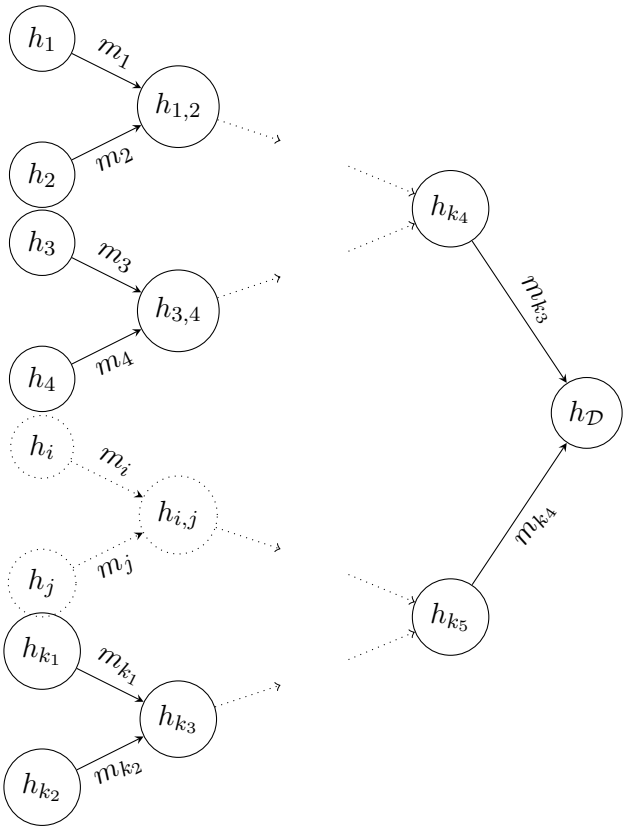

FIGURE 4.4: A 2^k converging diamond structure. A diverging diamond is similar with the direction of arcs reversed.

Based on the direction of arcs, we can have two different types of diamond structure:

1. *Converging Diamond* [2], where the direction of arcs is from the leaves to the root node. The computational complexity in building a 2^k converging diamond with forward queries is $O(n \cdot \sqrt{k} \cdot 2^{(n+k)/2})$ [38] and the message complexity is $O(2^{(n+k)/2})$. The detailed complexity analysis for this structure is a bit involved and beyond the scope of this thesis. For further exposition, please see [38]. If the computations are done on a strongly invertible function, then this structure can be built in $O(2^k)$ backward queries and the message complexity will reduce to $O(2^k)$. In this later case, this structure is also referred as *inverse diamond* [5]. We denote this algorithm as $\text{CONVDIAM}_f(\mathcal{H})$ for forward query construction and $\text{CONVDIAM}_{f^{-1}}(h_0, k)$ for backward query construction. In addition to the normal inputs, the subroutine can also take a message set $\mathcal{M}$ as input.

2. *Diverging Diamond*, where the direction of arcs is from the root node to the leaves. It takes $O(2^k)$ computations and $O(1)$ message blocks to build this structure using forward queries only. We denote this as subroutine $\text{DIVDIAM}_f(h, k, \mathcal{M})$ which returns a 2^k diverging diamond. DIVDIAM takes a chaining value h, a positive integer k and

a set of messages $\mathcal{M}$ as input. Note that, $\mathcal{M}$ can be null, in which case DIVDIAM will use random messages.

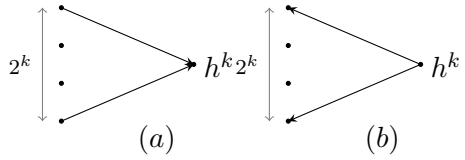

FIGURE 4.5: Shorthand for (a) 2^k-converging diamond, (b) 2^k-diverging diamond.

4.2.3.1 Application in Herding Attack

Kelsey and Kohno used the converging diamond to define a CTFP or herding attack [2] on the MD hash function. As illustrated in figure 4.6, the attack works as follows:

1. The adversary creates a 2^k-diamond structure $\mathcal{D}$ and commits $h_{\mathcal{D}}$ as target hash value. The challenger gives a prefix P (chosen uniformly) to the adversary.
2. The adversary computes the walk ω, corresponding to the prefix P. From ω the adversary tries to hit one of the chaining values in $\mathcal{L}_{\mathcal{D}}$ using 2^{n-k} random single message blocks.
3. There will be a match with high probability. Say the match happens for message block x and the matching is at hash value h_i, then $x\|$labels on the path from h_i to $h_{\mathcal{D}}$ gives the required suffix S.

Note that for simplicity we have not considered the length padding here. It can be easily incorporated in this attack by adding an expandable message set at the root of diamond.

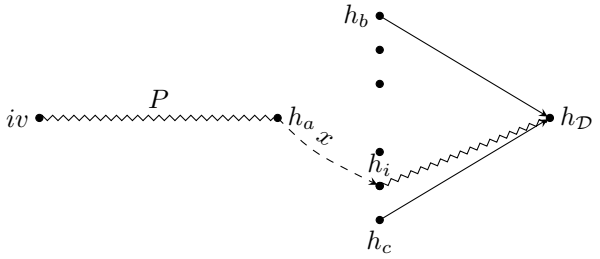

FIGURE 4.6: Herding attack on iterated hash using diamond structure [2].

Complexity analysis. This attack works in two phases. The computational complexity of the precomputation phase is $O(2^{(n+k)/2})$ and for online phase it is $O(2^{n-k})$. The message complexity is $O(2^{(n+k)/2})$ in the precomputation phase and $O(2^{n-k})$ in the online phase. The memory complexity is $O(2^k)$. For $k = \frac{n}{3}$, both precomputation and online complexity becomes $O(2^{2n/3})$.

4.2.4 Switch and Interchange structure

At Eurocrypt 2015, Leurent *et al.* have presented another interesting structure called the switch and interchange structure [3]. Suppose we have two n bits hash functions H and G simultaneously applied on a message M. We denote the walks for H and G as ω_j and α_k respectively. The individual chaining values are denoted as ω_j^i and α_k^ℓ. We denote the relation between the chaining values of H and G as:

$$(\omega_j^i, \alpha_k^i) \xrightarrow{m_i} (\omega_j^{i+1}, \alpha_k^{i+1})$$

Switch is a set of walks with the following property:

$$(\omega_{j_0}^i, \alpha_{k_0}^i) \xrightarrow{m_i} (\omega_{j_0}^{i+1}, \alpha_{k_0}^{i+1})$$

where m_i is the i^{th} substring (multiple message blocks) of M.

$$(\omega_{j_0}^i, \alpha_{k_0}^i) \xrightarrow{m_i^{'}} (\omega_{j_0}^{i+1}, \alpha_{k_1}^{i+1})$$

where $m_i^{'}$ is a secondary substring. That is it jumps from $(\omega_{j_0}^i, \alpha_{k_0}^i)$ to $(\omega_{j_0}^{i+1}, \alpha_{k_1}^{i+1})$ on a secondary substring $m_i^{'}$ (see figure 4.7). This property can be designed for both ω_j and α_k. The substrings m_i and $m_i^{'}$ are determined while building the switch. A switch structure can be built with a complexity of $O(2^{n/2})$ computations. By combining several simple switches, we can build an interchange structure with initial chaining values iv_1 and iv_2 and ending points $\{X_j, j = 0 \dots 2^k - 1\}$ and $\{Y_j, j = 0 \dots 2^k - 1\}$ (see figure 4.8). The strength of this structure lies in the fact that it provides a message ending in any state (X_j,Y_k) where X_j and Y_k are independent of each other. An interchange structure with 2^k walks per function requires about 2^{2k} switches. Therefore, the total structure can be built in $O(2^{2k+n/2})$.

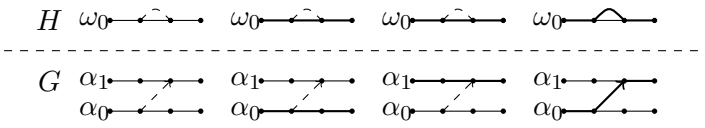

FIGURE 4.7: Switching from (ω_0, α_0) to (ω_0, α_1) by using $m^{'}$ (dashed lines) instead of m (solid lines) [3].

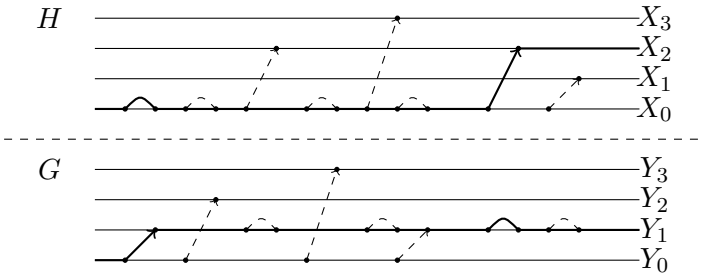

FIGURE 4.8: Navigating through the interchange structure to reach (X_2, Y_1) [3].

4.2.4.1 Application in Preimage attack on XOR Combiner

Leurent *et al.* further showed a direct application of switch and interchange structure by constructing a preimage attack on the XOR combiner in less than 2^n computational complexity [3]. Suppose the XOR combiner is using h and g compression functions, then the attack works as follows:

1. The adversary precomputes an interchange structure of 2^k size. Suppose the set of endpoints for the two hash functions be $\mathcal{X}$ and $\mathcal{Y}$.

2. When presented with a target hash value t, the adversary selects a random message block m and computes two lists as follows:

$$\mathcal{X}' = \{X_j' = h(X_j, m), X_j \in \mathcal{X}\}$$

and

$$\mathcal{Y}' = \{Y_k' = t \oplus g(Y_j, m), Y_j \in \mathcal{Y}\}.$$

3. A match between these two lists is expected with probability 2^{2k-n}. Therefore, after about 2^{n-2k} random choices of m we get a match (j^*, k^*) such that,

$$h(X_{j^*}, m) = t \oplus g(Y_{k^*}, m)$$

i.e.

$$h(X_{j^*}, m) \oplus g(Y_{k^*}, m) = t.$$

4. By choosing substring M corresponding to (X_j, Y_k) in the interchange structure, and concatenating it with block m the adversary gets the required preimage.

Complexity Analysis. The computational complexity for this attack can be analysed in two parts,

1. Building the interchange structure requires $O(2^{2k+n/2})$ computations.
2. The list collision requires $O(2^{n-k})$ computations.

Therefore, the total computational complexity of this attack is $O(2^{2k+n/2}) + O(2^{n-k})$. For $k = \frac{n}{6}$ the precomputation cost equals online cost.

Apart from the above structures there is yet another popular structure called the *kite structure* [38, 40]. The kite structure is a concatenation of a diverging and a converging diamond. It has been used to construct attacks on the dithered hash [40] and the concatenated hash [38].

4.3 Summary of Attacks on Some Iterated Hash Variants

In this section, we will summarise some of the attacks on hash twice, concatenated hash and zipper hash. Most of these attacks are due to Andreeva *et al.* [4], and the last attack, second preimage attack on zipper hash with strongly invertible compression functions, is due to Chen *et al.* [5]. Note that, all the attacks in this section are applicable for length padded messages (with minor modifications). So for simplicity we will ignore the length padding.

4.3.1 Herding Attack on Concatenated Hash

Andreeva *et al.* gave a herding attack on the concatenated hash design using a double pipe diamond (see [4] for detailed discussion). For compression functions f_1 and f_2, the complete attack (illustrated in 4.9) can be summarised as follows:

1. In the pre computation phase, the adversary computes a 2^ℓ converging diamond $\mathcal{D}_1$ using f_1 computations. Starting from $h_{\mathcal{D}_1}$, a $2^{n-\ell} + 2^{\ell \cdot \frac{n}{2}}$ Joux multicollision $\mathcal{J}$ is computed using f_1. The last $\ell \cdot \frac{n}{2}$ cycles of $\mathcal{J}$ are used to construct a 2^ℓ diamond structure $\mathcal{D}_2$ using f_2. The adversary commits $h \| g$ as the target hash value.
2. In the online phase, the adversary computes $h_a = f_1^+(iv_1, P)$ for the challenge prefix P. Then, she uses $O(2^{n-\ell})$ random message bocks to connect h_a to $\mathcal{D}_1$. Let the chain from h_a to $h_{\mathcal{D}_1}$ be ω. Then the adversary computes $g_a = f_2^+(iv_2, P \| \omega)$, and uses the first $n - \ell$ cycles of $\mathcal{J}$ to connect g_a to $\mathcal{D}_2$.

