Generic Attacks on Hash Functions

by

Jalaj Kumar Upadhyay

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2010

Abstract

The subject of this thesis is a security property of hash functions, called *chosen-target-forced-prefix preimage (*CTFP*) resistance* and the generic attack on this property, called the *herding attack*. The study of CTFP resistance started when Kelsey-Kohno introduced a new data structure, called a *diamond structure*, in order to show the strength of a CTFP resistance property of a hash function.

In this thesis, we concentrate on the complexity of the diamond structure and its application in the herding attack. We review the analysis done by Kelsey and Kohno and point out a subtle flaw in their analysis. We propose a correction of their analysis and based on our revised analysis, calculate the message complexity and the computational complexity of the generic attacks that are based on the diamond structure. As an application of the diamond structure on generic attacks, we propose a multiple herding attack on a special generalization of iterated hash functions, proposed by Nandi-Stinson.

Acknowledgments

First, I would like to thank my supervisor, Douglas Stinson, for being a constant source of encouragement, ideas, advice, and information. I am grateful that he worked with me closely when I needed his help, and yet let me work independently and choose my own problems when I wanted to. Most importantly, I would like to thank him for making me feel like a collaborator rather than a research assistant. I would also like to thank Douglas Stinson again for working with me on all the problems considered in this thesis. All the results presented in this thesis are the outcome of joint work with him. Needless to say, this thesis would not be possible without him.

Second, I would like to thank Ian Goldberg and Alfred Menezes, the other readers of my thesis for their careful review of my thesis, and for pointing out the corrections. Special thanks are due to my group members for providing such an excellent environment for research. As a member of the CrySP and CACR group, I have learned many things from them during their presenations. I thank them for their explicit explaination of the topics.

I would also like to thank my new friends in Waterloo for making my stay here enjoyable and my old friends for many years of friendship and support. In particular, I would like to thank Aashish and Akhilesh for supporting me during my early days in Waterloo. I would like to acknowledge beloved Darya for her love and constant motivation.

Finally, I would like to thank my family, particularly my brother Sarvagya, who always acted as a source of inspiration. I would like to acknowledge the motivation provided by my father Dr. D. K. Upadhyay and mother Mrs. Pushpa Upadhyay, who from such a long distance never let me feel that I am away from home.

To my family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Cryptographic hash functions and attacks

In their seminal paper that gave a new orientation to cryptography, Diffie and Hellman [19] 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 requirement of authentication in cryptographic protocols. At that time, cryptography concentrated mainly on the privacy of the data and authentication was considered as a subproblem, in the sense that protection of authenticity would follow automatically from privacy protection. The work by Diffie and Hellman showed a separation between the concept of privacy and authentication.

In this thesis, we study a special cryptographic primitive, known as a *hash function*, that has been long used in providing authentication of the data in the cryptographic protocols. A hash function is an easily computable function that maps an arbitrary-length input to a fixed-length output. A hash function with this property has application in wide areas, like in the construction of caches, managing databases (for example, in mining association rules [62]), construction of efficient data structures (for example, Bloom filters [48]) and efficient algorithms (for example, Rabin-Karp algorithm for string matching [41]).

In cryptography, we require a little more than just the property of mapping a long-length message to a shorter-length output. These additional properties are the so called security properties of the hash functions, and a hash function that has the desired security properties is termed a *cryptographic hash function*. In this thesis, whenever we use the term "hash function", we implicitly mean a cryptographic hash function.

Different protocols use different security properties of a hash function, depending upon the application of the hash function. One of the main reasons to use hash functions as a primitive in these protocols is that the output can be considered as compact and it is a one way transformation of the message. The term "compact" refers to the fact that for any finite length message, the output size of a hash function is fixed. So we do not need to care about variable-length messages when designing other primitives as we can use the hash function as a preprocessor. Another important aspect of a hash function is that it is not desirable that, from the output of a hash computation one can predict the input. This property is termed as "one-way".

In Section 1.2, we give an informal discussion of the security properties that are desirable for a hash function, the security properties that an adversary will try to attack, and some known methods of attacks. In Section 1.3, we give a brief exposition of the application of the hash functions at a very high level. In Section 1.4, we give a motivation for the problem that we study in this thesis and give the layout and the organization of the thesis.

1.2 Cryptographic hash functions and attacks

In this section, we will briefly discuss the security properties that are desired in a hash function if it is to have any cryptographic applications. We will also review some known methods of attacks, citing a few of the known attacks.

1.2.1 Security requirements

Security properties are motivated by the applications of hash functions, which we discuss later in this chapter. However, whenever one designs a hash function, say H that maps a set of all messages, $\mathcal{X}$ to a set $\mathcal{Y}$, it is often the case that the following three properties should hold:

1. Preimage resistance: Given an image $y \in \mathcal{Y}$, it should be difficult to find a preimage, $x \in \mathcal{X}$, such that $H(x) = y$.

2. Second preimage resistance: Given a message $x \in \mathcal{X}$, it should be difficult to find another message $x' \in \mathcal{X}$, such that $H(x') = H(x)$.

3. Collision resistance: It should be difficult to find two distinct messages, x and x' in the set $\mathcal{X}$, such that $H(x) = H(x')$.

In this thesis we study a variant of the preimage resistance property of a hash functions, termed as *chosen-target-forced-prefix (*CTFP*)* preimage resistance. Introduced in 2006 by Kelsey and Kohno [43], the CTFP resistance property has been discussed only informally in the literature. We do not aim to give a formal defintion. We work with the existing definition, which we include here, and use it in the same manner throughout the thesis.

CTFP resistance can be defined as a three-phase game between an adversary and a challenger.

1. In the first phase, the adversary performs some precomputation and commits to a hash output; which is her "chosen target".

2. In the second phase, the challenger selects a challenge and gives it to the adversary; the challenge is the "forced prefix".

3. In the third phase, in order to win the game, the adversary has to find a suffix, which when concatenated with forced prefix, yields the chosen target after hashing.

If the attacker is unable to find such a suffix, the hash function is CTFP *resistant.* In this thesis, we call the first phase the *offline* phase and the second phase and the third phase the *online* phase.

In this thesis, we also deal with the case of multiple second preimage resistance and multiple CTFP resistance properties of hash functions. We informally state them now and give a more formal version in Section 2.2.1. For any integer $r \geq 1$, r-way second *preimage resistance* requires that for any given message $x \in \mathcal{X}$, it should be difficult to find r distinct messages $x_1, \ldots, x_r \in \mathcal{X}$, different from x, such that $H(x_i) = H(x)$ for all $1 \leq i \leq r$. Similarly, *r-way collision resistance* requires that it should be difficult to find r distinct messages $x_1, \ldots, x_r \in \mathcal{X}$, such that $H(x_i) = H(x_j)$ for all $1 \leq i, j \leq r$. We alternatively term r-way collision resistance as *r-multicollision resistance.* For *r-way* CTFP *resistance*, we require that it should be difficult for the adversary to find r distinct suffixes, such that the hash computation of message formed by concatenation of any suffix with the forced prefix evaluates to the chosen target.

1.2.2 Known models of attack

In this section, we will briefly discuss some known model of attacks. We broadly divide them into two classes as follows:

(a) Attacks independent of the hash function construction.

(b) Attacks on a specific hash function construction.

The attacks that are independent of the hash function construction are more commonly known as *generic attacks,* while the attacks that are based on specific hash functions exploit the internal construction of the hash functions. We will discuss a few generic attacks that are relevant to the work of this thesis in more detail in Chapter 3.

Attacks independent of hash function construction

This class of attacks, which we from now on call generic attacks, are independent of the specific hash construction, and depend only on the size of the range of the hash function. The running time of such attacks is measured in terms of the number of calls made to the hash function; i.e., we treat the hash function as a black box or an oracle.

In the remainder of this thesis, we denote $H : \mathcal{X} \to \mathcal{Y}$ as the hash function and restrict our attention to the case when $\mathcal{X}$ and $\mathcal{Y}$ are finite sets. The basic assumption made in this type of attack is that the function is uniformly distributed, i.e., every element of the range has the same number of preimages. If this is not the case, then all the attack methods listed below will be more effective.

- Random attack: In a random attack against preimage resistance and second preimage resistance, the adversary randomly picks a message and hope that it maps to the given hash value. If the hash function in question exhibits the random behavior, the probability with which the attacker succeeds is $1/|\mathcal{Y}|$. This is the best that one can hope for a generic attack against preimage resistance and second preimage resistance.

- Birthday attack: The basic idea behind the birthday attack against collision resistance is that if we randomly sample 23 people, then the probability with which two of them share the same birthday is at least 1/2. This is much less than the number of days in a year, which is the total number of birthdays. For this reason, this attack is also termed as the "birthday paradox". This result assumes that birthdays are randomly distributed over the year; if this is not the case, the probability will be even higher. Translating the above in terms of a regular function, i.e., a function whose output is uniformly distributed, with the range size $|\mathcal{Y}|$, one needs to do about $\sqrt{|\mathcal{Y}|}$ computations of the function to obtain a collision with an appreciable probability. Therefore, it is common to call a hash function collision resistant if the minimum number of hash computations one needs to perform is of the order of square root of the size of the range of the hash function.

The above two attack methods form the basis of many generic attacks on hash functions [39, 44]. We will take up this matter in more detail in Chapter 3.

Attacks on specific hash function constructions

There are attacks on specific constructions of hash functions, like MD4 [76, 81], MD5 [78], SHA-0 [6, 79], and SHA-1 [77]. The attacks use some specific weakness in the internal construction of the hash functions, or some specific tools, which we now discuss briefly. In the following attack methods, a specific construction methodology of hash functions, known as iterated hash functions, is exploited. Readers may refer to Section 2.3 for the detailed description of the construction. We include a very informal description here for the sake of completeness.

In an iterated hash construction, we first define a compression function that acts on a small domain, and use this function iteratively to extend the domain size. The input message is first broken in to fixed-length blocks so that the blocks are of the size required by the compression function. The input to the compression function is a message block and the output from the previous iteration. The intermediate outputs are called *chaining values* and the final hash value is the output at the end of all the iterations.

In the informal discussion below, we omit the details of the attacks because it is not the main focus of this thesis. We refer the readers to the references cited for a more detailed exposition.

- Meet in the middle attack: This attack is a variation on the birthday attack, but instead of the hash value, intermediate chaining variables are compared. The attack enables an opponent to construct a message with a prespecified hash value, which is not possible in case of a simple birthday attack. For details and variations, we refer the reader to [60].
- Correcting block attack: This attack consists of substituting all blocks of the messages except for one block. This block is then calculated such that the hash function takes on a certain value. When one applies this method to the last block, then the attack is called a *correcting last block attack*. It should be noted that there are certain strengthening of hash constructions; such as Merkle-Damgård strengthening, which we will discuss in Chapter 2, that resist such an attack. However, they do not provide any security when this attack is applied to the first block or to some blocks in the middle. Hash functions based on modular arithmetic are especially sensitive to this attack.
- Fixed point attack: Dean used this attack in the case of the iterated hash function construction in his PhD thesis [17]. The idea of this attack is to find a chaining value, h, and a message, x, such that when the compression function applied to h and x gives h as an output.

- Differential attacks: Differential cryptanalysis [8] is based on the study of the relation between input and output differences. The attack is statistical as one searches for input differences that are likely to cause a certain output difference. If one is looking for collisions, we look for output differences that equal zero.
- Boomerang attacks: This is a refinement of differential cryptanalysis, introduced by Wagner [75] for the cryptanalysis of block ciphers. Later, it was also used in the cryptanalysis of hash functions.

There have been lot of attacks on specific hash construction using one or the other methods listed above, like the multi-block differential collision finding tool [13, 77, 79], which uses the differential attack considering multiple blocks at a time, and the Boomerang attack [40, 49]. In addition to the methods listed above, these attacks sometimes also use generic methods to perform certain other attacks. For example, an efficient generic multicollision attack by Joux [39] is used to generate second preimages for hash values in [47]. The same method has been used to find multicollisions for other specific hash functions like MD4, HAVAL, and Blender [47, 82]. Biham *et al.* [7] found collisions on SHA-0 and reduced SHA-1 by using near collisions in order to find collisions. Recently Guo *et al.* [31] constructed an attack on Tiger, MD4, and reduced SHA-2 using the meet-in-the-middle framework.

1.3 Motivation for security: applications

Hash functions are one of the most important primitives in the cryptography. They have a wide range of applications, mostly in digital signature [5, 15, 20, 27, 30, 33, 38], digital time-stamps [32], and message authentication codes [4, 63, 80]. It is not possible to include all the applications of hash functions. We just enumerate a few of them to give a general overview of how hash functions are applied. These applications are in no way complete, and we just aim to give an idea of the way in which hash functions are used as a primitives. We remark that there is additional sophistication involved in the actual construction of protocols that use hash functions.

1.3.1 Extending the domain

Hash functions are used as an underlying subroutine in extending the domain of a cryptographic protocol, mostly in the scenario when the protocol is an expensive process. The most important example is that of digital signature.

Digital Signatures

Historically, digital signatures were the first application of hash functions. The domain extension methodology for digital signatures is more commonly known as "hash then sign" paradigm. In such constructions, the user first hashes the message that he wishes to sign and then applies the standard digital signature algorithm on the hash output. This serves the purpose of extending the domain of messages that one can sign. However, this comes at the price that the scheme's security depends on the security of the underlying hash function.

The hash-then-sign paradigm can be also treated as an optimized digital signature, because signing a message is computationally more expensive process than hashing the same message. Also, it is easier to construct a digital signature scheme on a fixed-length input. Note that we require the hash function to be collision resistant in this type of construction. This is because signatures corresponding to colliding messages are the same, and hence a collision can be used to construct a forgery.

1.3.2 Authentication protocols

As mentioned earlier, hash functions are used for authentication in many cryptographic protocols. We just mention the most widely used application, which is in a message authentication code.

Message authentication codes

Consider the scenario where Alice and Bob share a secret key that identifies a hash function from a set of possible hash functions[1] and suppose that Alice wants to send a message, such that Bob can authenticate that the message was sent by Alice. Alice uses the hash function, indexed by the secret key, and computes the hash of the message using that hash function. She then sends the message and the hash value to Bob. Bob can authenticate the message by hashing the message with the hash function indexed by the secret key. This is called a *message authentication code* (MAC) and it is one of the most important applications of hash functions. Note that Alice and Bob can securely transmit information over an insecure channel if they are using cryptographic hash functions in this way.

[1]We are being very vague here in absence of proper definition of keyed hash family, which we define formally in Section 2.2. For the time being, we can consider a keyed hash family as a collection of hash functions, all of which map the same domain to the same range, and which are indexed in a certain way. A key can be that index or a way to find that index.

1.3.3 Applications

Identity based protocols

Identity based protocols were introduced by Shamir [69]. They have the property that a user's public key is an easily calculated function of his identity, while a user's private key can be calculated for him by a trusted authority, called the *private key generator*.

Hash functions are used in identity based schemes that use bilinear pairings. Let $e : G_1 \times G_2 \rightarrow \mathcal{G}_T$ be a bilinear pairing[2], where G_1, G_2, and G_T are properly chosen groups. The protocol uses a hash function whose range is the group, G_1, and uses this hash function to map the user's identity to an element of a group. The user can now use the normal group operations. It is easy to see that we require collision-resistant hash functions in identity based protocols.

Commitment schemes

A commitment scheme is a two-phase scheme: the commit phase and the reveal phase. For commitment schemes, the scheme allows the first party to commit to a value while keeping it hidden from the second party in the commit phase, with the ability to reveal the committed value later in the reveal phase. The basic commitment scheme has the following cryptographic requirements:

1. In the commit phase, the committing party should be unable to reveal any value other than the one committed to in the commit phase, and
2. The second party should be unable to gain any information about the committed value before the reveal phase.

Commitment schemes are very useful primitive in many cryptographic protocols. We give an informal description of a commitment scheme. For example, consider a simple electronic auction scheme in which the bidders commit to their bids and provide the commitment to the auctioneer. Once all the bids are collected, the auctioner asks for the commitments to be revealed which the bidders provide. The auctioneer then simply outputs the highest bid.

Hash functions are often suggested as a computationally efficient way of constructing commitment schemes: a commitment to x is $H(x)$, where H is a secure hash function. A security argument may go along the following lines: If H is one way, recovering x from $H(x)$ is infeasible, which implies the hiding property; if H is collision resistant, the commitment can only be opened to x (otherwise the cheating party is able to find a collision).

[2]A bilinear pairing is a non-degenerate bilinear mapping that is efficiently computable.

1.4 Related works and motivation for this work

In principle, a hash function takes an arbitrary-length input and produces a fixed-length output. In practice, this is done by first constructing a function that works on a fixed-size domain, such that the size of the domain is strictly greater than the size of the range. Such a function is termed a *compression function*. We then propose a rule to use the compression function so as to extend the domain to an arbitrary length. The rule that extends the domain is called a *combining rule*. The most common combining rule is iterative use of the underlying compression function.

In this thesis, we survey the weakness in this design principle and we present new ways to exploit the weakness, concentrating mainly on CTFP resistance.

The study of CTFP resistance started when Kelsey and Kohno introduced a new data structure, called the *diamond structure*, in order to show the strength of a CTFP resistance property of a hash function. They termed the attack against CTFP resistance as a *herding attack*. On a superficial level, CTFP seemed to be just another variant of the preimage resistance property of a hash function. Kelsey and Kohno [43] showed that at least in the case of iterated hash functions, the CTFP resistance property is ultimately limited by the collision resistance property of the compression function. We will make this point more clear in Chapter 4 and Chapter 5. Later it was shown that diamond structures have applications in other attacks where the known attack methods have failed [2, 3].

Prior to this, Joux [39] found a very efficient way of finding multicollisions on iterated hash functions. Kelsey and Scheneir [44] used Joux's idea to find second preimages on long messages. These works opened a new domain of research in the construction of hash functions. Since the use of simple iteration as the combining rule was shown susceptible to these attacks, attention was concentrated on the combining rule and using message blocks more than once. Liskov [50] used every block of the message twice, the second time in reverse order and proved the security in the indifferentiability model [51]. Hoch and Shamir [37] proposed an XOR-construction and showed that even in the powerful adversarial scenario in which collisions for the underlying compression functions can be found faster than the birthday attack, collisions in the concatenated hash cannot be created using less than $2^{n/2}$ work, where the n is the bit length of hash outputs. Lehmann and Fisclin [24] also proposed a new combining rule that amplifies the security of the underlying compression functions. Recently, Numayama and Tanaka [61] generalized the model of Hoch and Shamir to provide a new combining rule.

Apart from this idealized setting, there has been work in the complexity-theoretic treatment of the amplification of collision resistance. Canetti *et al.* [12] proposed two methods of amplification of collision resistance: by using concatenation and by using codes. It is important to note that among all the above constructions, only Liskov's construction, more commonly known as Zipper hash, is susceptible to the multicollision attack [36, 59]. Andreeva *et al.* [2] showed that this construction is susceptible to a variant of the herding attack, in which the challenger provides the suffix and the adversary has to find the prefix. They argued informally that in the case of Zipper hash, the herding attack as proposed by Kelsey and Kohno is as hard as finding a second preimage.

Under these recent developments of the desire for amplification of the collision-resistance property of the combining rule, and the herding attack of Andreeva *et al.* [2], it is desirable to analyze the CTFP resistance property of these constructions. Also, it is natural to ask whether the attack of Andreeva *et al.* translates to a more general class of hash functions. In this thesis, we will try to answer the second question. We will mainly deal with hash function constructions that use certain types of combining rules. We aim to analyze the security of a generalized family of hash functions against CTFP attack using recent data structures proposed for generic attacks. We perform the complexity analysis of our attack and prove bounds to determine when our attack is successful. For this, we need a more rigorous complexity analysis of the construction of the diamond structure, because it is the basic data structure used in our attacks.

1.5 Organization of thesis

This thesis consists of the following chapters.

1. Chapter 2 covers the notation that we use throughout the thesis, the basics of hash functions and their security definitions. We also give a brief discussion of the basics of sequences and partial orders to the level required to understand this thesis.

2. In Chapter 3, we review the previous generic attacks that have been proposed in the literature and which are used in our work.

3. In Chapter 4, we revisit the complexity analysis of the diamond structure. Using the concept of random graphs, we show that there is a gap in the analysis given by Kelsey and Kohno. We give a rigorous analysis to show that the message complexity of the construction is actually $\sqrt{k}$ times the estimate of Kelsey and Kohno, where k is the parameter that defines the size of the diamond structure. We also perform an analysis of computational complexity and substantiate the claim that it is also important parameter and should be taken into account when performing an analysis of any attack. To show the differences that arise from our new analysis, we compare the analysis of the attack done on other hash functions with an analysis done by us, taking into the account the above gap.

4. In Chapter 5, we propose a variant of the diamond structure, which we call a *kite* structure. We use this structure to herd multiple messages for a generalized family of hash functions, defined by Nandi and Stinson [59]. In our analysis, we use the bound of Hoch and Shamir [36] to find the minimum block length of the message for a successful attack. We define the adversary model in which the adversary has the freedom to permute the sequence on which the hash function is based.

Chapter 2

Preliminaries

2.1 Mathematical background

In this section, we give the mathematical notations that we use in this thesis. We also review some mathematical objects that are crucial to our study, like partial orders, sequences, and graph theory.

2.1.1 Notation

We let $\mathbb{N}$ denote the set of all natural numbers and $\mathbb{Z}$ denote the set of integers. For any integer $k \in \mathbb{N}$, we denote $\{1, 2, \ldots, k\}$ by $[k]$, and its binary representation by $\langle k \rangle$. Let $n \in \mathbb{N}$, then $\{0,1\}^n$ denotes all the n-bit strings. We denote the set of all finite strings by $\{0,1\}^*$. For any $x \in \{0,1\}^*$, we denote the length of x by $|x|$. We denote the concatenation of two strings x and y by $x\|y$ or simply as xy. We denote the message blocks of any message M by $M_1\|M_2\|\cdots\|M_\ell$, where ℓ denotes the number of message blocks.

If $\mathcal{X}$ is a finite set, then $x \overset{R}{\leftarrow} \mathcal{X}$ denotes that we pick x uniformly at random from the set $\mathcal{X}$. For any set $\mathcal{X}$ and $\mathcal{Y}$, $\mathcal{X} \cap \mathcal{Y}$ denotes the usual set intersection, $\mathcal{X} \cup \mathcal{Y}$ denotes the usual set union, $X \setminus Y$ denotes the set that contains all those elements of X that are not in Y, and $\mathcal{X} \times \mathcal{Y}$ denotes the cartesian product.