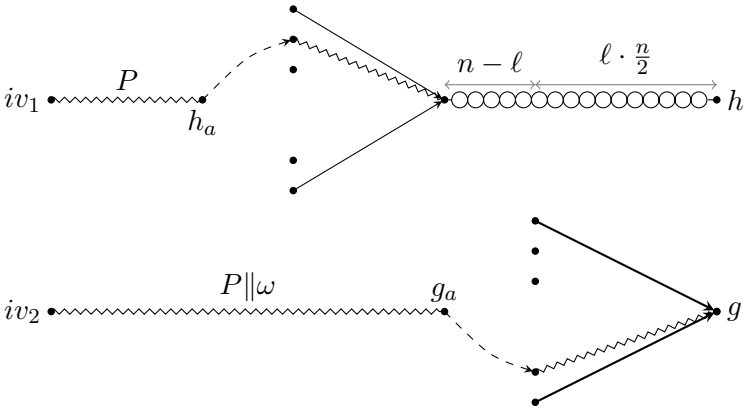

FIGURE 4.9: Herding attack on Concatenated Hash [4].

Complexity Analysis. The computational complexity of this attack can be analysed in two parts,

1. The precomputation cost is $O(2^{n/2} + 2^{\frac{n+\ell}{2}})$.
2. The online phase costs, $O(2^{n-\ell})$.

For $\ell = \frac{n}{3}$, this optimises to $O(2^{2n/3})$.

4.3.1.1 Extension to Herding Attack on Hash Twice

The herding attack on hash twice scheme by Andreeva *et al.* [4] is similar to their attack on concatenated hash. The only difference, here lies in the fact that here h_b works as iv_2 for f_2 (see figure 4.10). Also, the complexity analysis of this attack is similar to the previous attack.

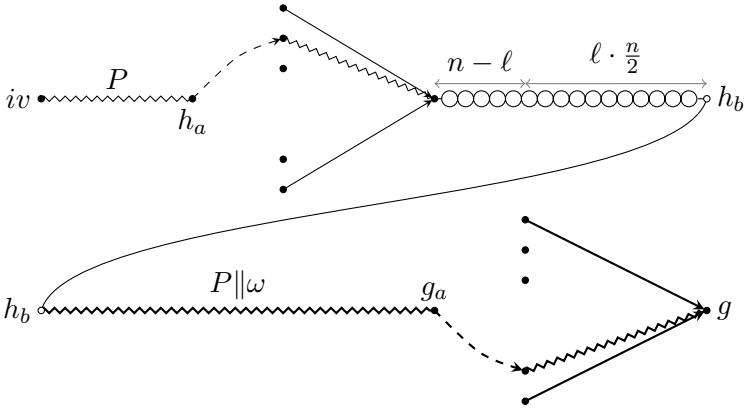

FIGURE 4.10: Herding attack on Hash Twice [4].

4.3.2 Second Preimage Attack on Hash Twice

The hash twice scheme is also susceptible to second preimage attacks, as shown by Andreeva *et al.* [4]. For compression functions f_1 and f_2, the attack can be described in two stages:

1. In the precomputation phase, the adversary creates a $[k, 2^k + k - 1]$ expandable message set $\mathcal{E}$, followed by a $(n - \ell + 2^{\frac{n\ell}{2}})$ Joux multicollision $\mathcal{J}$ using f_1. The last $\frac{n\ell}{2}$ cycles of $\mathcal{J}$ is used to construct a 2^ℓ converging diamond $\mathcal{D}$ using f_2.

2. In the online phase, the adversary tries to hit the walk generated by the challenged message from g_b using random single block messages. Say it hits at h_i, such that the walk from g_b to h is ω. Observe that, the adversary can now fix the message say P from $\mathcal{E}$ (initializing it to any message of length $i - n + \ell - \frac{n\ell}{2} - 1$). Then the adversary computes $h_b = f_1^+(h_a, \omega)$ and $g_a = f_2^+(h_b, P)$ and uses the first $n - \ell$ cycles of $\mathcal{J}$ to hit any leaf of $\mathcal{D}$. This completes the attack.

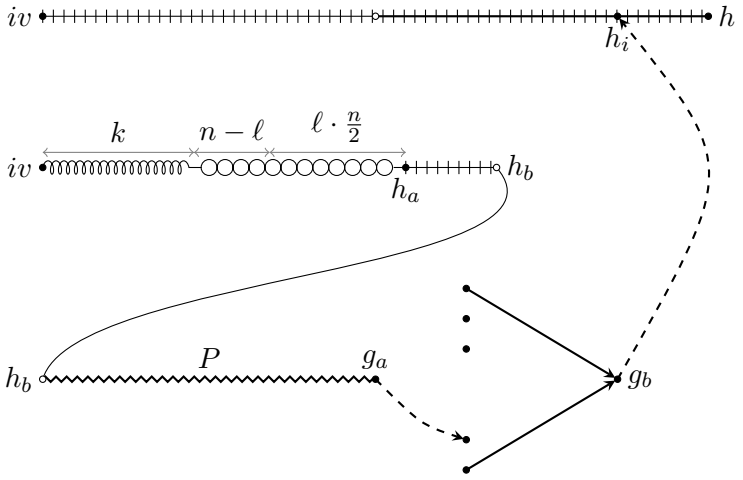

FIGURE 4.11: Second preimage attack on Hash Twice [4].

Complexity Analysis. The computational complexity of this attack can be analysed in two parts. For a $O(2^k)$ blocks message,

1. The precomputation phase takes $O(2^k + 2^{n/2} + 2^{\frac{n+\ell}{2}})$ computations, and

2. The online phase takes $O(2^{n-k} + 2^{n-\ell})$ computations.

For $k \leq \frac{n}{2}$ and $\ell = \frac{n}{3}$ the offline and online complexities become $O(2^{\frac{2n}{3}})$.

4.3.3 Existing Attacks on Zipper Hash

Zipper hash has been analysed with respect to various security properties. Andreeva *et al.* [4] have worked on the herding attack aspect. Bagheri [60] has presented a multicollision attack on single and multiple round zipper hash. Recently, Chen *et al.* [5] have presented a second preimage attack on zipper hash with strongly invertible compression functions. The multicollision attack is essentially a special case of the multicollision attacks by Nandi *et al.* [59]. We are briefly summarizing the other two attacks below:

4.3.3.1 Chosen Target Forced Suffix attack

The conventional herding attack (CTFP) is not viable for the zipper hash because last block of the second pass is the same as the first block of the first pass. So, an adversary that can construct CTFP attack can actually invert the compression function. Andreeva *et al.* [4] proposed a modified version of herding attack which works on forced suffix instead of forced prefix. For compression functions f_1 and f_2 the CTFS can be summarised as follows:

1. In the precomputation phase, the adversary will first create a $2^{\frac{n\ell}{2}+(n-\ell)}$ Joux multicollision set $\mathcal{J}$ for the first pass. Block-wise reverse of the first $\frac{n\ell}{2}$ cycles of $\mathcal{J}$ is then used to construct a 2^ℓ converging diamond structure $\mathcal{D}$ for the second pass. The adversary commits $h = h_{\mathcal{D}}$ as target hash value.

2. In the online phase, the adversary computes $h_b = f_1^+(h_a, S)$ and $h_c = f_2^+(h_b, S^r ev)$ for the challenge suffix S. The last $n - \ell$ cycles of $\mathcal{J}$ are used to connect h_c with one of the leaf of $\mathcal{D}$.

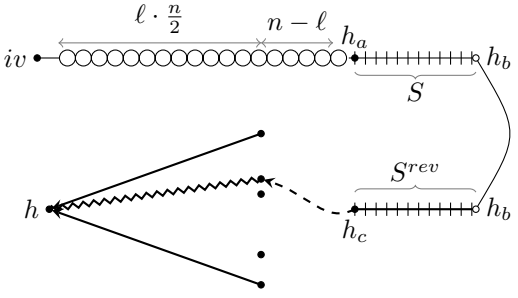

FIGURE 4.12: Herding Attack on Zipper Hash [4].

Complexity Analysis. The computational complexity is analysed in two phase:

1. The precomputation phase takes, $O(2^{\frac{n+\ell}{2}+2} + (n - \ell + \frac{n\ell}{2}) \cdot 2^{n/2})$, and

2. The online phase takes, $O(2^{n-\ell})$ compression function calls.

For $\ell = \frac{n}{3}$, the complexity optimises to $O(2^{2n/3})$.

4.3.3.2 Second Preimage Attack on Zipper Hash with Strongly Invertible f_1

If the underlying f_1 is strongly invertible then, we can construct a second preimage attack using a combination of Kelsey-Schneier long message attack in the second pass and meet-in-the-middle attack in the first pass. Chen *et al.* [5] have proposed an attack along similar lines. We are summarising their attack with a small variation:

1. Generate a 2^n Joux multicollision set $\mathcal{J}$ for the second pass. Split every n-blocks message $M \in \mathcal{J}$ into halves, and collect the first half $n/2$ blocks in set $\mathcal{M}_1$ and the other half $n/2$ blocks in $\mathcal{M}_2$.

2. Starting from h_m generate a $2^{n/2}$ converging diamond structure $\mathcal{D}_1$ for the first pass using the backward interface for f_1 and the block-wise reverse of messages $M_1 \in \mathcal{M}_1$.

3. Starting from h_a, generate a $[k, 2^k + k - 1]$-expandable message set $\mathcal{E}$ for the second pass.

4. Select at random a block m'_{i-1} until $f_2(h_b, m'_{i-1})$ hits a chaining value appearing in the second pass. Suppose it hits at position i.

5. Choose $M_e \in \mathcal{E}$ such that no. of blocks in $M_e = i - n - 1$.

6. Starting from iv compute $f_1^+(\overline{m}_{0,i}^{rev} || m'_{i-1} || M_e^{rev})$ to reach h_j.

7. Starting from h_j generate a $2^{n/2}$ diverging diamond structure $\mathcal{D}_2$ using f_1 computations and the block-wise reverse of messages $M_2 \in \mathcal{M}_2$.

8. Apply meet-in-the-middle attack on $\mathcal{L}_{\mathcal{D}_1}$ and $\mathcal{L}_{\mathcal{D}_2}$ to connect h_j and h_m.

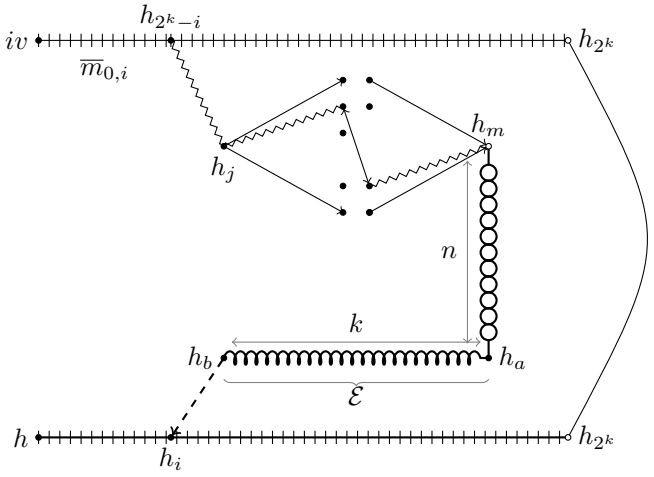

FIGURE 4.13: Second preimage attack over Zipper hash with strongly invertible compression functions [5].

Complexity Analysis. For a 2^k blocks message, the attack takes $O(2^{n/2}+2^{n-k}+2^k)$ computations (ignoring polynomial factors). This optimises to $O(2^{n/2})$ for a $k = \frac{n}{2}$.

Chapter 5

Some New Structures and Analysis of Concatenated Hash

In this chapter we are proposing two new structures, *chain* and *multi-pipe expandable message set*. The chain structure, in particular, is very interesting due to its simplicity. We will discuss one major application of this structure in bringing down the complexity of herding attack on iterated hash. Apart from the two structures, we will also present two attacks on the concatenated hash under certain assumptions on the underlying compression functions.

5.1 The Chain Structure

In essence, a chain is nothing but a walk generated by the iterated use of a compression function f repeatedly over a message block m, i.e., formally,

> A 2^k chain starting at h and generated by message block m can be defined as, the walk ω denoting $f^+(h, m^{2^k})$.

The structure is as powerful as it is simple in definition. We will show the potential of this structure by way of an example, the herding attack on iterated hash schemes.

5.1.1 Herding or chosen-target forced-prefix attack

Recall that the herding attack is a security game, in which the adversary has to first commit a target hash value h, and then find a suffix S for a challenge prefix P, such that $H(P\|S) = h$. This notion was first proposed by Kelsey and Kohno in [2].

In their attack, a diamond structure is precomputed to herd any prefix (of valid length) to the target hash value, in online phase. The diamond structure is a relatively costly structure [38] which requires $O(2^{\frac{n+k}{2}})$ computations to construct a 2^k diamond. So, the precomputation costs $O(2^{\frac{n+k}{2}})$ and the online complexity for herding any prefix to the target hash value is $O(2^{n-k})$. The memory complexity is $O(2^k)$, due to the diamond structure and the message complexity is $O(2^{\frac{n+k}{2}})$. The optimal computational complexity, i.e., $O(2^{2n/3})$ is achieved for $k = \frac{n}{3}$.

Here, we are presenting an attack with improved offline complexity and message complexity. The attack replaces the 2^k diamond structure with a 2^k chain structure. The complete algorithm is shown as procedure HERDING-ITERATED in algorithm 1 and illustrated in figure 5.1. The attack can be summarised as follows:

1. In the precomputation phase, the adversary computes a 2^k chain $\mathcal{C}$ starting from a random chaining value h_a and message block m. From h_b she computes a $[k, 2^k + k - 1]$ expandable message set $\mathcal{E}$ and commits the endpoint of $\mathcal{E}$, say h as the target hash value.

2. In the online phase, when the adversary is presented with a prefix P, she first computes $h_c = f^+(iv, P)$ and then tries to hit $\mathcal{C}$ using single block messages. In $O(2^{n-k})$ tries, she is expected to hit $\mathcal{C}$. Suppose the hit is at index i with message block x. Now, $\mathcal{E}$ is initialized to appropriate length and a message of that length , say $M_{\mathcal{E}}$ is used to return the required suffix $x \| m^{2^k - i + 1} \| M_{\mathcal{E}}$.

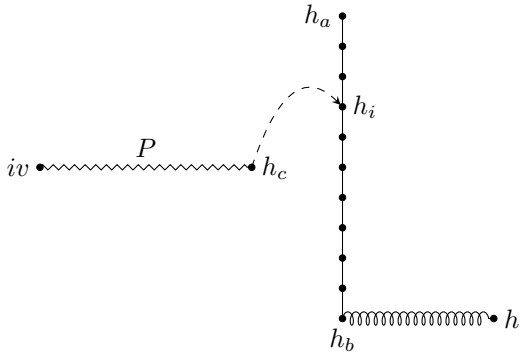

FIGURE 5.1: Herding iterated hash using chain structure.

Algorithm 1: Building a 2^k chain AND herding attack on iterated hash.

Procedure $\texttt{CHAIN}_f(h, k, m)$
- **Input**: $h \in \{0,1\}^n$
- **Input**: $k \in \mathbb{N}$
- **Input**: $m \in \{0,1\}^{n'}$
- **Output**: A 2^k chain and its endpoint h'.

1 $h' \leftarrow h$
2 $i \leftarrow 0$
3 **while** $i < 2^k$ **do**
4 $\quad \mathcal{C} \Leftarrow h'$
5 $\quad h' \leftarrow f(h', m)$
6 $\quad i \leftarrow i + 1$
7 **return** $(h', \mathcal{C})$

Interactive $\texttt{HERDING-ITERATED}_f()$
8 $h_a \xleftarrow{\$} \{0,1\}^n$
9 $m \xleftarrow{\$} \{0,1\}^{n'}$
10 $(h_b, \mathcal{C}) \leftarrow \texttt{CHAIN}_f(h_a, k, m)$;
11 $(h, \mathcal{E}) \leftarrow \texttt{EXPMSG}_f(h_b, k)$
- **Chosen Target**: h
- **Forced Prefix**: $P \in (\{0,1\}^{n'})^+$

12 $h_c \leftarrow f^+(iv, P)$
13 $(h_i, x) \leftarrow \texttt{HITTING}_f(h_c, \mathcal{C})$
14 $M_{\mathcal{E}} \leftarrow \mathcal{E}_i$
15 $S \leftarrow x \| m^{2^k - i + 1} \| M_{\mathcal{E}}$
16 **return** S

Complexity Analysis. The offline computational complexity is $O(2^k + 2^{n/2})$ due to the construction of the chain and expandable message set, and the online computational complexity is $O(2^{n-k})$. Observe that the complexity $O(2^k + 2^{n/2})$ of the offline phase of our attack is less than the complexity $O(2^{\frac{n+k}{2}})$ of the offline phase of earlier attack for $0 < k < n$.

In this attack, the optimal computational complexity of $O(2^{n/2})$ is achieved, for $k = \frac{n}{2}$, which is a significant improvement over the optimal computational complexity of $O(2^{2n/3})$, for $k = \frac{n}{3}$, achieved in the earlier attack.

Another significant improvement is in the message complexity where the complexity is reduced from $O(2^{\frac{n+k}{2}})$ to $O(1)$, as only a single block of message is required. Similarly, the offline memory complexity is reduced from $O(2^k)$ to $O(n)$.

We will also use the chain structure in our analysis of concatenated and zipper hash, and construct another structure called *rho* which uses the chain structure in principle. These applications will further emphasise the utility of the chain.

5.2 Multi-Pipe Expandable Message Set

A multi-pipe expandable message set is essentially an expandable message set but with an additional constraint that the expandable message set property should hold for multiple chaining value pairs. Formally,

> An ℓ-pipe is a set of ℓ 3-tuples of the form (h_a, h_b, f), such that $f^+(h_a, \overline{m}) = h_b$ for some message $\overline{m}$.

The ℓ-pipe settings are generally considered for same messages, i.e., the message $\overline{m}$ in the above definition is same for all of the ℓ tuples. For example, the concatenated hash evaluation, with underlying compression functions f and g, and initial values iv_1 and iv_2, over a message $\overline{m}$ can be viewed as a 2-pipe (or double-pipe), $\{(iv_1, f^+(iv_1, \overline{m}), f), (iv_2, g^+(iv_2, \overline{m}), g)\}$. Note that, the definition does not limit the use of same compression function in multi-pipe setting. Now, it is easy to define the multi-pipe expandable message set as,

> An ℓ-pipe $[a, b]$ expandable message set is an $[a, b]$ expandable message set for each of the ℓ pipes.

In other words, a multi-pipe expandable message set can simultaneously provide us with expandable message set for multiple chaining value pairs. This can be useful in constructing attacks on multi-pass schemes such as the zipper hash and hash twice. This property may also be useful in the analysis of hash combiners.

Before indulging ourselves in the construction of the multi-pipe structure observe that similar notion exists for Joux multicollision. For example, to get a single collision on double-pipe (see figure 5.2), we need $2^{n/2}$ multicollisions on the first pipe so that we get $2^{n/2}$ messages to get a single collision on the second-pipe, i.e., a single collision on double-pipe will have $\frac{n}{2}$ number of blocks in each arc.

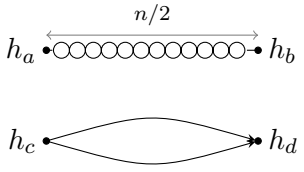

FIGURE 5.2: Single collision in double-pipe.

Therefore, to get a single double-pipe collision, we need $O(\frac{n}{2} \cdot 2^{n/2})$ computations. It is easy to observe that finding 2^k double-pipe multicollisions require $O(\frac{nk}{2} \cdot 2^{n/2})$ computations (k single double-pipe collisions). Now, we will generalise this to ℓ pipes using the following lemma.

Lemma 5.1. *The computational complexity of constructing a 2^k ℓ-pipe multicollisions is*

$$O((\frac{n}{2})^{\ell-1} \cdot k \cdot 2^{n/2})$$

.

Proof. The proof is by induction on the number of pipes. The base case of $\ell = 1$ is trivially true. Suppose the lemma is true for ℓ, then we will prove it for $\ell + 1$. First, observe that to get a $2^{n/2}$ ℓ-pipe multicollisions we need

$$O((\frac{n}{2})^{\ell-1} \cdot \frac{n}{2} \cdot 2^{n/2}) = O((\frac{n}{2})^{\ell} \cdot 2^{n/2})$$

computations (by inductive hypothesis). Now, since we have $2^{n/2}$ multicollision messages in ℓ pipes, therefore, we can have a single collision in $\ell + 1$ pipes (by birthday attack). So, a single $(\ell + 1)$-pipes multicollision requires $O((\frac{n}{2})^{\ell} \cdot 2^{n/2})$. By applying it k times we will get 2^k $(\ell + 1)$-pipes multicollisions. Hence, the total computational complexity is

$$O((\frac{n}{2})^{\ell} \cdot k \cdot 2^{n/2}),$$

which proves the claim. The result follows. □

Now, we will extend this idea to a single pipe expandable message set. Suppose, while constructing a 2^k expandable message set, we insert a $2^{n/2}$ Joux multicollision after each cycle (illustrated in figure 5.3) of the expandable message set. Now, we will apply these messages on another pipe in the following manner,

1. For cycles from expandable message set, construct two separate walks α and β.
2. Use the intermediate $2^{n/2}$ Joux multicollision to collide α and β.

This will give a $[\frac{nk}{2} + k, 2^k + \frac{nk}{2} + k - 1]$ double-pipe expandable message set.

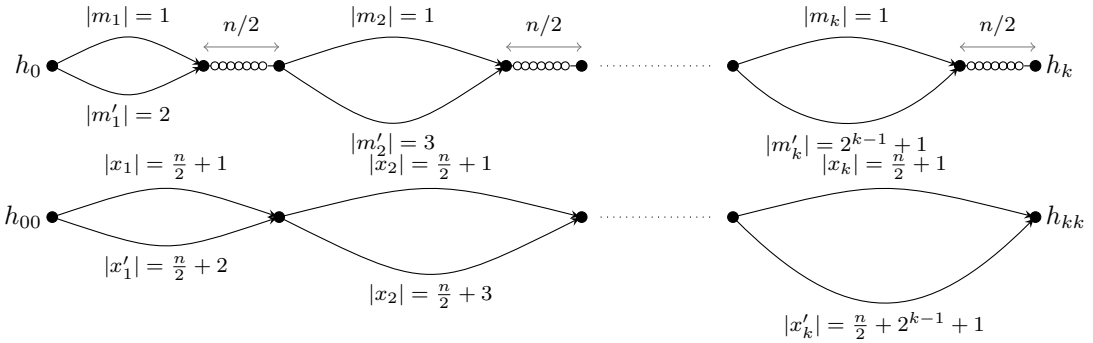

FIGURE 5.3: A $[\frac{nk}{2} + k, 2^k + \frac{nk}{2} + k - 1]$ double-pipe expandable message set.

Clearly it takes $O(2^k + (\frac{nk}{2} + k) \cdot 2^{n/2})$ computations to construct a $[\frac{nk}{2} + k, 2^k + \frac{nk}{2} + k - 1]$ double-pipe expandable message set. Again, this can be generalised to ℓ-pipes, where the computational complexity follows from the next theorem.

Theorem 5.2. *The computational complexity of constructing a* $[(\frac{n}{2})^{\ell-1} \cdot k + k, 2^k + (\frac{n}{2})^{\ell-1} \cdot k + k - 1]$ *ℓ-pipe expandable message set is*

$$O(2^k + ((\frac{n}{2})^{\ell-1} + 1) \cdot k \cdot 2^{n/2})$$

.