We use the following asymptotic notation. If $f : \mathbb{N} \to \mathbb{R}$ and $g : \mathbb{N} \to \mathbb{R}$ are two functions such that $g(n) > 0$ for n sufficiently large, then we write:

$$\begin{aligned} f \ll g \quad &\text{if} \quad f(n)/g(n) \to 0 \text{ as } n \to \infty \\ f \gg g \quad &\text{if} \quad f(n)/g(n) \to \infty \text{ as } n \to \infty. \end{aligned}$$

We use some complexity-theoretic notations, namely, $O(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$. Let $f, g : \mathbb{N} \to \mathbb{N}$ be two functions. We write:

$$\begin{aligned}
f(n) \in O(g(n)) &\quad \text{if} \quad \exists c, n_0 > 0 \text{ such that } f(n) \leq cg(n), \text{ for all } n \geq n_0 \\
f(n) \in \Omega(g(n)) &\quad \text{if} \quad \exists c, n_0 > 0 \text{ such that } f(n) \geq cg(n), \text{ for all } n \geq n_0 \\
f(n) \in \Theta(g(n)) &\quad \text{if} \quad f(n) \in O(g(n)) \text{ and } g(n) \in O(f(n)).
\end{aligned}$$

See [14] for more details.

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

We now give a brief overview of sequences, partial orders, and graph theory to the level of generality required to understand this thesis.

2.1.2 A note on sequences

We require a little understanding of sequences and terminology associated with it. For more details, we refer the reader to reference [1].

We start with some basic definitions.

Definition 2.1. *A* sequence *ψ is a finite or infinite list of elements from some finite set $\mathcal{S}$. If a sequence ψ can be written as xyz (where y is a non-empty sequence and x and/or z can be empty sequences), then y is called a* factor *of ψ. A sequence ψ is called* square-free *if no factor of ψ can be represented in the form yy for any non-empty sequence y. A sequence ψ is said to be periodic with period $p > 0$ if it satisfies $\psi_i = \psi_{i+np}$ for $n = 1, 2, \ldots$.*

For the rest of this thesis, we reserve the Greek letter ψ to denote a sequence, and denote the i^{th} element of the sequence ψ by $\psi_{(i)}$.

It is easy to see that all square-free sequences are non-periodic. Rivest [64] used a generalization of the square-free sequences called an *abelian square-free sequence*, which we define next.

Definition 2.2. *A sequence ψ is said to be* abelian square free *if it cannot be written in the form $\psi = xyy'z$, where y is a non-empty sequence and y' is a permutation of y (x and/or z can be an empty sequence).*

We illustrate the difference between the square-free sequences and abelian square-free sequences with the help of following example.

Example. abcacb is square-free, but not abelian square-free, as abc is followed by acb, which is a permutation of abc.

There are many known sequences that are abelian square-free or just simply square-free. A natural requirement is to get an intuitive understanding of such sequences, specially from an algorithmic point of view where we are required to generate such sequences online. The picture is intuitively more clear for finite sequence but not so clear for infinite sequences.

In fact, the two sequences used in Rivest's construction are infinite sequences. One requires some measure to understand such sequences and to answer questions like how easy it is to construct the sequence, or how high is the periodicity of the sequence, or do all subsequences occur infinitely many times in the sequence, etc. Such questions can be partially answered by studying the finite subsequences of such infinite sequences and answering questions like, how many distinct factors of length n are there? The measure for the growth of the number of distinct subsequences of a specified length n is called the *complexity* of the sequence. For a sequence ψ, we let $Fact_{\psi}(n)$ denote the complexity of the sequence ψ, where n is the parameter that defines the length of the factors we are interested in.

In this whole thesis, for a sequence ψ, we denote the set on which the sequence depends by $alph(\psi)$; this is the *alphabet* set of the sequence. For any set $\mathcal{Y} \subseteq alph(\psi)$, we denote by $\psi|_{\mathcal{Y}}$ the sequence obtained by restricting ψ to the set $\mathcal{Y}$ and then replacing all the consecutive occurrences of any symbol in the resulting sequence by a single occurrence of the symbol. In the context of a finite sequence and the alphabet set that defines the (finite or infinite) sequence, we let $|\cdot|$ denote the number of symbols in the sequence and the cardinality of the alphabet set, respectively.

We give an example to make the above-defined notations more clear.

Example. Let $\psi = (2, 3, 1, 3, 2, 2, 3, 4, 2, 3)$, then $\mathcal{X} = alph(\psi) = \{1, 2, 3, 4\}$. Let $\mathcal{Y} = \{2, 3\} \subseteq \mathcal{X}$, then we obtain $\psi' = \{2, 3, 3, 2, 2, 3, 2, 3\}$ by restricting the sequence to alphabet set $\mathcal{Y}$, and $\psi|_{\mathcal{Y}} = \{2, 3, 2, 3, 2, 3\}$ by removing the consecutive occurrences of all symbols in ψ'. Also, $|\psi| = 10$ and $|alph(\psi)| = 4$.

2.1.3 A note on partial orders

We will need some basic notions of partial orders to understand Chapter 5. We give some basic definitions. For more details, we refer the reader to [71].

Definition 2.3. *A binary relation $\prec$ on a set $\mathcal{X}$ is a* partial order *if it is anti-symmetric, reflexive, and transitive. Then we call $(\mathcal{X}, \prec)$ a* partially ordered set. *The elements $x, y \in \mathcal{X}$ are called* incomparable *elements if neither $x \prec y$ nor $y \prec x$ holds. A set $\{x_1, \ldots, x_n\} \subseteq \mathcal{X}$ is called an* anti-chain *if all the elements are mutually incomparable. A set $\{x_1, \ldots, x_n\} \subseteq \mathcal{X}$ is called a* chain *if $x_i \prec x_j$ for $1 \leq i < j \leq n$. We say that n is the* chain-length.

Some examples of partially ordered set are:

1. The real numbers ordered by the standard less-than-or-equal relation, i.e., $(\mathbb{R}, \leq)$.
2. The set of natural numbers equipped with the relation of divisibility.
3. The set of subsets of a given set (its power set) ordered by inclusion $(\subseteq)$.
4. The set of subspaces of a vector space ordered by inclusion $(\subseteq)$.

2.1.4 A note on graph theory

In this section, we give some basic definitions from graph theory (for more information, see a standard textbook such as Bondy and Murty [11] or Diestel [18]).

We start with the definition of a graph.

Definition 2.4. *A* graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *consists of a finite set of* vertices *(or* nodes*),* $\mathcal{V}$*, and a finite set of* edges*,* $\mathcal{E}$*, where an edge joins two distinct vertices. The edge joining two vertices* u *and* v *is denoted by the pair* (u, v) *or* (v, u)*, and the vertices* u *and* v *are called* adjacent *vertices.*

Associated with any two vertices in a graph, a natural question that arises is whether it is possible to reach one vertex from the other vertex by tracing any set of edges. This leads to the definition of a *path* and *connectivity* of the graph.

Definition 2.5. *A* path *is a finite sequence of vertices* $(v_1, \ldots, v_k)$*, such that* v_i *and* v_{i+1} *are adjacent for* $1 \leq i \leq k-1$*. If it is possible to reach any vertex from any other vertex in the graph via some finite path, then the graph is* connected*. A maximal connected subgraph of a graph is called a* connected component*.*

In this thesis, we will be interested in finding whether a graph has certain characteristics, like is there a set of edges such that every vertex of the graph is incident to at least one edge of the set, or is there a set of vertices such that each edge of the graph is incident to at least one vertex of the set? These properties are called an *edge cover* and a *vertex cover* of the graph. We follow up with a formal definition.

Definition 2.6. *An* edge cover *of a graph is a set of edges such that every vertex of the graph is incident with at least one edge of the set. If an edge cover has an additional property that no two edges share a vertex, then such edge cover is called a* perfect matching *or a* one-factor*. On the other hand, if there exists a set of vertices such that each edge of the graph is incident with at least one vertex of the set, then the set of such vertices is called a* vertex cover *of the graph.*

Remark 2.7. *An alternate definition of a perfect matching is from the concept of a matching. If there exists a set of edges, no two of which share a vertex, then the set of edges is called a* matching*. M is a* maximum matching *in the graph $\mathcal{G}$ if no matching in $\mathcal{G}$ contains more edges than M does. If a matching M contains every vertex of $\mathcal{G}$, then M is called a* perfect matching *or a* one-factor*. One can easily verify the equivalence of the two definitions.*

All the above-defined characteristics of graph are preserved under any relabelling of the vertices. We term such a characteristics of a graph as a *property*. More formally, we have the following definition.

Definition 2.8. *A* graph property *is defined to be a predicate that is preserved under all possible isomorphisms of a graph. A property of a graph is* monotone *if the property is preserved by addition of arbitrary new edges to the graph.*

Many natural properties of graphs are monotone properties, e.g., being connected, being two-connected, containing a Hamiltonian cycle, containing a perfect matching, containing a triangle, etc. However, not all natural properties are monotone properties. For eg, being a bipartite graph, being a perfect graph, etc.

2.2 Hash function basics

Hash functions have been defined in two settings: as an unkeyed hash function and as a keyed hash family. We start by giving a formal definition of a keyed hash family.

Definition 2.9. *For a finite* key space *$\mathcal{K}$, a space of* messages *$\mathcal{X}$, and a finite space of possible outputs called* message digests*, $\mathcal{Y}$, we define a* keyed hash family *as a set of functions,*

$$H_k : \mathcal{X} \to \mathcal{Y},$$

for every $k \in \mathcal{K}$. We denote the set as $\mathcal{H} := \{H_k : k \in \mathcal{K}\}$.

There is a notational point to be made clear for the above definition. A key $K \in \mathcal{K}$ acts as an index which defines which hash function is chosen from the family. We usually drop the subscript for the hash function, $H_k(\cdot)$ for $k \in \mathcal{K}$, when the function is clear from the context, or sometimes we use the key as a parameter to the hash function and write $H(k, \cdot)$ for $H_k(\cdot)$. Also, $\mathcal{X}$ can be a finite or infinite set. If $\mathcal{X}$ is finite, then a hash function is sometimes called a *compression function*. In this thesis, we restrict our attention to the case when $\mathcal{X}$ is finite.

An *unkeyed hash function* is a function $H : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ have the same meaning as in Definition 2.9. In a sense, we can treat an unkeyed hash function as a keyed hash family with $|\mathcal{K}| = 1$.

Equipped with the formal definition of a keyed hash family, we can formalize the security definition stated in Section 1.2.1. The security of a cryptographic hash function is usually defined in the so-called *random oracle* model. So, before giving a formal definition of the desirable security properties of a hash function, we give a brief introduction of the random oracle model.

Random oracle model

The random oracle model aims at capturing the idea of an "ideal" hash function. In this model, a hash function is randomly chosen from the keyed hash family, and we are given only oracle access to the hash function. This means that we do not have an access to the actual implementation of the hash function, but have an access to an interface through which we feed input and get the output. Therefore, the only way to compute the hash value is to query the oracle. Another important property of the random oracle is that if we query the same value multiple times, we get the same output each time.

In the random oracle model, we assume that each query takes constant time. Thus the security of the hash function is measured in relation with the number of queries made to the oracle, which we call the *message complexity*. From now on, we assume that hash computation of a messsage takes unit time, viewing the hash function in the random oracle model.

2.2.1 Security definitions

We gave an informal definition of the security properties of cryptographic hash functions in Section 1.2.1. Let $\mathcal{H} := \{H_k : k \in \mathcal{K}\}$ be a keyed hash family of functions from a set $\mathcal{X}$ to a set $\mathcal{Y}$. A keyed hash family is often required to have three basic security properties [42, 53, 70, 72]: collision resistance, preimage resistance, and second preimage resistance. We formally define them as follows:

1. For a keyed hash family, $\mathcal{H}$, the advantage of an adversary $\mathcal{A}$ in finding a *collision* is a real number that measures the probability with which $\mathcal{A}$ finds a collision in H_k if a random key k is provided to it. Mathematically,

$$Adv_{\mathcal{H}}^{Coll}(\mathcal{A}) := \mathbb{P}r[k \overset{R}{\leftarrow} \mathcal{K}; \{M_1, M_2\} \leftarrow \mathcal{A}(k) : H_k(M_1) = H_k(M_2) \wedge M_1 \neq M_2].$$

In the above expression, a random key k is chosen from the key space $\mathcal{K}$, and $\mathcal{A}$ is required to find two distinct messages, M_1 and M_2, such that $H_k(M_1) = H_k(M_2)$. The advantage of $\mathcal{A}$, denoted by $Adv_{\mathcal{H}}^{Coll}(\mathcal{A})$, is the probability with which $\mathcal{A}$ wins this game, where the probability is over the random choice of the key and the adversary's randomness.