Proof. A formal proof for this theorem can be given on similar lines as in lemma 5.1. Note that we are still performing the computations required for $[k, 2^k + k - 1]$ expandable messagse set in single pipe setting. Add to this the complexity for constructing a 2^k ℓ-pipes multicollision and we get the required complexity. The result follows. □

For double-pipe, we will denote this construction as a set of two algorithms DPIPE-ONE-EXPMSG and DPIPE-TWO-EXPMSG. The first of these will be used to construct the multicollision appended expandable message set (structure with black arcs in figure 5.3), and the second will be used to construct the double pipe expandable message using the output set of first algorithm.

5.3 Analysis of Concatenated Hash

In this section, we will present two attacks on weaker version of concatenated hash. The first attack is a second preimage attack on concatenated hash with a strongly invertible component and the second attack is a preimage attack on concatenated hash with one weakly and one strongly invertible component.

5.3.1 Second Preimage Attack on Concatenated Hash

Here we assume that one component of the concatenated hash is MD hash and the other component is strongly invertible. Under these assumptions, we can construct a second preimage attack on the concatenated hash as follows:

1. For the MD hash component, start from iv_1 and compute a 2^n Joux multicollision set $\mathcal{J}$. From the endpoint of $\mathcal{J}$ construct a Kelsey-Schneier long-message second preimage attack on the rest of the chain. Together with the multicollision set, this step gives 2^n second preimages for the MD component.

2. For the strongly invertible component, divide $\mathcal{J}$ into two sets $\mathcal{M}_1$ and $\mathcal{M}_2$ of $\frac{n}{2}$ blocks each. Invert the strongly invertible component on the second preimage

string found in step 1 till h_i. Say this inversion ends at g_a. Now, construct a meet-in-the-middle attack between iv_2 and g_a using $\mathcal{M}_1$ and $\mathcal{M}_\in$ respectively to get the valid second preimage message.

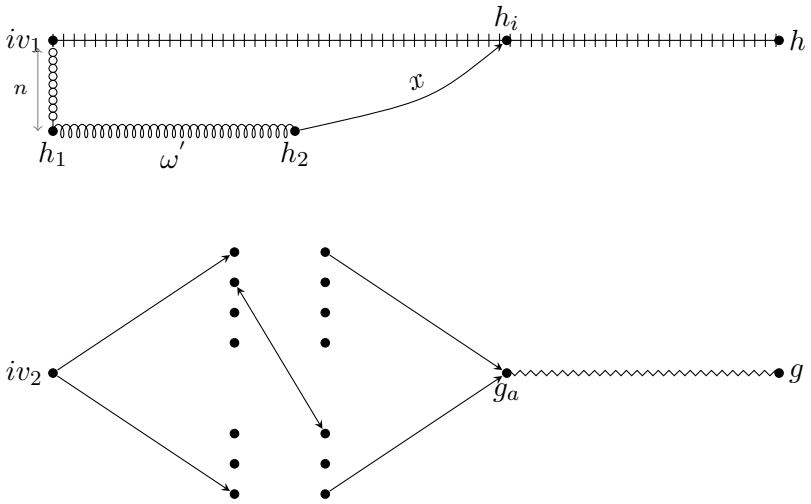

FIGURE 5.4: Second preimage attack on concatenated hash.

Complexity Analysis. The computational complexity for this algorithm follows from the Kelsey-Schneier attack complexity, i.e., $O(2^{n/2})$ in optimal case. The message and memory complexity are similar to those in the Kelsey-Schneier attack.

5.3.2 Preimage Attack on Concatenated Hash

This attack assumes that one of the component of concatenated hash is weakly invertible and the other one is strongly invertible. The attack as illustrated in figure 5.5 can be summarised as follows:

1. For the weakly invertible component, we start at iv_1 and compute a 2^n Joux multicollision set $\mathcal{J}$. Divide $\mathcal{J}$ into two sets $\mathcal{M}_1$ and $\mathcal{M}_2$ of $\frac{n}{2}$ blocks each.

2. From the target hash value compute $2^{n/2}$ random chaining values using the weakly invertible interface, and compute $2^{n/2}$ random chaining values from h_a using forward queries. Use these two sets of $2^{n/2}$ chaining values to compute a 2-block message x between h_a and h.

3. Invert the strongly invertible component on x from g and then use $\mathcal{M}_1$ from iv_2 and $\mathcal{M}_2$ from g_a to construct a meet-in-the-middle attack. The walk from iv_2 to g will give a preimage in both the components, and hence in the concatenated hash.

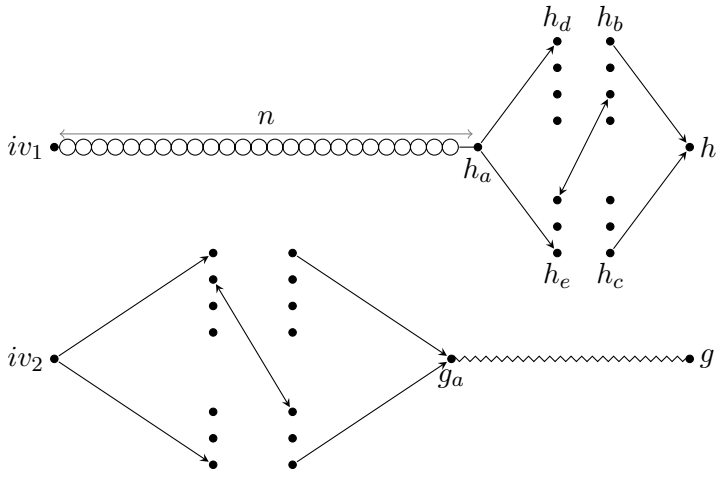

FIGURE 5.5: Preimage attack on concatenated hash.

Complexity Analysis. The complexity of this algorithm can be analysed in a similar way as in 5.3.1 and it gives similar bounds of $O(2^{n/2})$. Note that, this attack is a minor improvement over the attack suggested by Leurent *et al.* in [3]. The earlier attack works only when both the components are strongly invertible, whereas, our attack even works for a relaxed condition. This also shows that the bound computed by Hoch *et al.* in [41] is also applicable when one of the components is weakly invertible.

Chapter 6

Analysis of Zipper Hash

In this chapter, we are presenting second preimage, preimage and herding attacks on the zipper hash design. The attacks on general zipper hash requires a precomputed structure of $O(n)$ space. We refer this structure as *Rho structure*. We will start off with a detailed discussion on the construction of this structure. The second preimage attack on relaxed zipper hash requires the construction of a double pipe expandable message set and a chain structure (as discussed in chapter 5). Note that, all the attacks in this section are equally applicable in the presence of padding rules also. So, for the sake of simplicity, we will be ignoring the padding rules here. Instead, we will explain the modifications required for incorporating padding.

6.1 Rho Structure

A 2^k-rho structure is a rho-shaped structure with an unit cycle length, i.e., a self loop and a tail of size 2^k denoted by $\mathcal{T}_\rho$. Moreover the labels for the tail as well as the self-loop are the same (which is m in Fig. 6.1).

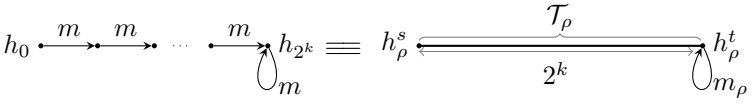

FIGURE 6.1: Rho structure and its shorthand representation.

The Rho structure is the core of our (second) preimage attacks on the zipper hash. Based on the properties of the underlying compression functions, it can be constructed in two ways:

6.1.1 Under Random Oracle Assumption

When the underlying compression functions f_1 and f_2 behave as random oracle without any weaker interface, the rho structure can be constructed as follows:

1. In the first phase, a fixed point chaining value h and the corresponding message block m is found for the given compression function. This is shown as subroutine FIXED-POINT in Algo. 2.

2. In the second phase, the aim is to construct a 2^k-blocks chain to the fixed point found above using the block m repeatedly. This is done by choosing a random chaining value, and repeatedly applying m. In at most 2^n computations, it is expected that the chain will hit h[1].

We denote the algorithm to construct this structure as RHO(k) which returns h^s_ρ, the *start of rho*, h^t_ρ, the *tip of rho*, and the *rho label* m_ρ. The rho structure can be extended, in which case, a 2^k long chain is computed from the tip of rho.[2] We will be using this extended version of rho in our second preimage attack.

Algorithm 2: Building a 2^k-rho structure.

Algorithm $\texttt{RHO}_{f,g}(k)$
Input: $k \in \mathbb{N}$
Output: A 2^k Rho, ρ.
1 $(h_{\rho^t}, m) \leftarrow \texttt{FIXED-POINT}_f()$
2 $h \xleftarrow{\$} \{0,1\}^n$
3 $i \leftarrow 0$
4 $h' \leftarrow h$
5 **while** (1) **do**
6 $h' = g(h', m)$
7 $i \leftarrow i + 1$
8 **if** $i = 2^k$ **then**
9 **if** $h' = h^t_\rho$ **then**
10 $h^s_\rho \leftarrow h$
11 **return** $\rho : (h^s_\rho, h^t_\rho, m)$
12 **else**
13 $h \leftarrow g(h, m)$
14 $i \leftarrow i - 1$

Procedure $\texttt{FIXED-POINT}_f()$
Output: A fixed point pair, (h, m).
15 $h \xleftarrow{\$} \{0,1\}^n$
16 **while** (1) **do**
17 $m \xleftarrow{\$} \{0,1\}^{n'}$
18 **if** $f(h, m) = h$ **then**
19 **return** (h, m)

Complexity Analysis. Both the FIXED-POINT algorithm and the RHO algorithm takes $O(2^n)$ computations. So, the computational complexity of this algorithm is $O(2^n)$. The

[1]Note that, we are ignoring the cases in which either h is reached before 2^k or there is a cycle in the path. In these cases, one can start with a new node to reach already obtained nodes from which the tip node is reachable.

[2]Note that, this requires either a message block other than m_ρ or a different compression function computation.

algorithm requires $O(n)$ memory for book-keeping. Since about 2^n single block messages are tried in FIXED-POINT algorithm, the message complexity is $O(2^n)$.

6.1.2 Under Random Oracle with Multiple Fixed Points Assumption

A compression function is said to have multiple fixed points weakness, if computing multiple fixed points for a given message block is easy. Some of the insecure PGV schemes [49] have this property. We do not have knowledge about any secure scheme exhibiting this property, but under this weak assumption the rho structure can be constructed in less than 2^n computations. The main steps can be summarised as follows:

1. The fixed point algorithm MULTI-FIXED-POINT queries a multiple fixed point oracle internally. It will return a list $\mathcal{L}$ of 2^k fixed points and corresponding message block m for input k.

2. The rho construction algorithm MULTI-RHO applies the 2^k chain construction technique of RHO algorithm. But here the process stops at 2^k steps. Suppose the set of endpoints of these chains be $\mathcal{C}$. The algorithm then applies list collision between $\mathcal{L}$ and $\mathcal{C}$ to find the required tip of rho from $\mathcal{L}$ and the tail of rho from $\mathcal{C}$.

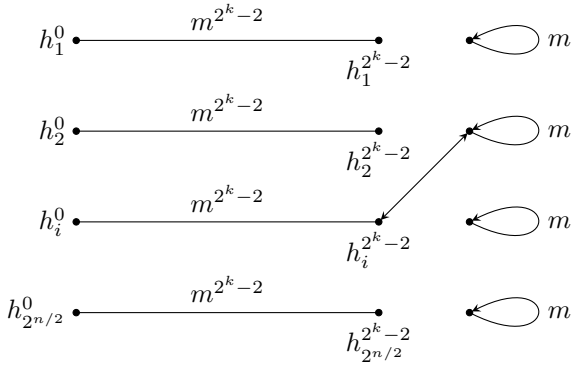

FIGURE 6.2: Illustration of rho construction in multiple fixed points case.

Algorithm 3: Building a 2^k-rho structure in multiple fixed points case.

Algorithm MULTI-RHO$_{f,g}(k)$
Input: $k \in \mathbb{N}$
Output: A 2^k Rho, ρ.
1 $(\mathcal{L}, m) \leftarrow$ MULTI-FIXED-POINT$_f()$
2 $i \leftarrow 0$
3 **while** $i < 2^{n/2}$ **do**
4 $h \xleftarrow{\$} \{0,1\}^n$
5 $\mathcal{C}_1 \Leftarrow h$
6 $\mathcal{C}_2 \Leftarrow h$
7 $i \leftarrow i + 1$
8 $i, j \leftarrow 0$
9 **while** $i < 2^k$ **do**
10 **while** $j < 2^{n/2}$ **do**
11 $h_j \Leftarrow \mathcal{C}_2^j$
12 $h_j \leftarrow g(h_j, m)$
13 $\mathcal{C}_2^j \Leftarrow h_j$
14 $j \leftarrow j + 1$
15 $i \leftarrow i + 1$
16 $(i, j) \leftarrow$LISTCOLL$(\mathcal{L}, \mathcal{C}_2))$
17 **return** $\rho : (\mathcal{C}_1^j, \mathcal{L}^i, m)$

Procedure MULTI-FIXED-POINT$_f()$
Output: A message block m and a $2^{n/2}$ list of fixed points $\mathcal{L}$ for m, $(\mathcal{L}, m)$.
18 $m \xleftarrow{\$} \{0,1\}^{n'}$
19 $i \leftarrow 0$
20 **while** $i < 2^{n/2}$ **do**
21 $h \leftarrow \mathcal{O}^f$
22 $\mathcal{L} \Leftarrow h$
23 $i \leftarrow i + 1$
24 **return** $(\mathcal{L}, m)$

Complexity Analysis. The computational complexity for the MULTI-FIXED-POINT algorithm is $O(2^{n/2})$ and MULTI-RHO algorithm takes $O(2^{k+\frac{n}{2}})$ computations. So, the computational complexity of this algorithm is $O(2^{k+\frac{n}{2}})$, which is less than 2^n for $k < \frac{n}{2}$. The algorithm requires $O(2^{n/2})$ memory for storing $\mathcal{L}$, $\mathcal{C}_1$ and $\mathcal{C}_2$. Since only a constant number of single block messages are tried in MULTI-FIXED-POINT algorithm, the message complexity is $O(1)$.

6.2 Second Preimage Attacks

The problem of a long-message second preimage attack on the zipper hash construction is to the best of our knowledge an open problem. Though, Chen *et al.* showed a second preimage attack on the zipper hash design with weaker compression functions [5], but the zipper hash based on compression functions under random oracle model has seen very less analysis. The forced-suffix attack by Andreeva *et al.* [4] is the only substantial attack on this design. The difficulty lies in three facts.

1. First, f_1 and f_2 are independent, i.e., the two pass scheme can use completely independent compression functions for the two passes.

2. Second, the message blocks which are processed last in the first pass are processed first in the second pass.

3. Third, and a bit easier problem is to accommodate the length padding operations.

These three facts compel the adversary to fix the meeting point of first and second pass operations in such a way that it does not violate the message block symmetry and incorporates the padding rule. This makes it hard to apply known attack techniques and structures[1, 2, 39, 40], which generally work for iterated hash designs and its variants. We are presenting here two attacks, one on the zipper hash and another on the relaxed variant of zipper hash ($f_1 = f_2$). Both these attacks require less than $O(2^n)$ online computational complexity.

6.2.1 Second Preimage Attack on Zipper Hash

The second preimage attack requires a one-time $O(2^n)$ precomputation. This is similar to the TMTO attack technique (as discussed in chapter 4). The complete algorithm is given as procedure SECOND-PREIMAGE in algorithm 4, and illustrated in figure 6.3. We are summarising the main points below:

1. In the precomputation phase, a 2^k rho structure is computed using f_1 computations.
2. In the online phase, the padding block is applied after the tip of rho and the structure is extended using f_2 computations.
3. From the endpoint h_a of the extension, a 2^{n-k} Joux multicollision $\mathcal{J}$ is constructed using f_2 computations.
4. From h_b a $[t, 2^t + t - 1]$ expandable message set $\mathcal{E}$ is computed. This takes care of the length padding.
5. From h_c hitting technique is used to compute a one block connection to the second pass chain. This fixes the message length for $\mathcal{E}$ (set to $i - 2^k - (n - k) - 1$).
6. Suppose the walk from h_b to h is ω and $h_d = f_1^+(iv, \omega^{rev})$. At last, the messages from $\mathcal{J}$ are used in block-wise reverse fashion to get a link with $\mathcal{T}_\rho$.
7. The walk from iv to h_ρ^t gives the required second preimage message.

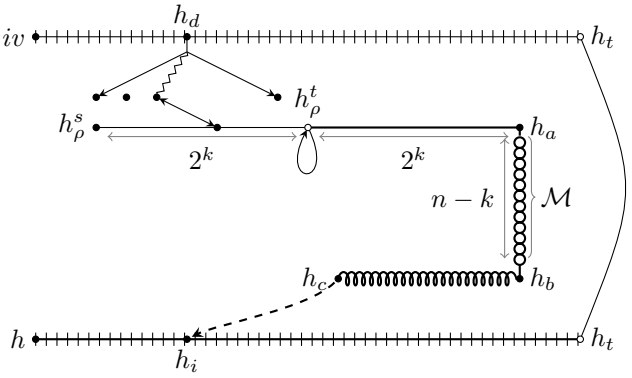

FIGURE 6.3: Second preimage attack on Zipper hash.

Algorithm 4: Second preimage attack on Zipper and Relaxed Zipper hash.

Algorithm $\texttt{SECOND-PREIMAGE}_{f_1,f_2}(\overline{m})$
Input: $\overline{m} \in (\{0,1\}^{n'})^+$ of 2^t blocks.
Output: $\overline{m'} \in (\{0,1\}^{n'})^+$
Data: 2^k-Rho, ρ.
1 $(\mathcal{L}_1, \mathcal{L}_2) \leftarrow \texttt{ZIPPER}_{f_1,f_2}(iv, \overline{m})$
2 $h_a \leftarrow f_2^+(h_\rho^t, m_\rho^{2^k})$
3 $(h_b, \mathcal{J}) \leftarrow \texttt{COLL}_{f_2}(h_a, n-k)$
4 $(h_c, \mathcal{E}) \leftarrow \texttt{EXPMSG}_{f_2}(h_b, t)$
5 $(m, h_i) \leftarrow \texttt{HITTING}_{f_2}(h_c, \mathcal{L}_2)$
6 $M_\mathcal{E} \leftarrow \mathcal{E}_{(i-2^k-(n-k)-1)}$
7 $h_d \leftarrow f_1^+(iv, \overline{m}_{2^t-i}||m||M^{rev})$
8 $\mathcal{D} \leftarrow \texttt{DIVDIAM}_{f_1}(h_d, k, \mathcal{J}^{rev})$
9 $(h_j, M_j) \leftarrow \texttt{LISTCOLL}(\mathcal{L}_\mathcal{D}, \mathcal{T}_\rho)$
10 $\overline{m'} \leftarrow \overline{m}_{2^t-i}||m\|M_\mathcal{E}^{rev}\|M_j\|m_\rho^{2^k-1}$
11 **return** $\overline{m'}$

Algorithm $\texttt{RELAXED-SECOND-PREIMAGE}_{f_1,f_2}(\overline{m})$
Input: $\overline{m} \in (\{0,1\}^{n'})^+$
Output: $\overline{m'} \in (\{0,1\}^{n'})^+$
12 $(\mathcal{L}_1, \mathcal{L}_2) \leftarrow \texttt{ZIPPER}_{f_1,f_2}(iv, \overline{m})$
13 $h_a \xleftarrow{\$} \{0,1\}^n$
14 $m \leftarrow \overline{m}_{pad}$
15 $\mathcal{C} \leftarrow \texttt{CHAIN-VARIANT}_{f_1}(h_a, k, m)$
16 $(h_c, \mathcal{J}) \leftarrow \texttt{COLL}_{f_2}(h_b, n-k)$
17 $(h_d, \mathcal{DE}_1) \leftarrow \texttt{DPIPE-ONE-EXPMSG}_{f_2}(h_c, k)$
18 $(h_e, \mathcal{E}) \leftarrow \texttt{EXPMSG}_{f_2}(h_d, t)$
19 $(m', h_i) \leftarrow \texttt{HITTING}_{f_2}(h_e, \mathcal{L}_2)$
20 $M_\mathcal{E} \leftarrow \mathcal{E}_{(i-2^k-\frac{nk}{2}-n-1)}$
21 $h_f \leftarrow f_1^+(iv, \overline{m}_{2^t-i}\|m'\|M_\mathcal{E}^{rev})$
22 $(h_g, \mathcal{DE}_2) \leftarrow \texttt{DPIPE-TWO-EXPMSG}_{f_2}(h_f, k, \mathcal{DE}_1)$;
23 $\mathcal{D} \leftarrow \texttt{DIVDIAM}_{f_1}(h_g, k, \mathcal{J}^{rev})$
24 $(h_j, M_j) \leftarrow \texttt{LISTCOLL}(\mathcal{L}_\mathcal{D}, \mathcal{P})$
25 $M_{\mathcal{DE}} \leftarrow \mathcal{DE}_{(2^k+\frac{nk}{2}+k-\frac{j}{2})}$
26 $M_\mathcal{C} \leftarrow m^{\frac{2^{k+1}-j-2}{2}}$
27 $\overline{m'} \leftarrow \overline{m}_{2^t-i}\|m'\|M_\mathcal{E}\|M_j\|M_{\mathcal{DE}}\|M_\mathcal{C}$
28 **return** $\overline{m'}$

Complexity Analysis. Clearly, the precomputation phase costs, $O(2^n)$ in a random oracle model and $O(2^{n/2+k})$ in multiple fixed points oracle. Assuming $k \leq \frac{n}{2}$, and a 2^t blocks message (where $t = O(k)$), the overall online complexity is $O(2^{n-k})$. For $k = \frac{n}{2}$ this optimises to $O(2^{n/2})$. The memory complexity for this algorithm is $O(2^k)$ and the message complexity is $O(2^{n/2})$. Note that, in the above algorithm as the message size increases beyond $O(2^{n/2})$, the complexity increases.

6.2.2 Second Preimage Attack on Relaxed Zipper Hash

Recall that a relaxed zipper hash is zipper hash with f_1 and f_2. This attack utilises two structures, namely the chain and the double pipe expandable message set, as introduced in the previous chapter. It uses a variant of the chain in which we only store the even indices. This algorithm is denoted by procedure CHAIN-VARIANT$_f(h, k, m)$ as shown in algorithm 5. The attack is similar to the previous one with some critical changes in the handling of length padding. The complete attack algorithm is shown as procedure RELAXED-SECOND-PREIMAGE in algorithm 4 and illustrated in 6.4. The main points of the attack can be summarised as:

1. First, a $2^{k+1}-1$ chain structure is computed using the padding block of $\overline{m}$, storing the even indices in $\mathcal{C}$.

2. From the endpoint h_b of $\mathcal{C}$, a 2^{n-k} Joux multicollision $\mathcal{J}$ is constructed.

3. From the endpoint h_c of $\mathcal{J}$ first pipe $\mathcal{DE}_1$ of a $[\frac{nk}{2}+k, 2^k+\frac{nk}{2}+k-1]$ double pipe expandable message $\mathcal{DE}$ is constructed and from its endpoint h_d a $[t, 2^t+t-1]$ expandable message set $\mathcal{E}$ is constructed.

4. From the endpoint h_e of $\mathcal{E}$ hitting technique is used to compute a one block connection to the second pass chain. This fixes the message length from $\mathcal{E}$ (set to $i-2^k-\frac{nk}{2}-n-1$).

5. Suppose the walk from h to h_e is ω and $h_f = f_1^+(iv, \omega^{rev})$. From h_f the messages in $\mathcal{DE}_1$ are used to construct the second pipe $\mathcal{DE}_2$ of the double pipe expandable message set $\mathcal{DE}$. This will help in compensating for the length uncertainty produced by the chain collision in next step.

6. Now, the messages from $\mathcal{J}$ are used in block-wise reverse fashion to get a matching with $\mathcal{C}$. This fixes the lengths of messages in $\mathcal{DE}_1$ and $\mathcal{DE}_2$ (set to $\frac{nk}{2}+k+\frac{j}{2}$), for matching at index j in $\mathcal{C}$). The midpoint from index j to 2^{k+1} is shown as h_m.