2. For a keyed hash family, $\mathcal{H}$, the advantage of an adversary $\mathcal{A}$ in finding a *preimage* is a real number

$$Adv_{\mathcal{H}}^{preimage}(\mathcal{A}) := \mathbb{P}r[k \overset{R}{\leftarrow} \mathcal{K}; h \overset{R}{\leftarrow} \mathcal{Y}; M \leftarrow \mathcal{A}(k,h) : H_k(M) = h]$$

   that measures the probability with which $\mathcal{A}$ finds the preimage for a random hash value if the hash function is chosen randomly from the hash family.

3. For a keyed hash family, $\mathcal{H}$, the advantage of an adversary $\mathcal{A}$ in finding a *second preimage* is a real number

$$Adv_{\mathcal{H}}^{second}(\mathcal{A}) := \mathbb{P}r[k \overset{R}{\leftarrow} \mathcal{K}; M_1 \overset{R}{\leftarrow} \mathcal{X}; M_2 \leftarrow \mathcal{A}(k, M_1) : (M_1 \neq M_2) \wedge (H_k(M_1) = H_k(M_2))].$$

   In this experiment, a random key k is chosen and a message M_1 is chosen randomly from the domain of the hash family. $Adv_{\mathcal{H}}^{second}(\mathcal{A})$ measures the probability with which $\mathcal{A}$ finds a message M_2, different from M_1, such that $H_k(M_1) = H_k(M_2)$.

Informally, if a hash function with a hash result of bit-length n resists all such attacks that finds a second preimage (and a preimage respectively) with less than 2^n hash computations on average, then the hash function is called *second-preimage resistant* (and *preimage resistant* respectively). A hash function that is preimage resistant is called a *one-way hash function* (OWHF)[1]. Recall that by the birthday paradox, we can always find a collision for any hash function with about $2^{n/2}$ hash computations. If a hash function resists all attacks that finds a collision in less than $c \times 2^{n/2}$ hash computations for some small constant c, then the hash function is called *collision resistant.* Though it seems reasonable to say that collision resistance implies preimage and second-preimage resistance, this is not always true. A toy example is an identity function on a fixed length input. This is trivially collision resistant but not preimage resistant.

Remark 2.10. *It should be noted that in cryptographic practice, a collision resistant hash function maps arbitrary length strings to fixed length ones; it is a single function. However, in theory, a collision resistant hash function is always keyed; it is a hash family [66]. This discrepancy arises because any function, $H : \{0,1\}^* \to \{0,1\}^n$ always admits an efficient*

[1]Some authors as in [63] term a hash function OWHF if it is preimage resistant as well as second preimage resistant.

collision finding adversary. Now, as we have seen in Section 1.3, hash functions are usually used as a subroutine in cryptographic protocols. Conventionally, the security of a protocol Π *using a hash function* H *is proven by a reduction which captures the idea that existence of an effective adversary against* Π *implies existence of an effective adversary against* H*. For an unkeyed version of collision resistance, this won't work, as there is always an effective adversary against* H*.*

Apart from the above security properties, there are certain other security properties defined in the literature. Rogaway and Shrimpton [65] coined seven basic security properties of hash functions and showed the relationships between them. The properties are the collision resistance, three variants of the preimage resistance, and three variants of the second-preimage resistance. The three variants of the preimage resistance are traditional preimage resistance, where challenger can choose the key and the message randomly, *everywhere preimage resistance*, where the challenger can choose only the key randomly, and *always preimage resistance*, in which the challenger can only choose the message randomly. The adversary is required to find the preimage of the range point. Analagously, they defined the three variants of second-preimage resistance. For more detail, we refer the reader to [65].

Kelsey and Kohno [43] proposed a new security property, chosen-target-forced-prefix (CTFP) resistance, which is the primary interest of this thesis. CTFP resistance has not been studied in the keyed version in the literature. We therefore just present the informal definition, which is the same as defined in the Section 1.2.

Definition 2.11. (Chosen-Target-Forced-Prefix resistance)*. Let* H *be a hash function. For an adversary* $\mathcal{A}$*,* chosen-target-forced-prefix (CTFP) resistance *can be defined as a three-phase game between* $\mathcal{A}$ *and a challenger.*

1. *In the first phase,* $\mathcal{A}$ *commits to a hash output; which is her "chosen target".*
2. *In the second phase, the challenger selects a challenge and gives it to* $\mathcal{A}$*; the challenge is the "forced prefix".*
3. *In the third phase, in order to win the game,* $\mathcal{A}$ *has to find a suffix, which when concatenated with forced prefix, yields the chosen target after hashing.*

If the attacker is unable to find such a suffix, the hash function is CTFP resistant*. An attack on* CTFP *is more commonly called a* herding attack.

From the above definition of CTFP resistance, it is tempting to say that CTFP resistance is as hard as the second preimage resistance. However, Kelsey and Kohno [43] showed that at least in the case of iterated hash functions, the CTFP resistance property is ultimately limited by the collision resistance property of the compression function. They proposed an attack, which they called *herding attack*, to show this reduction. We give a brief overview of their method in Section 3.4.

In this thesis, we propose multiple second preimage attacks and multiple herding attacks on a certain class of hash functions. Therefore, we next generalize the notion of the second preimage resistance and CTFP resistance to the case of multiple second preimage resistance and multiple CTFP resistance, analogous to the notion of multicollision resistance.

Definition 2.12. (s-way second preimage resistance). *Extending the concept of the second preimage resistance, we define s-way second preimage as the following game:*

$$Adv_{\mathcal{H}}^{s-SP}(\mathcal{A}) := \mathbb{P}r\left[\begin{array}{ccc} k \xleftarrow{R} \mathcal{K}, M \xleftarrow{R} \mathcal{X}; & & M \neq M_i \ \forall i \in \mathbb{Z}_s \\ & : & M_i \neq M_j \forall 1 \leq i, j \leq s, i \neq j \\ \{M_1, M_2, \ldots, M_s\} \leftarrow \mathcal{A}(k, M) & & H_k(M) = H_k(M_i) \ \forall i \in \mathbb{Z}_s \end{array}\right]$$

In this experiment, a random key k is chosen and a message M is chosen randomly from the domain of the hash family. $Adv_{\mathcal{H}}^{s-SP}(\mathcal{A})$ measures the probability with which $\mathcal{A}$ finds s distinct messages, all different from M, such that every message has the same hash output as M.

Definition 2.13. (s-way Chosen-Target-Forced-Prefix resistance). *For an adversary $\mathcal{A}$,* s-way CTFP resistance *can be defined as a three-phase game between $\mathcal{A}$ and a challenger.*

1. *In the first phase, $\mathcal{A}$ commits to a hash output; which is her "chosen target".*
2. *In the second phase, the challenger selects a challenge and gives it to $\mathcal{A}$; the challenge is the "forced prefix".*
3. *In the third phase, in order to win the game, $\mathcal{A}$ has to find s-distinct suffixes, which when concatenated with forced prefix, yields the chosen target after hashing.*

If the attacker is unable to find such suffixes, the hash function is s-way CTFP *resistant.*

In the remainder of this thesis, we consider unkeyed hash functions only.

2.3 Design of iterated hash functions

From the mathematical definition of a hash function, a hash function maps an arbitrary-length input to a fixed-length output. However, in practice, it is not convenient to directly construct a function that takes an arbitrary length input. The hash function is therefore constructed in two steps. We first design a compression function, f on a small domain. Then we use the compression function to extend the domain to get the final hash function, H. The domain extension consists of the following two operations, which can be regarded as the preprocessing steps.

1. Padding Rule: We pad the message with a certain number of bits so that the bit length of the resulting message is a multiple of some fixed constant m. We then divide the message into blocks of bit length m.

2. Combining rule: We use the compression function to combine different blocks of the message.

We require that the padding rule and combining rule preserves the basic underlying properties of the compression function. One of the most widely used combining rules is the one in which the compression function is used iteratively. In this thesis, we deal with the hash constructions that use iteration as the combining rule.

It should be noted that without padding, iterated hash functions are susceptible to an extension attack[2]. This is especially a concern when hash functions are used in MACs. Also, improper padding can lead to an easy collision attack (see Nandi [58] for details). Merkle [54] and Damgård [16] proved the sufficient condition for the padding rule that preserves the collision resistance property of the compression function when we use iteration as the combining rule. Recently, Nandi [58] gave a characterization of collision-preserving padding rules. In this thesis, we deal with the padding rule, introduced independently by Merkle and Damgård, and define it next.

2.3.1 Merkle-Damgård construction

We define the simplified version of Merkle-Damgård domain extension algorithm for the compression function, $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. Let M be a message such that $|M| < 2^m$, and with a little abuse of notation, let $\langle M \rangle$ be the binary representation of $|M|$. Let $i > 0$ be the smallest integer such that $|M| + |\langle M \rangle| + i + 1$ is a multiple of m. We pad the message M with $10^i \| \langle M \rangle$. This padding method is known as MD-*strengthening*. We write $M \| 10^i \| \langle M \rangle = M_1 \| \ldots \| M_\ell$ for some positive integer ℓ and $|M_j| = m$ for all $j \in [\ell]$. We fix $h_0 = IV$, where IV is publicly known hash value, and define $H^f(IV, M)$ as follows:

- $h_j = f(h_{j-1}, M_j)$, for all $j \in [\ell]$,

[2]In an extension attack, given $H(M)$, and $|M|$, but not M, by choosing a suitable M', an attacker can calculate $H(M\|M')$.

- $H^f(IV, M) = h_\ell$.

The intermediate hash values $h_0, h_1, \ldots, h_{\ell-1}$ are called *chaining values*. Figure 2.1 shows the simple iterated hash construction, with the message padded in accordance with the MD-strengthening.

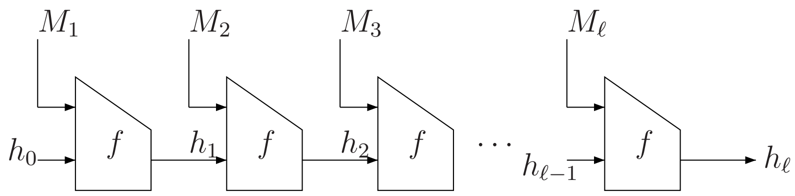


Figure 2.1: A simple iterated hash function

The superscript denotes the underlying compression function. We drop the superscript when it is clear from context. The input parameters are the initial hash value and the message. When viewing an iterated hash function as a function from some keyed hash family, IV can be seen as a publicly known key.

For the remainder of this thesis, we denote the message by M and the blocks of the padded message by $M_1, M_2, \ldots, M_\ell$.

Many other variants of iterated hash functions have been proposed in the literature. In this thesis, we will deal with three famous variants: the dithered hash construction, the hash twice construction, and the concatenated hash construction. We briefly define them next. In Chapter 5, we propose an attack on a generalized family of hash functions. We give the formal description of this generalized family of hash functions in Section 2.3.2.

Hash twice construction

In the *hash twice* construction, one hashes two consecutive copies of the message M. Formally, for a hash function H as defined above, it is defined as

$$\mathcal{HT} := H(H(IV, M), M),$$

where IV is the publicly known initial hash value. Figure 2.2 shows the construction of the hash twice hash function.