7. The walk from iv to index h_m in the chain structure gives the required second preimage message.

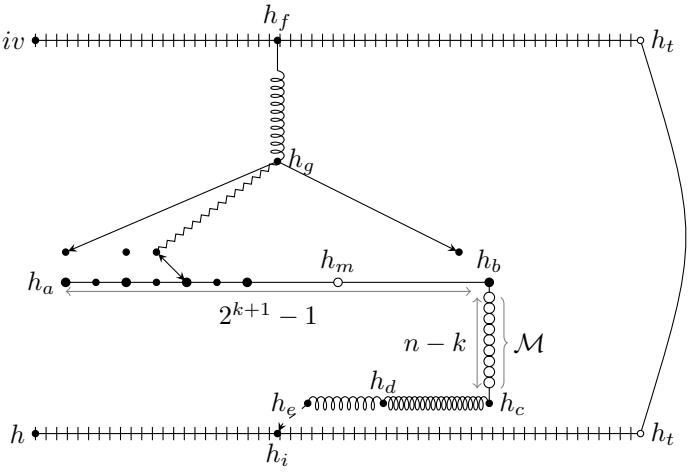

FIGURE 6.4: Second preimage attack on Relaxed Zipper Hash.

Complexity Analysis. Note that, the precomputation phase is not necessary for this algorithm. The computational complexity can be analysed in four parts,

1. The chain structure takes $O(2^k)$ computations.
2. The double pipe expandable message set takes $O(2^k + (\frac{nk}{2} + k) \cdot 2^{n/2})$ computations.
3. The expandable message set $\mathcal{E}$ and the Joux multicollision cost $O(2^t + t \cdot 2^{n/2} + 2^{n-k})$.
4. The hitting process takes $O(2^{n-k})$ computations.

For $t = O(k)$ and $k = \frac{n}{2}$ the overall complexity becomes $O(2^{n/2})$. The memory complexity is $O(2^k)$ and the message complexity is $O(2^{n/2})$. As in 6.2.1, the complexity increases as the message size increases beyond $O(2^{n/2})$.

Algorithm 5: Building a variant of chain of $2^{k+1} - 1$ length using m repeatedly.

Procedure `CHAIN-VARIANT`$_f(h, k, m)$
Input: $h \in \{0,1\}^n$
Input: $k \in \mathbb{N}$
Input: $m \in \{0,1\}^{n'}$
Output: A 2^k list of chaining values at 2 block difference.

```
1  h' ← h
2  i ← 1
3  C ⇐ h'
4  while i < 2^k do
5      h' ← f(h', m^2)
6      P ⇐ h'
7      i ← i + 1
8  return C
```

6.3 Preimage Attacks

The preimage security of zipper hash has not been studied up until now. With an assumption, that the finalisation function is identity, we are presenting three preimage attacks on the zipper hash and its relaxed variants. The first two attacks require a weakly invertible f_2, and the third attack requires a strongly invertible f_2 (see chapter 2 for notion of weakness).

6.3.1 Using Weak Invertibility of f_2

Recall that a compression function f is said to have weak invertible if given $h^{'}$ it is easy to compute (h, m) pairs such that $h^{'} = f(h, m)$. The attacks in this subsection may not work for padded message, as the adversary has no control whatsoever on the preimages returned by f^{-1}.

6.3.1.1 Preimage Attack on Zipper Hash

The preimage attack is similar in its philosophy to the second preimage attack discussed in 6.2.1. It uses a precomputed $2^{n/2}$ rho structure where the fixed point is computed on f_2. The complete attack algorithm is shown as procedure WEAK-PREIMAGE in algorithm 6 and illustrated in figure 6.5. The attack can be summarised as follows:

1. In the precomputation phase, a $2^{n/2}$ rho structure is constructed where the tail is constructed using f_1 and the tip is computed using f_2. From the tip of rho, a $2^{n/2}$ Joux multicollision set J is constructed.

2. In the online phase, meet-in-the-middle is used to find a two block linking message between h_b and the target hash value h[3]. The two block string is ω (say) and $h_c = f_1^+(iv, \omega)$ (say). The last step is to use the messages from $\mathcal{J}$ in block-wise reverse fashion to get a matching with $\mathcal{T}_\rho$. The walk from iv to h_ρ^t gives the required preimage message.

[3]Note that, here we are assuming that f_2 produces random and distinct preimages at each invocations.

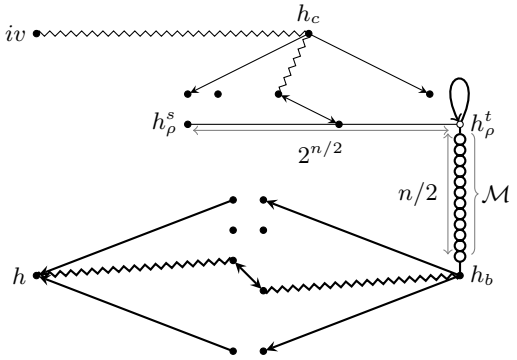

FIGURE 6.5: Preimage attack on Zipper hash with weakly invertible f_2.

Algorithm 6: Preimage attacks with weakly invertible f_2.

Algorithm $\texttt{WEAK-PREIMAGE}_{f_1,f_2}(h)$
Input: $h \in \{0,1\}^n$
Output: $\overline{m} \in (\{0,1\}^{n'})^+$
Data: 2^k RHO, ρ.
1 $(h_a, \mathcal{M}) \leftarrow \texttt{COLL}_{f_2}(h_\rho^t, \frac{n}{2})$
2 $(m_1, m_2) \leftarrow \texttt{MEET-IN-MID}_{f_2, f_2^{-1}}(h_a, h)$
3 $h_b \leftarrow f_1^+(iv, m_2||m_1)$
4 $\mathcal{D} \leftarrow \texttt{DIVDIAM}_{f_1}(h_b, \mathcal{M}^{rev})$
5 $(h_j, M_j) \leftarrow \texttt{LISTCOLL}(\mathcal{L}_\mathcal{D}, \mathcal{T}_\rho)$
6 $\overline{m} \leftarrow m_2||m_1||M_j||m_\rho^{2^k-j}$
7 **return** $\overline{m}$

Algorithm $\texttt{RELAXED-WEAK-PREIMAGE}_{f_1,f_2}(h)$
Input: $h \in \{0,1\}^n$
Output: $\overline{m} \in (\{0,1\}^{n'})^+$
8 $h \xleftarrow{\$} \{0,1\}^n$
9 $m \xleftarrow{\$} \{0,1\}^{n'}$
10 $\mathcal{C} = \texttt{VARIANT-CHAIN}_{f_2}(h, k, m)$
11 $(h_a, \mathcal{M}) \leftarrow \texttt{COLL}_{f_2}(h', \frac{n}{2})$
12 $(m_1, m_2) \leftarrow \texttt{MEET-IN-MID}_{f_2, f_2^{-1}}(h_a, h)$
13 $h_b \leftarrow f_1^+(iv, m_2||m_1)$
14 $\mathcal{D} \leftarrow \texttt{DIVDIAM}_{f_1}(h_b, \mathcal{M}^{rev})$
15 $(h_j, M_j) \leftarrow \texttt{LISTCOLL}(\mathcal{L}_\mathcal{D}, \mathcal{C})$
16 $\overline{m} \leftarrow m_2||m_1||M_j||m_\rho^{\frac{2^{k+1}-j-2}{2}}$
17 **return** $\overline{m}$

Complexity Analysis. All the online computations cost $O(2^{n/2})$ and hence the overall online complexity is $O(2^{n/2})$. The offline phase requires $O(2^n)$. The memory and message complexity are also $O(2^{n/2})$. Here, the offline complexity becomes $O(2^{k+n/2})$ for multiple fixed points assumption, in which case, the offline and online computational complexity becomes $O(2^{3n/4})$ and memory complexity reduces to $O(2^{n/4})$ for $k = \frac{n}{4}$.

6.3.1.2 Preimage Attack on Relaxed Zipper Hash

As in 6.2.2, the preimage attack on the relaxed zipper hash uses a $2^{\frac{n}{2}+1}-1$ chain instead of the rho structure and hence does not require any precomputation. The rest of the attack is similar to the previous attack on zipper hash. The complete algorithm is shown as procedure RELAXED-WEAK-PREIMAGE in algorithm 6 and illustrated in figure 6.6. The complexity analysis is also similar to the previous attack.

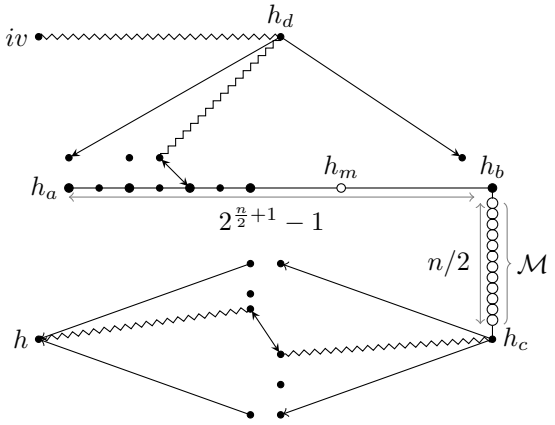

FIGURE 6.6: Preimage attack on Relaxed Zipper Hash with weakly invertible f_2.

6.3.2 Using Strong Invertibility of f_2.

If the underlying f_2 function is strongly invertible then we can easily construct a preimage attack using Joux multicollision structure and meet in the middle technique. The complete algorithm is shown as procedure STRONG-PREIMAGE in algorithm 7 and illustrated in figure 6.7. The attack can be summarised as follows:

1. In the first phase, starting at iv a 2^n Joux multicollision $\mathcal{J}$ is constructed using f_1. $\mathcal{J}$ is then divided into two sets $\mathcal{M}_1$ and $\mathcal{M}_2$ of $\frac{n}{2}$ cycles each.

2. In the second phase, messages from $\mathcal{M}_1$ and $\mathcal{M}_2$ are used in block-wise reverse to construct a meet-in-the-middle attack between h_a and the target hash value h. The block-wise reverse of the linking message between h_a and h gives the required preimage.

Complexity Analysis. The computational, memory and message complexity are $O(2^{n/2})$ each.

6.3.2.1 Application to Herding Attack on Zipper Hash

If the underlying f_2 function of the zipper hash is strongly invertible then, the adversary can use the preimage attack procedure, to herd any prefix or suffix to a target hash value. This is shown as the interactive procedure STRONG-HERDING in algorithm 7.

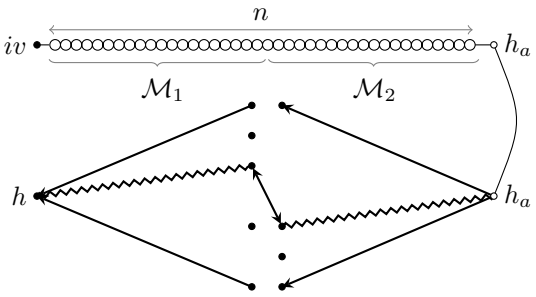

FIGURE 6.7: Preimage attack on Zipper hash with strongly invertible f_2.

Algorithm 7: Preimage and Herding attack on Zipper hash with strongly invertible f_2.

Algorithm $\texttt{STRONG-PREIMAGE}_{f_1,f_2}(h)$
Input: $h \in \{0,1\}^n$
Output: $\overline{m} \in (\{0,1\}^{n'})^+$
1 $(h_a, \mathcal{M}_1, \mathcal{M}_2) \leftarrow \texttt{COLL}_{f_1}(iv, n)$
2 $\mathcal{D}_1 \leftarrow \texttt{DIVDIAM}_{f_2}(h_a, \mathcal{M}_2^{rev})$
3 $\mathcal{D}_2 \leftarrow \texttt{CONVDIAM}_{f_2^{-1}}(h, \mathcal{M}_1^{rev})$
4 $(m_{\mathcal{D}_1}, m_{\mathcal{D}_2}) \leftarrow \texttt{LISTCOLL}(\mathcal{L}_{\mathcal{D}_1}, \mathcal{L}_{\mathcal{D}_2})$
5 $\overline{m} \leftarrow m_{\mathcal{D}_2}^{rev} || m_{\mathcal{D}_1}^{rev}$
6 **return** $\overline{m}$

Interactive $\texttt{STRONG-HERDING}_{f_1,f_2}()$
Chosen Target: $h \xleftarrow{\$} \{0,1\}^n$
Forced Prefix: $m_p \in (\{0,1\}^{n'})^+$
7 $h_a \leftarrow f_1^+(iv, m_p)$
8 $h_b \leftarrow (f_2^{-1})^+(h, m_p^{rev})$
9 **reset** $iv := h_a$
10 $m_s \leftarrow \texttt{STRONG-PREIMAGE}_{f_1,f_2}(h_b)$
11 **return** m_s

Chapter 7

Conclusion and Future Work

In this thesis, we have mainly focussed on two verticals. First, to summarise the state-of-the-art in cryptanalysis of iterated hash function and its variants. Second, we have proposed some new structures and their application in constructing attacks on iterated hash, concatenated hash, and zipper hash. In section 7.1, we will summarise all the existing results, and section 7.2 summarises the results proposed in this thesis. Section 7.3 enumerates some possible applications and future work based on our results.

7.1 Summary of Existing Structures and Attacks

We have reviewed (see chapter 4) most of the popular attack structures such as Joux multicollision, Kelsey-Schneier expandable message set, Diamond structure and others. We have also reviewed the applications of these structures in constructing attacks on the iterated hash design and its variants such as concatenated hash, hash twice, XOR combiner and others. We summarise the complexity results for these structures and attacks in tables 7.1, 7.2, 7.3, 7.4, and 7.5.

Structures	Computational Complexity	Memory Complexity	Message Complexity	Remarks
Joux multicollision [37]	$O(k \cdot 2^{n/2})$	$O(kn)$	$O(2^{n/2})$	For a 2^k Joux multicollision set.
Expandable Message [1]	$O(2^k + k \cdot 2^{n/2})$	$O(kn)$	$O(2^{n/2})$	For a $[k, 2^k+k-1]$ expandable message set.
Diverging Diamond	$O(2^k)$	$O(2^k)$	$O(1)$	For a 2^k diverging diamond.
Converging Diamond [2]	$O(2^{(n+k)/2})$	$O(2^k)$	$O(2^{(n+k)/2})$	For a 2^k converging diamond using forward queries.
Reverse Diamond [5]	$O(2^k)$	$O(2^k)$	$O(1)$	For a 2^k reverse diamond.
Switch and Interchange Structure [3]	$O(2^{2k+n/2})$	$O(2^k)$	$O(2^{2k+n/2})$	For a 2^k switch and interchange structure.

TABLE 7.1: Complexity results for existing attack structures.

Attacks	Computational Complexity	Memory Complexity	Message Complexity	Remarks
Multicollision Attack [37]	$O(2^{n/2})$	$O(kn)$	$O(2^{n/2})$	Ignoring the factor k.
Second Preimage Attack [1]	$O(2^{n/2} + 2^{n-k} + 2^k)$	$O(2^k)$	$O(2^{n/2}+2^{n-k})$	For $k = \frac{n}{2}$ this optimises to $O(2^{n/2})$.
Herding Attack [2]	$O(2^{n/2} + 2^{(n+k)/2} + 2^{n-k})$	$O(2^k)$	$O(2^{(n+k)/2})$	The offline and online complexity is $O(2^{2n/3})$ for $k = \frac{n}{3}$.

TABLE 7.2: Complexity results for existing attacks on the iterated hash.

Attacks	Computational Complexity	Memory Complexity	Message Complexity	Remarks
Collision Attack [37]	$O(2^{n/2})$	$O(n^2)$	$O(2^{n/2})$	Ignoring the factor $\frac{n}{2}$.
Preimage Attack [37]	$O(2^n)$	$O(n^2)$	$O(2^{n/2})$	Ignoring the factor n.
Herding Attack [4]	$O(2^{n/2} + 2^{(n+k)/2} + 2^{n-k})$	$O(2^k)$	$O(2^{(n+k)/2})$	The offline and online complexity is $O(2^{2n/3})$ for $k = \frac{n}{3}$.

TABLE 7.3: Complexity results for existing attacks on the concatenated hash.

Attacks	Computational Complexity	Memory Complexity	Message Complexity	Remarks
Herding Attack on Hash Twice [4]	$O(2^{n/2} + 2^{(n+k)/2} + 2^{n-k})$	$O(2^k)$	$O(2^{(n+k)/2})$	The offline and online complexity is $O(2^{2n/3})$ for $k = \frac{n}{3}$.
Second Preimage Attack on Hash Twice [4]	$O(2^k + 2^{n/2} + 2^{(n+\ell)/2} + 2^{n-k} + 2^{n-\ell})$	$O(2^\ell)$	$O(2^{(n+\ell)/2})$	The offline and online complexity is $O(2^{2n/3})$ for $k = O(\ell)$, and $\ell = \frac{n}{3}$.
Preimage Attack on XOR combiner [3]	$O(2^{2k+n/2} + 2^{n-k})$	$O(2^k)$	$O(2^{2k+n/2})$	The offline and online complexity is $O(2^{5n/6})$ for $k = \frac{n}{6}$.

TABLE 7.4: Complexity results for existing attacks on Hash twice and XOR hash combiner.

Attacks	Computational Complexity	Memory Complexity	Message Complexity	Remarks
Herding Attack [4]	$O(2^{n/2} + 2^{(n+k)/2} + 2^{n-k})$	$O(2^k)$	$O(2^{(n+k)/2})$	This is a forced suffix attack, where offline and online complexity is $O(2^{2n/3})$ for $k = \frac{n}{3}$.
Second Preimage Attack [5]	$O(2^{n/2} + 2^{n-k} + 2^k)$	$O(2^k)$	$O(2^k + 2^{n/2})$	This attack works for strongly invertible f_1, where offline and online complexity is $O(2^{n/2})$ for $k = \frac{n}{2}$.

TABLE 7.5: Complexity results for existing attacks on the zipper hash.

7.2 Summary of Our Structures and Attacks

We have proposed three new structures, viz., the chain structure, the multi-pipe expandable message set and the rho structure (see chapter 5 and chapter 6). We have also showed significant applications of these structures in constructing attacks. Though the chain structure seems an ordinary structure, but it has some important applications as demonstrated in our herding attack on the iterated hash, (second) preimage attack on the zipper hash and preimage attack on the concatenated hash. In particular, the herding attack is a surprising result where the complexity is reduced from $O(2^{2n/3})$ to $O(2^{n/2})$. The multi-pipe expandable message set is used in combination with a variant of the chain structure in constructing a second preimage attack on the relaxed zipper hash. This is again an important result, as it shows that if the underlying functions are identical, then the extra pass of zipper hash doesn't offer any amplification in the second preimage security. The rho structure is the core of our (second) preimage attack on the zipper hash. Though the structure is inefficient to compute in online phase for random oracle model, but if the compression function has multiple fixed points, the rho construction can be done in less than 2^n computations. We compile the complexity results for the proposed structures in table 7.6.

Structures	Computational Complexity	Memory Complexity	Message Complexity	Remarks
Chain Structure (see chapter 5)	$O(2^k)$	$O(n)$	$O(1)$	For a 2^k chain structure.
Multi-pipe Expandable message (see chapter 5)	$O(2^k + ((\frac{n}{2})^{\ell-1} + 1) \cdot k \cdot 2^{n/2})$	$O(kn^{\ell-1})$	$O(2^{n/2})$	For a $[(\frac{n}{2})^{\ell-1} \cdot k + k, 2^k + (\frac{n}{2})^{\ell-1} \cdot k + k - 1]$ expandable message set.
Rho structure (see chapter 6)	$O(2^n)$	$O(n)$	$O(2^n)$	For a 2^k rho structure, under random oracle assumption.
Rho structure (see chapter 6)	$O(2^{k+n/2})$	$O(2^{n/2})$	$O(1)$	For a 2^k rho structure, under multiple fixed point assumptions.

TABLE 7.6: Complexity results for attack structures proposed in this thesis.

With regard to attacks, we have presented a complete analysis of the zipper hash and its relaxed variant (see chapter 6), with regard to three security properties, namely, preimage, second preimage and herding attack resistance. Our results are surprising, as the zipper hash has resisted most of the known attacks [1, 2, 4]. We have also proposed (second) preimage attacks on concatenated hash with weak compression functions. Our preimage attack on the concatenated hash is a minor improvement over the attack given by Leurent *et al.* [3]. It also shows that the bound given by Hoch *et al.* in [41] is also applicable when one of the components is weakly invertible. We assemble the complexity results for all these attacks in tables 7.7, 7.8, and 7.9.

Attacks	Computational Complexity	Memory Complexity	Message Complexity	Remarks
Herding Attack on Itertaed Hash (see chapter 5)	$O(2^k + 2^{n/2} + 2^{n-k})$	$O(2^k)$	$O(2^{n/2})$	The offline and online complexity is $O(2^{n/2})$ for $k = \frac{n}{2}$.
Second Preimage Attack on Concatenated Hash (see chapter 5)	$O(2^{n/2} + 2^{n-k} + 2^k)$	$O(2^k)$	$O(2^{n/2})$	This attack works for one weakly invertible and one MD hash component. The offline and online complexity optimises to $O(2^{n/2})$ for $k = \frac{n}{2}$.
Preimage Attack on Concatenated Hash (see chapter 5)	$O(2^{n/2})$	$O(2^{n/2})$	$O(2^{n/2})$	This attack works for one weakly invertible and one strongly invertible component.

TABLE 7.7: Complexity results for attacks on iterated hash and concatenated hash proposed in this thesis.

Attacks	Computational Complexity	Memory Complexity	Message Complexity	Remarks
Second Preimage Attack (see chapter 6)	$O(2^n + 2^{n/2} + 2^{n-k} + 2^k)$	$O(2^k)$	$O(2^{n/2})$	The online complexity optimises to $O(2^{n/2})$ for $k = \frac{n}{2}$. Note that the offline cost becomes $O(2^{k+n/2})$ for multiple fixed points oracle, i.e., the offline and online complexity become $2^{3n/4}$ for $k = \frac{n}{4}$.
Preimage Attack (see chapter 6)	$O(2^n + 2^{n/2})$	$O(2^{n/2})$	$O(2^{n/2})$	This attack works for weakly invertible f_2. The offline cost becomes $O(2^{k+n/2})$ for multiple fixed points oracle, i.e., the offline and online complexity become $2^{3n/4}$ and memory complexity reduces to $O(2^{n/4})$ for $k = \frac{n}{4}$.
Preimage Attack (see chapter 6)	$O(2^{n/2})$	$O(2^{n/2})$	$O(2^{n/2})$	This attack works for strongly invertible f_2.
Herding Attack (see chapter 6)	$O(2^{n/2})$	$O(2^{n/2})$	$O(2^{n/2})$	This attack works for strongly invertible f_2, and utilises the preimage attack technique.

TABLE 7.8: Complexity results for attacks on zipper hash proposed in this thesis.

Attacks	Computational Complexity	Memory Complexity	Message Complexity	Remarks
Second Preimage Attack (see chapter 6)	$O(2^{n/2} + 2^k + 2^{n-k})$	$O(2^k)$	$O(2^{n/2})$	The offline and online complexity optimises to $O(2^{n/2})$ for $k = \frac{n}{2}$.
Preimage Attack (see chapter 6)	$O(2^{n/2})$	$O(2^{n/2})$	$O(2^{n/2})$	This attack works for weakly invertible f_2.

TABLE 7.9: Complexity results for attacks on relaxed zipper hash proposed in this thesis.

We can summarise our main contributions and results as follows:

1. The chain structure and its applications, particularly, the improved herding attack on iterated hash (see chapter 5).
2. The multi-pipe expandable message set, which is used along with the chain structure in attacks on relaxed zipper hash (see chapter 5 and 6).
3. The rho structure and (second) preimage attacks on the zipper hash (see chapter 6).
4. (Second) preimage attacks on the concatenated hash, under certain weakness assumptions on the underlying compression functions (see chapter 5).