Concatenated hash function construction

In the *concatenated hash function construction*, one hashes two copies of the message M. Formally, for hash functions H and G, where H and G may be based on different

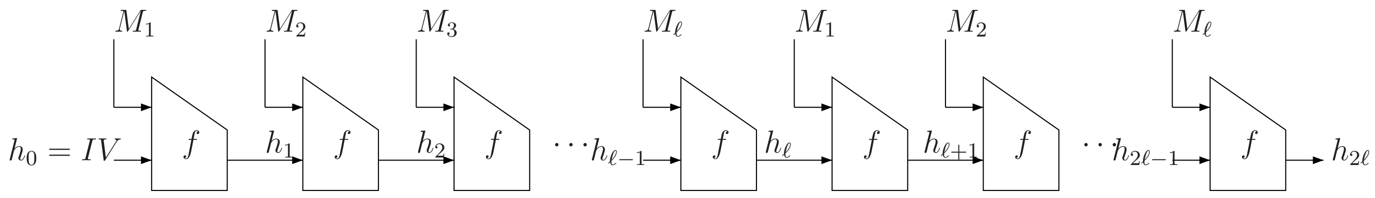


Figure 2.2: The hash twice construction

compression functions, it is defined as

$$\mathcal{CH} := H(IV, M) \| G(IV', M),$$

where IV and IV' are the publicly known initial hash values. Figure 2.3 shows the hash twice construction.

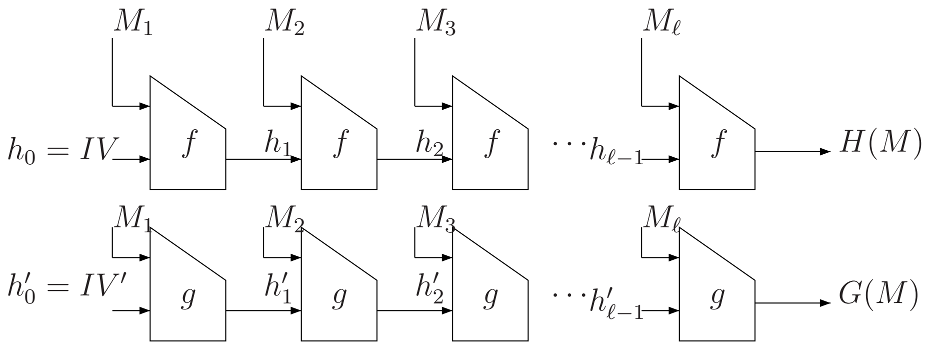


Figure 2.3: The concatenated hash construction

Dithered hash function construction

Let $\psi = \{\psi_{(i)}\}_{i=0}^{\infty}$ be any abelian square-free sequence. Then the construction of Rivest [64], also known as the *dithered hash* function construction[3], is

$$h_j = f(h_{j-1}, M_j \| \psi_{(j)}) \quad \text{for all} \quad j \in [\ell],$$

where $|M_j| + |\psi_{(j)}| = m$, $h_0 = IV$ is some publicly known value, and $\mathcal{DH}^{f,\psi}(IV, M) := h_\ell$.

Example. Let $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ be a compression function. Let us consider a sequence over the alphabet, $\{0,1,2\}$ as follow:

$$\psi = (2,1,0,2,0,1,2,1,0,1,2,0,2,1,0,2,0,1,2,0,2,1,0,1,2,1,0,2,0,1,2,1,0,1,2,0,\ldots).$$

This sequence is called the *Thue sequence*[4] and is square-free sequence (for details, refer to [1]). Let M' be the message after padding, such that the bitlength of M' is a multiple of $m-2$. We break the message as $M' = M'_1 \| M'_2 \| \cdots \| M'_\ell$. Further, writing the symbols of the sequence in the binary form $(2 \mapsto 10, 1 \mapsto 01, 0 \mapsto 00)$, we dither the message with the sequence and write $M = M'_1 \| 10 \| M'_2 \| 01 \| M'_3 \| 00 \| \cdots \| M'_\ell \| \langle \psi_{(\ell)} \rangle$. Let $M_i = M'_i \| \langle \psi_{(i)} \rangle$ for $1 \le i \le \ell$. Then the dithered hash function based on dithering sequence ψ is then defined as,

- $h_j = f(h_{j-1}, M_j)$, for all $j \in [\ell]$,
- $\mathcal{DH}^{f,\psi}(IV, M) = h_\ell$.

In the subsequent presentation, we often drop IV as one of the parameters, and denote by $H(M)$, any hash computation of the form $H(IV, M)$.

2.3.2 Generalized sequential hash function

Definition 2.14. *Let $\psi = (\psi_{(i)})_{i=1}^{\lambda}$ be a sequence that consist of elements from $[\ell]$, such that each element occurs at most twice and at least once*[5]*. Let $H^f(\cdot,\cdot)$ be an n-bit hash function based on the compression function f. For a message $M = M_1 \| M_2 \| \ldots \| M_\ell$, we define an iterated hash function, $\mathcal{H}^{f,\psi}$, based on ψ as follows:*

[3]Rivest adapted the word "dithering" from image processing to refer to the process of adding an additional *dithering* input to a sequence of processing steps, to prevent an adversary from causing and exploiting simple repetitive patterns in the input.

[4]We use the Thue sequence [74], which is not abelian square-free just for an example. In the actual construction, Rivest advocated to use the Kernänen sequence [45, 46], which is abelian square-free.

[5]Throughout this thesis, we will implicitly require that each message block is used at least once in the computation of the hash function.

- $h_i = f(h_{i-1}, M_{\psi_{(i)}})$, *for all* $1 \le i \le \lambda$,
- $\mathcal{H}^{f,\psi} = h_\lambda$.

Figure 2.4 shows the generalized sequential hash function defined on the sequence ψ.

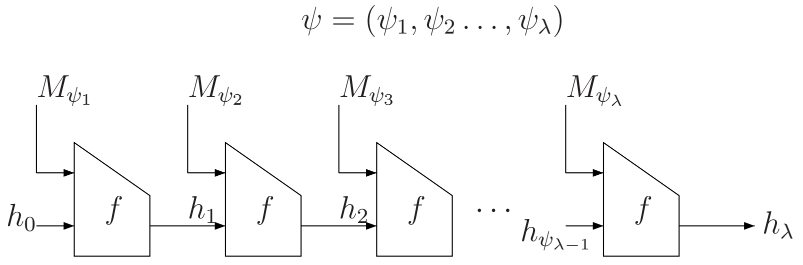


Figure 2.4: A generalized hash function construction

We denote $(\psi_{(i)}, \psi_{(i+1)}, \ldots, \psi_{(j)}) \subseteq \psi$ as $\psi[i,j]$ and any permutation π of the subsequence $\psi[i,j]$ by $\pi(\psi[i,j])$.

Example. The hash twice construction uses $\psi = (1, 2, \ldots, \ell, 1, 2, \ldots, \ell)$ and the Zipper hash construction uses $\psi = (1, 2, \ldots, \ell, \ell, \ell - 1, \ldots, 1)$.

Chapter 3

Related work

In this chapter, we perform a detailed literature review of some known generic attacks. We only cover those attacks that are used in this thesis.

3.1 Two important data structures

Before performing the literature survey, we give an overview of two important data structures that are used in generic attacks on the second preimage resistance and CTFP resistance of simple iterated hash functions and its variants.

3.1.1 Expandable message

Unlike what the term suggests, an expandable message is not a single message but a set of messages with different block lengths. This set is paramaterized by a variable ℓ, which defines the upper and lower bound on the block length of the messages in the set.

For functions $f_1, f_2 : \mathbb{N} \to \mathbb{N}$, an $(f_1(\ell), f_2(\ell))$-*expandable message* is a set that has messages of every block length ranging from $f_1(\ell)$-blocks to $f_2(\ell)$-blocks. In addition, an $(f_1(\ell), f_2(\ell))$-expandable message is a set of multicollisions for a publicly known fixed initial hash value and a fixed hash function. We give an explicit construction of an $(\ell, \ell + 2^\ell - 1)$-expandable message in Section 3.3.1.

3.1.2 Diamond structure

A diamond structure is parameterized by a variable k, which defines the size of the diamond structure and the complexity of the construction of such a diamond structure.

A 2^k-*diamond structure* is a complete binary tree structure, with the nodes labelled by a hash value. At any level ℓ for $0 \leq \ell \leq k$, there are $2^{k-\ell}$ nodes. The nodes at the level 0 of the diamond structure are called the *leaves*, and the node at the level k is called the *root* node. There is a single node which is connected to all the leaves, called the *source* node.

The edges of a diamond structure are labelled by a string. For an edge, e, let $\sigma(e)$ be its labelling, and let the label of the source node be IV. Then we label the nodes of the diamond structure in the following manner. Let $(e_0, e_1, \ldots e_\ell)$ be a path from the source node to a node ν at level ℓ. We label the node ν as,

$$h(\nu) = H(IV, \sigma(e_0)\|\sigma(e_1)\| \ldots \|\sigma(e_\ell))$$

The diamond structure is constructed in such a way that all paths leading to a given node N yield the same value $h(N)$, so $h(N)$ will be well-defined.

At any level ℓ of the structure there are $2^{k-\ell}$ hash values. These must be paired up in such a way that, when the next message blocks are appended, $2^{k-\ell-1}$ collisions occur. Thus there are $2^{k-\ell-1}$ hash values at the next level, which is level $\ell + 1$. The entire structure yields a set of 2^k-multicollisions, but it has other uses, as well.

Figure 3.1 shows a 2^3-diamond structure. In this figure, h_0 is the source node, h_3 is the root node, and there are eight leaf nodes.

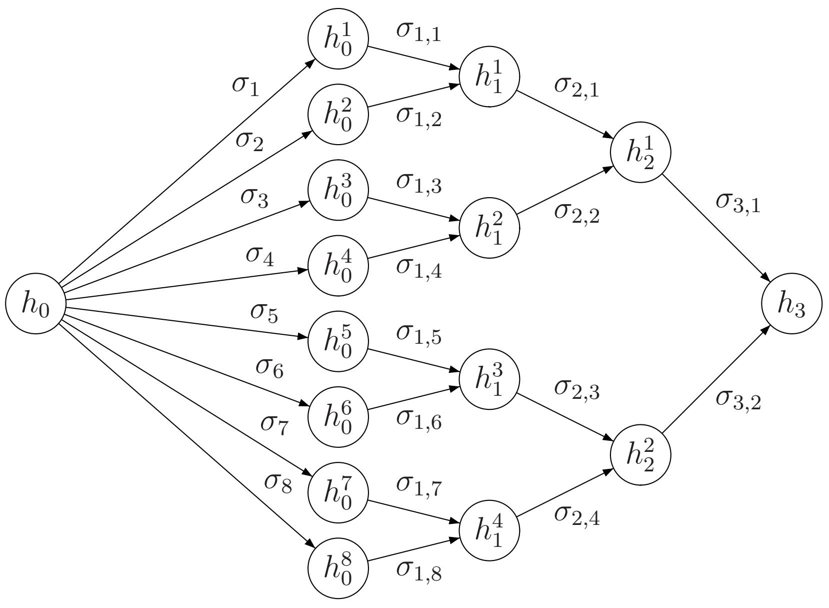


Figure 3.1: A 2^3 diamond structure

$$\begin{tikzcd}
h_0 \arrow[r, bend left, "m_0"] \arrow[r, bend right, "m_0^*"'] & h_1 \arrow[r, bend left, "m_1"] \arrow[r, bend right, "m_1^*"'] & h_2 \arrow[r, bend left, "m_2"] \arrow[r, bend right, "m_2^*"'] & h_3 \arrow[r, bend left, "m_3"] \arrow[r, bend right, "m_3^*"'] & h_4 & \bullet\ \bullet\ \bullet & h_{r-1} \arrow[r, bend left, "m_{r-1}"] \arrow[r, bend right, "m_{r-1}^*"'] & h_r
\end{tikzcd}$$

Figure 3.2: A 2^r-way collision

We next do a survey of recent generic attacks that have been proposed in the literature, and used in this thesis.