7.3 Possible Applications and Future Work

The structures and attacks presented in this thesis seem to have some nice applications. These results have also raised some hope for solution to some existing problems. Here we are enumerating few of them:

1. **Applications of chain structure.** The power of chain structure is evident from the improved herding attack (refer chapter 5). We believe that application of this structure to other attacks, especially, to the attacks based on diamond or elongated diamond structure, can help in reducing their complexity.

2. **Low memory attacks using chain structure.** Another line of research can be the application of chain structure in constructing attacks with memory constraints. Note that, a chain structure can simply be represented by its start and end points, and a single message block. The intermediate chaining values can be computed on demand. It is our belief that this structure can be used in combination with lambda attack to reduce memory usage.

3. **Applications of multi-pipe expandable message set.** The multi-pipe expandable message set seems to give a lot of power to the adversary in constructing attacks on multi-pass schemes. We have presented one application, second preimage on relaxed zipper hash. It will be interesting to apply this on other constructions.

4. **Second preimage attacks and their complexity.** One problem which remains open is the problem of long-message second preimage attack on concatenated hash. We have solved this problem to some extent for the zipper hash scheme. Future works can be carried out in reducing the complexity of precomputation phase of this attack. Another approach to this problem, requires a diamond structure with same labels along all the paths from the leaf nodes to the root node. Investigating the possibility of such a structure will be interesting.

Bibliography

[1] John Kelsey and Bruce Schneier. Second Preimages on n-bit Hash Functions for Much Less than 2^n Work. *IACR Cryptology ePrint Archive*, 2004. URL `http://eprint.iacr.org/2004/304`.

[2] John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In *Advances in Cryptology - EUROCRYPT '06 Proceedings*, 2006. URL `http://dx.doi.org/10.1007/11761679_12`.

[3] Gaëtan Leurent and Lei Wang. The Sum Can Be Weaker Than Each Part. *IACR Cryptology ePrint Archive*, 2015. URL `http://eprint.iacr.org/2015/070`.

[4] Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, and John Kelsey. Herding, Second Preimage and Trojan Message Attacks beyond Merkle-Damgård. In *Selected Areas in Cryptography, 16th Annual International Workshop, SAC '09 Revised Selected Papers*, 2009. URL `http://dx.doi.org/10.1007/978-3-642-05445-7_25`.

[5] Shiwei Chen and Chenhui Jin. A Second Preimage Attack on Zipper Hash. *Security and Communication Networks*, 2015. URL `http://dx.doi.org/10.1002/sec.1210`.

[6] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 1976. URL `http://dx.doi.org/10.1109/TIT.1976.1055638`.

[7] R. Rivest. The MD5 Message-Digest Algorithm, 1992.

[8] *Secure Hash Standard*. NIST, fips pub 180-1 edition, 1995.

[9] Mihir Bellare and Phillip Rogaway. The Exact Security of Digital Signatures - How to Sign with RSA. In *Advances in Cryptology - EUROCRYPT '96 Proceeding*, 1996. URL `http://dx.doi.org/10.1007/3-540-68339-9_34`.

[10] Mihir Bellare and Phillip Rogaway. Optimal Asymmetric Encryption. In *Advances in Cryptology - EUROCRYPT '94 Proceedings*, 1994. URL `http://dx.doi.org/10.1007/BFb0053428`.

[11] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Advances in Cryptology - CRYPTO '96 Proceedings*, 1996. URL http://dx.doi.org/10.1007/3-540-68697-5_1.

[12] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. Stronger Password Authentication Using Browser Extensions. In *Usenix Security*, 2005.

[13] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. URL http://bitcoin.org/bitcoin.pdf.

[14] Donald E. Knuth. *The Art of Computer Programming Volumes 1-3 Boxed Set*. Addison-Wesley Longman Publishing Co., 1998.

[15] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In *Advances in Cryptology - EUROCRYPT '05 Proceedings*, 2005. URL http://www.iacr.org/cryptodb/archive/2005/EUROCRYPT/2866/2866.pdf.

[16] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In *Advances in Cryptology - EUROCRYPT '05 Proceedings*, 2005. URL http://dx.doi.org/10.1007/11426639_2.

[17] Vlastimil Klima. Tunnels in Hash Functions: MD5 Collisions Within a Minute. *IACR Cryptology ePrint Archive*, 2006. URL https://eprint.iacr.org/2006/105.

[18] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In *Advances in Cryptology - CRYPTO '05 Proceedings*, 2005. URL http://dx.doi.org/10.1007/11535218_1.

[19] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In *Advances in Cryptology - EUROCRYPT '05 Proceedings*, 2005. URL http://dx.doi.org/10.1007/11426639_3.

[20] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In *Advances in Cryptology - CRYPTO '05 Proceedings*, 2005. URL http://dx.doi.org/10.1007/11535218_2.

[21] Jian Guo, Jérémy Jean, Gaëtan Leurent, Thomas Peyrin, and Lei Wang. The Usage of Counter Revisited: Second-Preimage Attack on New Russian Standardized Hash Function. In *Selected Areas in Cryptography, 21st International Workshop, SAC '14 Proceedings*, 2014. URL http://dx.doi.org/10.1007/978-3-319-13051-4_12.

[22] Riham AlTawy and Amr M. Youssef. Watch your Constants: Malicious Streebog. *IACR Cryptology ePrint Archive*, 2014. URL `http://eprint.iacr.org/2014/675`.

[23] K. Nishimura and M. Sibuya. Probability to Meet in the Middle. *Journal of Cryptology*, 1990.

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

[25] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *Advances in Cryptology - CRYPTO '90 Proceedings*, 1990.

[26] Bruce Schneier. Cryptanalysis of MD5 and SHA: Time for a New Standard, 2014. URL `https://www.schneier.com/essays/archives/2004/08/cryptanalysis_of_md5.html`.

[27] Chad Perrin. Use MD5 Hashes to Verify Software Downloads, 2007. URL `http://www.techrepublic.com/blog/it-security/use-md5-hashes-to-verify-software-downloads/`.

[28] Bart Preneel. Davies–meyer. In *Encyclopedia of Cryptography and Security*. 2011. URL `http://dx.doi.org/10.1007/978-1-4419-5906-5_569`.

[29] S. M. Matyas, C. H. Meyer, and J. Oseas. Generating Strong One-Way Functions with Cryptographic Algorithm. *IBM Technical Disclosure Bulletin*, 1985.

[30] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. ISBN 0849385237.

[31] Ralph C. Merkle. One way Hash Functions and DES. In *Advances in Cryptology - CRYPTO '89 Proceedings*. 1989. URL `http://dx.doi.org/10.1007/0-387-34805-0_40`.

[32] Ivan B. Damgård. A Design Principle for Hash Functions. In *Advances in Cryptology - CRYPTO '89 Proceedings*, 1989. URL `http://dl.acm.org/citation.cfm?id=118209.118248`.

[33] R. Canettia, R. L. Rivest, M. Sudan, L. Trevisan, S. P. Vadhan, and H. Wee. Amplifying Collision Resistance: A Complexity-Theoretic Treatment. In *Advances in Cryptology - CRYPTO '07 Proceedings*, 2007.

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

[35] Guido Bertoni, John Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic Sponge Functions, 2011.

[36] Moses Liskov. Constructing an Ideal Hash function from Weak Ideal Compression Functions. In *Selected Areas in Cryptography, 13th International Workshop, SAC '06 Proceedings*, 2006. URL http://dx.doi.org/10.1007/978-3-540-74462-7_25.

[37] Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In *Advances in Cryptology - CRYPTO '04 Proceedings*, 2004. URL http://dx.doi.org/10.1007/978-3-540-28628-8_19.

[38] Jalaj Upadhyay. Generic Attacks on Hash Functions. Master's thesis, University of Waterloo, 2003.

[39] Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, Pierre-Alain Fouque, Jonathan Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. New Second-Preimage Attacks on Hash Functions. *Journal of Cryptology*, 2010. URL http://www.lifl.fr/~bouillag/pub/JOC2010.pdf.

[40] Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second Preimage Attacks on Dithered Hash Functions. In *Advances in Cryptology - EUROCRYPT '08 Proceedings*, 2008. URL http://dx.doi.org/10.1007/978-3-540-78967-3_16.

[41] Jonathan J. Hoch and Adi Shamir. On the Strength of the Concatenated Hash Combiner When All the Hash Functions Are Weak. In *Automata, Languages and Programming, ICALP '08 Proceedings*, 2008. URL http://dx.doi.org/10.1007/978-3-540-70583-3_50.

[42] M. Fischlin and A. Lehmann. Security-Amplifying Combiners for Collision-Resistant Hash Functions. In *Advances in Cryptology - CRYPTO '07 Proceedings*, 2007.

[43] A. Numayama and K. Tanaka. On the Weak Ideal Compression Functions. In *Information Security and Privacy - ACISP '09 Proceedings*, 2009.

[44] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In *Fast Software Encryption, 11th International Workshop, FSE '04 Revised Papers*, 2004. URL http://dx.doi.org/10.1007/978-3-540-25937-4_24.

[45] Antoine Joux and Thomas Peyrin. Hash Functions and the (Amplified) Boomerang Attack. In *Advances in Cryptology - CRYPTO '07 Proceedings*, 2007. URL http://dx.doi.org/10.1007/978-3-540-74143-5_14.

[46] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. *IACR Cryptology ePrint Archive*, 2003. URL `http://eprint.iacr.org/2003/161`.

[47] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In *Advances in Cryptology - CRYPTO '05 Proceedings*, 2005. URL `http://dx.doi.org/10.1007/11535218_26`.

[48] Pin Lin, Wenling Wu, Chuankun Wu, and Tian Qiu. Analysis of Zipper as a Hash Function. In *Information Security Practice and Experience*. 2008. URL `http://dx.doi.org/10.1007/978-3-540-79104-1_28`.

[49] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In *Advances in Cryptology - CRYPTO '93 Proceedings*, 1993. URL `http://dx.doi.org/10.1007/3-540-48329-2_31`.

[50] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block Cipher Based Hash Function Constructions from PGV. In *Advances in Cryptology - CRYPTO '02 Proceedings*. 2002. URL `http://dx.doi.org/10.1007/3-540-45708-9_21`.

[51] Paulo S. L. M. Barreto and Vincent Rijmen. Whirlpool. In *Encyclopedia of Cryptography and Security*. 2011. URL `http://dx.doi.org/10.1007/978-1-4419-5906-5_626`.

[52] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2000. ISBN 0521791723.

[53] Martin E. Hellman. A Cryptanalytic Time-Memory Trade-Off. *IEEE Transactions on Information Theory*, 1980. URL `http://doi.ieeecomputersociety.org/10.1109/TIT.1980.1056220`.

[54] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In *Advances in Cryptology - ASIACRYPT '00 Proceedings*, 2000. URL `http://dx.doi.org/10.1007/3-540-44448-3_1`.

[55] Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In *Advances in Cryptology - CRYPTO '03 Proceedings*, 2003. URL `http://www.iacr.org/cryptodb/archive/2003/CRYPTO/1615/1615.pdf`.

[56] Jin Hong and Palash Sarkar. Rediscovery of Time Memory Tradeoffs. *IACR Cryptology ePrint Archive*, 2005. URL `http://eprint.iacr.org/2005/090`.

[57] Jovan Dj. Golic. Cryptanalysis of Alleged A5 Stream Cipher. In *Advances in Cryptology - EUROCRYPT '97 Proceeding*, 1997. URL `http://dx.doi.org/10.1007/3-540-69053-0_17`.

[58] Alex Biryukov, Sourav Mukhopadhyay, and Palash Sarkar. Improved Time-Memory Trade-Offs with Multiple Data. In *Selected Areas in Cryptography, 12th International Workshop, SAC '05 Proceedings*, 2005. URL `http://dx.doi.org/10.1007/11693383_8`.

[59] Mridul Nandi and Douglas R. Stinson. Multicollision Attacks on Some Generalized Sequential Hash Functions. *IEEE Transactions on Information Theory*, 2007. URL `http://dx.doi.org/10.1109/TIT.2006.889721`.

[60] Nasour Bagheri. Security Analysis of Zipper Hash Against Multicollisions Attacks. *Engineering, Technology and Applied Science Research*, 2012. URL `http://etasr.com/index.php/ETASR/article/view/17`.