3.2 Multicollision attacks

Recall from Section 1.2.1 that a collision on any function $f : \mathcal{X} \to \mathcal{Y}$ is a pair of distinct elements $\{X_1, X_2\} \subseteq \mathcal{X}$, such that $f(X_1) = f(X_2)$. The generalization of the concept of collision is an s-way collision. An s-way collision (or s-multicollisions) is a set of s distinct elements, $\{X_1, \ldots, X_s\} \subseteq \mathcal{X}$ such that $f(X_1) = f(X_2) = \ldots = f(X_s)$.

Any function that maps a larger set to a smaller set is susceptible to the birthday attack, and hash functions are no exception. Assuming the computation of the underlying function takes constant time, the computational complexity of the basic birthday attack for finding an s-way collision on any random function is $\Theta(s \times |\mathcal{Y}|^{(s-1)/s})$ [73]. However, Joux [39] showed that it is very easy to construct a 2^r-way collision on an iterated hash function. We next describe Joux's multicollision attack.

3.2.1 Joux's multicollision

Joux's multicollision attack [39] is a 2^r-way collision attack on the Merkle-Damgård construction. Let f be the underlying compression function. Let $\mathcal{C}$ be a collision finding oracle, which when given the chaining value h as an input, outputs two messages m and m^*, such that $f(h, m) = f(h, m^*)$. The idea of Joux is that by r calls to $\mathcal{C}$, we get r pairs of messages, $\{(m_1, m_1^*), (m_2, m_2^*), \ldots, (m_r, m_r^*)\}$. Note that any message $M = M_1 \| M_2 \| \ldots \| M_r$, where each M_i is either of m_i or m_i^*, yields the same hash value. Hence, the adversary can construct a 2^r-way collision using r calls to $\mathcal{C}$.

Assuming that the hash function behaves as a random oracle, Joux stated the following theorem.

Theorem 3.1. [39] *For an iterated hash function, the complexity of finding a 2^r-way collision on an n-bit hash function is $O(r \times 2^{n/2})$.*

Using the above attack, Joux also found a collision on concatenated hash constructions with exponentially fewer queries to the compression function as compared to the basic birthday attack. Let $\mathcal{CH} = F(M)\|G(M)$ be a concatenated hash function and assume that F outputs an n_f-bit hash value and G outputs an n_g-bit hash value. Joux proposed the following collision attack. Without loss of generality, let $n_f \geq n_g$. First, construct a $2^{n_f/2}$-way collision on G. Since $n_f \geq n_g$, we can perform a direct application of the birthday attack on this set of $2^{n_f/2}$ messages, and expect that a collision occurs in the function F with reasonable probability. The messages that collide for the function G form the required collision. It is easy to verify that the complexity of the attack is of the order $n_f \times 2^{n_g/2} + 2^{n_f/2}$, which is exponentially less than $2^{(n_f+n_g)/2}$.

Using the same method, Joux's attack extends to finding collisions on various hash constructions, like Schneier's construction[1] [67] and the hash twice construction.

However, Joux's multicollision attack fails when we reuse any message block. This was the main idea behind Liskov's Zipper hash construction [50]. In the Zipper hash, each message block is used twice in the following way. Let the message be $M = M_1\| \ldots \|M_\ell$; then we first use the iterated hash function with IV as initial hash value, and M as message to get h. Once we have h, we use h as the initial hash value and use the iterated hash function with $M' = M_\ell\| \ldots \|M_1$ as the message. The output is the hash value obtained at the end of the second iteration.

Within a year of Liskov's construction, Nandi and Stinson [59] found a multicollision attack on a generalized family of iterated hash functions of which Liskov's construction is a special case. They generalized the construction of hash functions (Definition 2.14) so that the hash function uses any message block at most twice and in any order, and showed that even under this generalization, we can have a Joux-type multicollision attack. We next give a brief overview of their attack.

3.2.2 Nandi and Stinson's attack

For any sequence ψ, define a partial order $(\psi, \prec)$ in the following manner: $a \prec b$ if every occurence of a in the sequence precedes any occurrence of b in the sequence. Nandi and Stinson [59] observed that for $\psi = (\psi_{(i)})_{i=1}^{\lambda}$ in which every symbol is repeated at most twice and $|alph(\psi)| \geq uv$, one of the following holds:

1. The length of the maximal chain of $(\psi, \prec)$ is at least u, or

[1]Schneier's construction is a variant of the concatenated hash construction and is of the form $F(IV, G(IV', M)\|M)$, where F and G can be based on the same or different compression functions and IV and IV' are the same or different publicly known initial hash values.

2. There exists a $w < \lambda$, such that $\psi[1, w]$ has at least v elements that occur just once in $\psi[1, w]$.

Let $\mathcal{C}$ be a collision-finding oracle. They proved the following two theorems,

Theorem 3.2. *For any sequence ψ, if the maximal chain of $(\psi, \prec)$ has a length is greater than r, then it is possible to find a 2^r-way collision for $\mathcal{H}^{f,\psi}$ by using $|\psi|$ queries to $\mathcal{C}$.*

Theorem 3.3. *For any sequence ψ in which no element is repeated more than twice, if there exists an initial subsequence with at least tr elements that occur exactly once in the subsequence, then it is possible to find a 2^r-way collision by using $O(r(n + \ln r))$ queries to $\mathcal{C}$, where*

$$t = \frac{n+1}{2} + \frac{\ln \ln 2r}{2r},$$

and n is the length of hash output.

It is easy to see that by choosing $\ell = tr^2$, we can find a 2^r-way collision on a generalized class of hash function as defined in Definition 2.14. Since the Zipper hash and hash twice are specific cases of the generalized family of hash functions, they are also susceptible to this attack.

The attack of Nandi and Stinson is constructive in the sense that it is possible to construct a 2^r-way collision from the proof of Theorem 3.2 and Theorem 3.3. The only thing that they did not discuss is how to differentiate the two cases for any given sequence. This can be easily done as the two cases covers all the possibilities. The procedure followed is to find the maximal chain of the sequence. If it is at least u, then we use the steps stated in the proof of Theorem 3.2, otherwise we use the steps in the proof of Theorem 3.3. Finding a maximal chain of the subsequence can be done by computing the longest subsequence [68] which can be done by a standard dynamic programming technique.

Hoch and Shamir [36] extended the work of Nandi and Stinson for the case when any symbol is repeated at most q times. However, for large values of q, the attack is mainly of theoretical interest as the complexity of the attack is a super-exponential function of q [34].

3.2.3 Hoch and Shamir's attack

Hoch and Shamir [36] generalized the attack proposed by [59] by considering the case when any element that occurs in the sequence is repeated at most q times. We first present a high-level description of the attack. In this thesis, whenever we use the phrase "q successive permutations" for a sequence ψ, we mean that we can rewrite the sequence ψ in the form $\psi = \psi_1 \| \psi_2 \ldots \| \psi_q$, such that ψ_i is a permutation of ψ_j for all $1 \leq i, j \leq q$. For a sequence ψ, the attack has following steps:

1. For any sequence ψ, such that $\mathbb{Z} = alph(\psi)$, there exists a $\mathcal{Y} \subseteq \mathbb{Z}$, such that $\psi|_{\mathcal{Y}}$ is in the form of (up to) q successive permutations.
2. If we can find a 2^r-way collision for a hash function based on $\psi|_{\mathcal{Y}}$, then we can find a 2^r-way collision for the hash function based on ψ.
3. If $\mathcal{Y}$ is a reasonably large set, then we can find a 2^r-way collision on q successive permutations.

We now give a more detailed description of the attack. Throughout this section, we will consider the message in the form of blocks. We denote the set of the first ℓ integers by $L = [\ell]$. The number of collisions to be found is 2^r.

- **Step 1 of the attack: Lower bound on the size of the alphabet set of a sequence to allow successive permutation.**

  Hoch and Shamir proved that any arbitrary sequence based on a certain alphabet size can be reduced to the case of successive permutations. This is the first step of the attack.

  Lemma 3.4. *Let ψ be a sequence such that any element from L occurs in ψ at most q times. For large enough L, and for any integer x, we can find a subset of indices $\mathcal{Y}$, such that $|\mathcal{Y}| \geq x$, and $\psi|_{\mathcal{Y}}$ is in the form of up to q successive permutations over the same set of indices $\mathcal{Y}$.*

  Recently, Halunen *et al.* [34] corrected the lower bound on the size of the alphabet set. We state the correct bound in Theorem 3.7.

- **Step 2 of the attack: Finding a 2^r-way collision on ψ, given a 2^r-way collision on $\psi|_{\mathcal{Y}}$.** We state the extension of the attack from $\psi|_{\mathcal{Y}}$ to ψ for any subset $\mathcal{Y} \subseteq alph(\psi)$ in the form of the following lemma:

  Lemma 3.5. *If we can construct a 2^r-way collision on a hash function based on sequence $\psi|_{\mathcal{Y}}$, we can construct a 2^r-way collision on a hash function based on ψ.*

- **Step 3 of the attack: Finding a 2^r-way collision in the case of successive permutations.** The final step of the attack uses a very nice result on the permutation of any finite set, which we state next:

  Lemma 3.6. *Let B and C be two permutations of L. Divide both permutations into k consecutive groups of size $\sigma = \frac{\ell}{k}$ (say $\{B_i\}_{i\in[\sigma]}$ and $\{C_i\}_{i\in[\sigma]}$). Then for $x \geq 0$, such that $\ell \geq k^3 x$, and for every group B_i, we can find a C_j such that $|B_i \cap C_j| \geq x$.*

The proof of the lemma follows from Hall's theorem and some basic counting principles.

Using the above-stated lemmas, we can construct a 2^r-way collision when the sequence is in the form of successive permutations. Let ψ be a sequence defined on sufficiently large alphabet set. Let $\mathcal{Y}'$ be a subset of $alph(\psi)$ as defined in Lemma 3.4, such that $|\mathcal{Y}'| = \ell$ for $\ell \geq r^3 n^{3(q-3)+2}$. Let $\psi|_{\mathcal{Y}'} = \pi_1 \| \dots \| \pi_q$ be the factoring of $\psi|_{\mathcal{Y}'}$ in the form of q successive permutations. The procedure is as follows:

1. Divide π_q and π_{q-1} into r equal-length intervals. Since $\ell \geq r^3 n^{3(q-3)+2} = r^3 x$, from Lemma 3.6, we can find an intersection set of size $n^{3(q-3)+2}$. Let the disjoint union of the indices from all the intersections be denoted by $\mathcal{Y}$. We modify this set iteratively, and call it the *active* indices.
2. Construct $\psi_1 = \psi|_{\mathcal{Y}}$.
3. Take π_{q-1} and π_{q-2}, and divide them into rn equal length intervals. Follow steps 1 and 2. Since in this case $\ell \geq (rn)^3 n^{3(q-4)+2}$, we have an intersection of size $n^{3(q-4)+2}$.
4. Continue the above procedure until we reach π_3 and π_2 after $q-2$ iterations, where the number of active indices is n^2 for each rn^{q-3} segments. Let $\mathcal{Y}$ be the smallest set of indices formed in the last step of the iteration.
5. Construct 2^k Joux multicollisions on $\psi|_{\mathcal{Y}}$, as in Lemma 3.5.

Halunen *et al.* [34] calculated the query complexity of the attack by Hoch and Shamir. They gave the following result:

Theorem 3.7. [34] *Let m, n, and q be positive integers such that $m > n$ and $q \geq 2$, and let $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ be a compression function on which the iterated hash function is based. Let ψ be a sequence such that $alph(\psi) = [\ell]$. Then there exists a probabilistic algorithm, which, given any $r \in \mathbb{N}$, constructs a 2^r-way collision set on the generalized iterated hash function so that the number of queries made to f in the attack of* [36] *is $O(C_q n^{(q^2-4q+5)D_q})$, where C_q and D_q are defined recursively as,*

$$C_q = (q \times C_{q-1})^{D_q+1}, \; C_1 = 1,$$

and

$$D_q = D_{q-1}^2 + 1, D_1 = 1.$$

It is straightforward to see that $C_q = \Theta(2^{2^{q-1}})$ and $D_q = \Theta(2^{2^{2^q}})$. Due to this super-exponential complexity (in terms of q), the attack is mainly of theoretical interest.

3.3 Second preimage attacks

In this section we will discuss few generic second preimage attacks. One constraint with all the attacks is that they are feasible only when the message is "long". In this thesis, in order to make the equations simpler, we assume that the second preimage attack has been carried on a message which is $L = 2^\ell$ blocks long, an exact power of 2. However, similar analysis can be carried out for different values of L.

We will discuss the recent progress in chronological order:

1. Attack on iterated hash functions [44].
2. Attack on the dithered hash construction [3].
3. Attack on the hash twice construction [2].

3.3.1 Attack on an iterated hash function

In 2005, Kelsey and Schneier [44] extended the idea of Joux's multicollision attack to find second preimages on a long message. They introduced the concept of an expandable message, which formed the basis of many subsequent second preimage attack. We give a brief overview of their attack.

Let M_c be the challenge message, which is 2^ℓ-blocks long. Let the hash function be based on the compression function f. Let $\mathcal{C}$ be a collision finding oracle, which on being queried with a hash value h and an integer t, produces two colliding messages, M and M^*, with block lengths 1 and t respectively, and $h' = H(h, M) = H(h, M^*)$.

Construction of an expandable message.

The attack uses $\mathcal{C}$ to construct an $(\ell, \ell + 2^\ell - 1)$-expandable message. Let h_0 be the initial hash value. The adversary runs $\mathcal{C}$ in each iteration of ℓ rounds. In iteration i, for $0 \leq i \leq \ell - 1$, the adversary queries $\mathcal{C}$ with h_{i-1} and $1 + 2^i$. $\mathcal{C}$ returns a pair of messages, M_i and M_i^*, where M_i is one-block long and M_i^* is $1 + 2^i$ blocks long, and $h_i = H(h_{i-1}, M_i) = H(h_{i-1}, M_i^*)$. At the end of all rounds of iterations, we have two lists of messages.

Constructing an $(\ell, \ell+2^\ell-1)$-expandable message requires concatenation of the elements of the two lists together, in an appropriate way.

Now the question arises that from the two lists, how can the adversary construct a set of messages which have block lengths ranging from ℓ to $\ell + 2^\ell - 1$? The answer is simple: suppose we want to find a message of block length $i + \ell$. We compute the binary representation of $i = \langle i_1, \ldots, i_\ell \rangle$. If $i_j = 1$, then set $m_j = M_j^*$, else pick $m_j = M_j$. It can be easily verified that the resulting message $m = m_1 \| \ldots \| m_\ell$ is a message of the required block length.

The attack can be now summed up in the following steps:

- Find an $(\ell, \ell + 2^\ell - 1)$-expandable message, and find a *linking message*, m^*, that links the hash value of the expandable message set[2] to one of the chaining values of the hash computation of M_c. In more detail, let h be the hash value of the expandable message. m^* is a linking message if $f(h, m^*)$ is one of the chaining value in the computation of M_c. m^* is found by random selection.
- Using the expandable message set, find a message of the required length so that the length of the original message and the second preimage is the same. This step is to overcome the MD-strengthening.

Figure 3.3 shows the attack. In the figure, series of solid vertical lines denotes the message blocks, while the curve line denotes the linking message block.

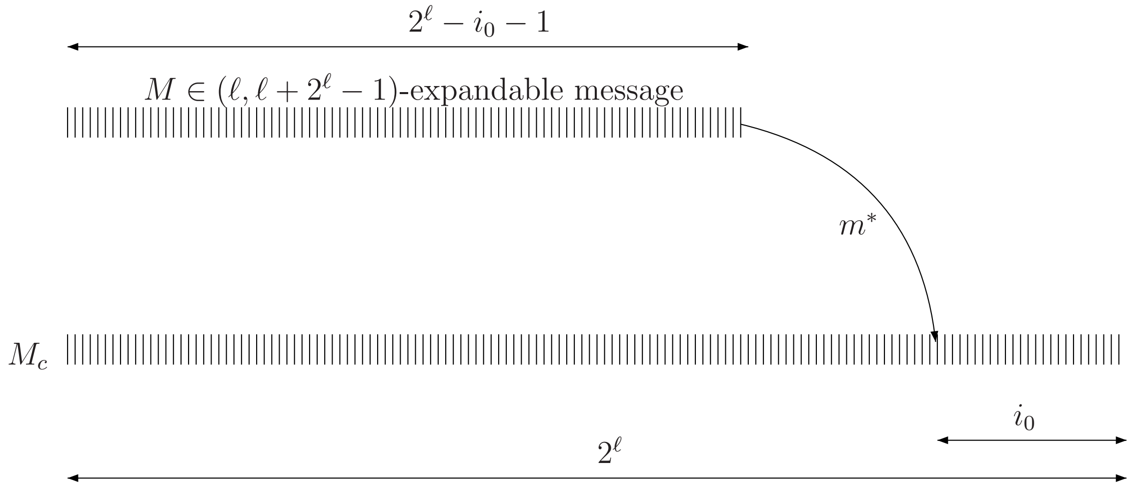


Figure 3.3: Second Preimage attack of Kelsey-Schenier

Kelsey and Schenier calculated the complexity of the attack as $O(2^{n-\ell} + \ell \times 2^{n/2})$.

Subsequently, various hash constructions were proposed to resist the expandable message attack. Rivest proposed a construction that is based on dithering the messages with a sequence to prevent such an attack. However, using the diamond structure, Andreeva *et al.* showed that Rivest's construction is still susceptible to such an attack. Halevi and

[2]Note that the expandable message is by construction, a multicollision set.

Krawczyk [33] proposed a randomized hash construction that resists such attacks. Gauravaram *et al.* [28] proposed a new construction which they called 3C construction to resist this attack. However, this construction was shown susceptible to an easy collision attack that uses the internal construction of 3C construction [26].

3.3.2 Attack on the dithered hash function

Andreeva *et al* [3] found an attack on the dithered hash function using a diamond structure. The expandable message attack fails because the adversary has to match the dithering sequence that are used in the message and its second preimage. The attack by [3] utilizes the flexibility one has when choosing the prefix. For this, the adversary constructs a 2^k-diamond structure based on a subsequence of the original sequence. Let ψ be the dithering sequence used in the dithered hash function. Since the edge of the diamond structure is labelled by message block(s), one also requires the specification as to how the dithering sequence is used for the edge labelling of the diamond structure. Let ω (written $\omega_{(0)}, \ldots, \omega_{(k)}$) be one of the most frequently occuring factor of ψ. The attack then uses ω as the dithering sequence for the edges of the diamond structure by dithering all the messages on the edges that joins the nodes at level i to level $i+1$ by $\omega_{(i)}$. The steps of attack can now be described as:

1. Choose arbitrarily[3] the most frequent factor of ψ of length $k+1$ as ω.
2. Build a 2^k-diamond structure by dithering the edge labelling with the sequence, ω. Let the hash value at the root be h_T.
3. Find a linking block m^*, such that $f(h_T, m^*) = h_i$, $i \geq k+2$, and $\psi[i-k-1, i] = \omega$.
4. Let $j = i - k - 1$. Find a prefix of the required length using the dithering sequence $\psi[1, j]$.

Recall that $Fact_\psi(\ell)$ denote the number of factors of size k in the sequence ψ. Andreeva *et al* made a crucial observation that, although the sequence used in Rivest's construction is non-periodic, $Fact_\psi(k) = O(k)$. Using this fact, they calculated the message complexity to be

$$O(Fact_\psi(k) \times 2^{n-\ell} + 2^{n-k} + 2^{(n+k)/2}) = O(k \times 2^{n-\ell} + 2^{n-k} + 2^{(n+k)/2}).$$

Later, [9] corrected the analysis by noting a correction in the analysis of diamond structure and proved the following:

[3]There can be more than one factor that have same frequency and same length.

Theorem 3.8. [9] *Assuming that hash computation takes unit time, the message complexity of the attack on dithered hash construction is,*

$$O\left(\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k} + k \times 2^{n-\ell}\right),$$

and the computational complexity is,

$$O\left(k \times 2^{n-k} + k \times \ell \times 2^{n-\ell} + n \times \sqrt{k} \times 2^{(n+k)/2}\right).$$

3.3.3 Attack on the hash twice construction

The original attack [2] uses an expandable message to find a second preimage on the hash twice construction. In this thesis, we give a different attack that uses the diamond structure. However, for the sake of completion, we first present the attack of [2].

Using expandable message

In the second preimage attack on the hash twice construction based on the compression function f, the attack of Andreeva *et al* [2] uses the following steps in the offline phase:

1. Generate an $(\ell, 2^\ell + l - 1)$-expandable message, starting from the initial hash value yielding a chaining value h_a.
2. Starting from h_a, generate a set of multicollisions, $\mathcal{M}$ that yields a chaining value h_b. Denote the first $n-k$ blocks of $\mathcal{M}$ by $\mathcal{M}_1$ and the remaining blocks by $\mathcal{M}_2$.
3. Randomly choose 2^k random hash values as the labelling of the leaf nodes, and using the messages from $\mathcal{M}_2$ as the messages on the edge, construct a diamond structure, $\mathcal{D}_2$. It yields a chaining value h_d.

In the online phase, given a message M of block length 2^ℓ, the attacker follows the following steps:

1. Find a linking message m, such that $f(h_d, m)$ equals to a chaining value h_i appearing in the second pass of the hashing of M.
2. Find a message of appropriate length from the $(\ell, 2^\ell + l - 1)$-expandable message. Let the message be m_0.

3. Compute $i' = i - 2^\ell$. Let $M' = M_{i'+1} \| \ldots \| M_{2^\ell}$. Compute $h_{2,1} = f(h_b, M' \| m_0)$. Now, using the messages from $\mathcal{M}_1$, find a linking message m^* that links $h_{2,1}$ to one of the leaf nodes of the diamond structure.

4. Find the path inside the diamond structure to the root of $\mathcal{D}_2$.

The second preimage is the message formed by appending the message blocks in the proper order.

Using diamond structure

We next present our variation of the above attack. In the attack that uses diamond structure instead of an expandable message, the attacker constructs two diamond structures, one on each pass. The first diamond structure is a simple diamond structure while the second diamond structure is a diamond structure based on a set of multicollisions[4]. The steps followed in the offline phase are as follows:

1. Randomly choose 2^k hash values as the labelling of the leaf nodes, and construct a diamond structure $\mathcal{D}_1$ using these hash values as the leaf nodes. Let the hash value of the root node be $h_{1,1}$.

2. Starting from $h_{1,1}$, generate a set of multicollisions, $\mathcal{M}$ that yields a chaining value $h_{1,2}$. Denote the first $n - k$ blocks of $\mathcal{M}$ by $\mathcal{M}_1$ and the remaining blocks by $\mathcal{M}_2$.

3. Randomly choose 2^k random hash values as the labelling of the leaf nodes, and using the messages from $\mathcal{M}_2$ as the messages on the edge, construct a diamond structure, $\mathcal{D}_2$. Let the hash value of the root node be $h_{2,2}$.

The diamond structure $\mathcal{D}_i$ is constructed for the i^{th} pass.

The online phase remains the same as in [2], except that instead of finding a message from the $(\ell, 2^\ell + l - 1)$-expandable message, the attacker finds a linking message of appropriate length that links the initial hash value to one of the leaf nodes of $\mathcal{D}_1$. Figure 3.4 shows the attack. In the figure, series of solid vertical lines denotes the message blocks, the curve line denotes the single message block, the series of closed curve denotes the set of multicollisions. A diamond structure is shown by a triangle and a series of zig-zag lines denotes the messages which are found in the online stage. We will use this notation of figure to draw different structure in the remainder of this thesis.

[4]The edges on the diamond structure are the messages from the set of multicollisions.

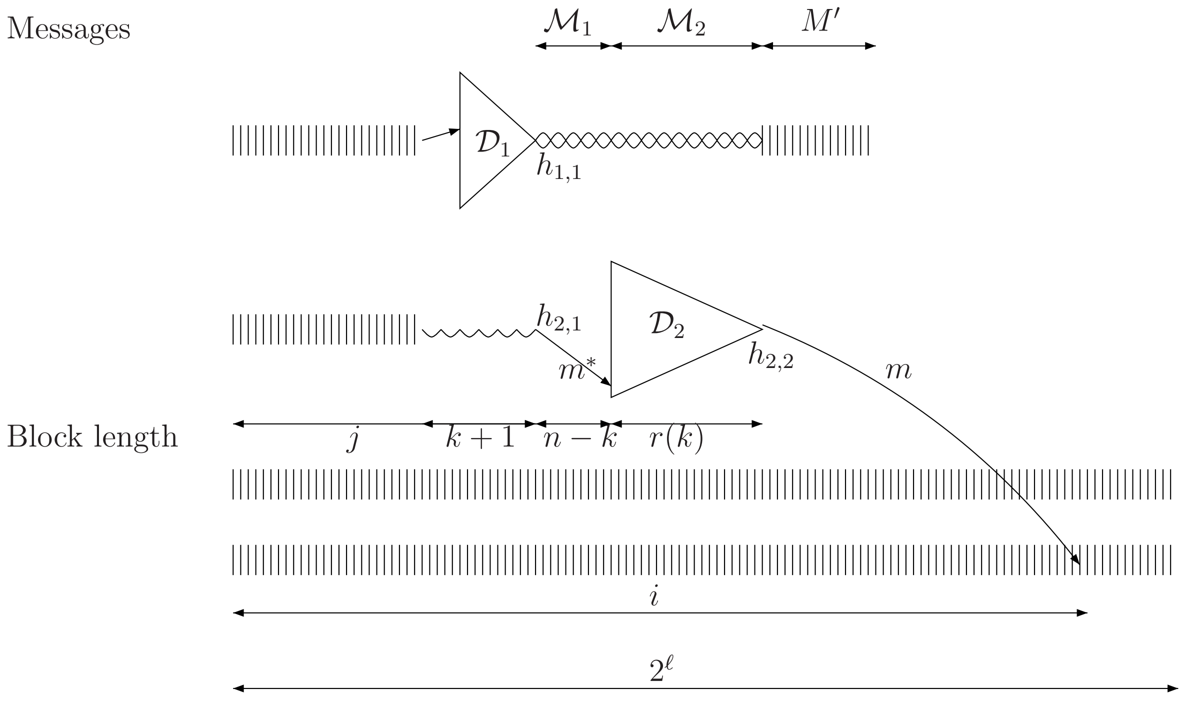


Figure 3.4: Second preimage on the hash twice construction

3.4 Herding attack

3.4.1 Herding attack on an iterated hash function

The herding attack [43] is an attack on the CTFP resistance property of a hash function. Recall that CTFP resistance requires that it should be hard for an adversary to win the following two round game: in the first round, the adversary commits to a hash value h, whereupon she is challenged by a prefix P. In the second round, in order to win the game, the adversary has to produce a suffix S, such that $H(P\|S) = h$.

In the attack that Kelsey and Kohno proposed, in the offline phase, the adversary constructs a *diamond structure* and commits to hash value at the root node.

In the online phase, the adversary is challenged with a prefix P. The adversary computes $H(P)$, and finds a string $\sigma(e_0)$ such that $H(P\|\sigma(e_0))$ is one of the intermediate nodes, ν, of the diamond structure. The adversary then finds the unique path $\mathcal{P} = (e_1, e_2, \ldots, e_k)$ from ν to the root node, and outputs the suffix $S = \sigma(e_0)\|\sigma(e_1)\| \ldots \|\sigma(e_k)$.

Kelsey and Kohno computed the message complexity of the herding attack to be $O(2^{(n+k)/2} + 2^{n-k})$. Recently, using the idea from the random graph theory, [9] corrected their analysis of a diamond structure. We provided a concrete method to construct the diamond structure and proved the following:

Theorem 3.9. [9] *Assuming that a hash computation of every message takes unit time, the message complexity required to construct a 2^k-diamond structure using Kelsey-Kohno's algorithm is $\Theta(\sqrt{k} \times 2^{(n+k)/2})$ and the time complexity is $O(n \times \sqrt{k} \times 2^{(n+k)/2})$. Therefore, for a successful herding attack on an iterated hash function, the message complexity is $\Theta(\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k})$ and the time complexity is $O(n \times \sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k})$.*

By the above theorem, the optimal choice of k is approximately $n/3$ up to a constant factor. We discuss the correct analysis in detail in Chapter 4.

3.4.2 Herding attack on variants of iterated hash function

Andreeva *et al.* [2] showed that the diamond structure can be used for the herding attack on some specific hash function constructions, like the concatenated hash and the hash twice.

In the offline phase, the adversary constructs two diamond structures and multicollisions in the similar manner as in the second preimage attack on the hash twice constructions. The adversary commits to the hash value of the root node of the second diamond structure. Let the hash values of the root nodes of the two diamond structure be $h_{1,1}$ and $h_{2,2}$ respectively, and the common hash value of the multicollisions be $h_{1,2}$.

The only difference between the herding attack on the hash twice and concatenated hash is the chosen-target. In the case of a concatenated hash function, the adversary commits to $h_{1,2} \| h_{2,2}$. In the case of hash twice, she commits to $h_{2,2}$.

In the online phase, once the adversary is given a prefix P, she follows the following steps:

1. Find the linking message m, that links $H(IV, P)$ to one of the leaf nodes of $\mathcal{D}_1$.
2. Find the path through $\mathcal{D}_1$. Let the string formed by concatenating the edge labellings of the path be σ_1.
3. Compute $h_{2,1} = H(h_{1,2}, P \| m \| \sigma_1)$.
4. Using the messages from $\mathcal{M}_1$, she tries to find a linking message m^*, that links $h_{2,1}$ to one of the leaf nodes of $\mathcal{D}_2$.
5. Find the path through $\mathcal{D}_2$. Let the string formed by concatenating the edge labellings of the path be σ_2.

The adversary outputs the suffix as $m \| \sigma_1 \| m^* \| \sigma_2$. Figure 3.5 shows the attack. $\mathcal{D}_1, \mathcal{D}_2, \mathcal{M}_1$, and $\mathcal{M}_2$ are as in Section 3.3.3.

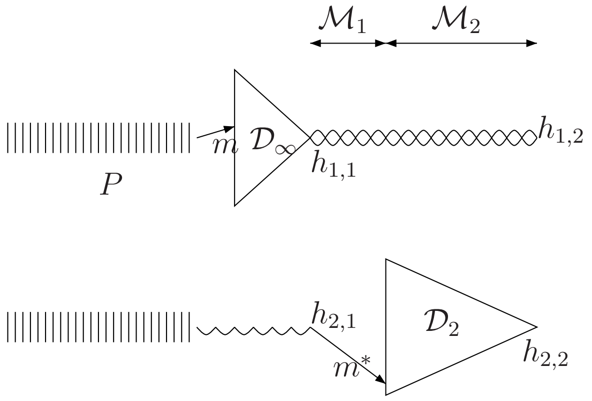


Figure 3.5: Herding attack on the hash twice construction

Chapter 4

Complexity of diamond structure and its impact

In this chapter, we revisit the construction of the diamond structure [43], which is central to many generic attacks [2, 3, 43], and led to the construction of various new iterated hash functions like [50, 52]. Basically, a diamond structure is a structure to produce multicollisions, though more expensive than the Joux multicollisions. However, at the expense of some overhead in constructing multicollisions, the adversary achieves enough flexibility to find a herding message much faster than a naive approach would suggest. In subsequent years after Kelsey and Kohno proposed the diamond structure, it has been used in the second preimage attacks on certain hash constructions that resisted Kelsey-Schneier's expandable message attack, and in the herding attacks on certain variants of iterated hash functions.

The diamond structure is a directed complete binary tree (Section 3.1.2), with the nodes labelled by a hash value and edges by messages[1]. The hash value of any node ν, except for the leaf nodes, is the hash value obtained by applying the compression function on the hash value of either of the child nodes and using the labelling of the edge connecting that child node to ν as the message block. The diamond structure is constructed in such a way that all paths leading to a given node N yield the same value hash value $h(N)$, so $h(N)$ will be well defined. This allows the adversary to commit to the hash value corresponding to the root node. When the adversary is challenged by any prefix, she tries to link the hash of the prefix to one of the intermediate node of the diamond structure, and then find the path leading to the root.

Apart from the herding attack, the diamond structure has many applications. Kelsey and Kohno have enumerated a few applications in [43]. For example, there is a simple

[1]The message can be a single block or multiple blocks.

attack on commitment schemes based on hash functions. In the commit phase, the adversary commits to the hash value of the root node. In the reveal phase, she can reveal a message different from the message that she originally committed to in the commit phase. Apart from this, the diamond structure has been used in many generic attacks on the hash constructions that were constructed specifically to resist the existing generic attacks. Therefore, it is very important to analyse the complexity of the construction of the diamond structure. In the literature, only the message complexity of the construction of the diamond structure has been studied. There has been no work to compute its computational complexity, which is also an important parameter if the attack is actually put in to practice.

In this chapter, we revisit the analysis of the construction of the diamond structure and show that there is a gap between the message complexity and the computational complexity of the construction. We first state the analysis of the diamond structure done by Kelsey and Kohno [43] and point out the problem in their analysis. We follow up with a more detailed analysis. Based on our analysis, we calculate the message complexity and the computational complexity of the construction of the diamond structure and existing attacks based on the diamond structure.

Organization of the chapter

In this chapter, we revisit the earlier analysis done for the construction of the diamond structure. We first present the analysis given by Kelsey and Kohno, which is intuitive but not complete. We prove that it is highly unlikely to construct the diamond structure with the message complexity claimed in [43]. We substantiate our claim by reducing the problem of constructing the diamond structure to a problem of constructing a random graph with a certain property, say $\mathcal{P}$. We then give the bound on the message complexity by finding the number of hash computations required to get the desired probability of collision, which in turn gives the edge probability of a random graph with the property $\mathcal{P}$. Building upon the idea we develop for computing the message complexity, we give the asymptotic bound on the computational complexity of the construction. We observe that the message complexity and the computational complexity of the construction of diamond structure are not equal. In fact, asymptotically, the message complexity differs from the computational complexity by a multiplicative factor of the bit-size of the hash output.

We substantiate our claim regarding the importance of our detailed analysis in Section 4.3 when we analyse the recent attacks based on the diamond structure. We also perform an analysis of the computational complexity of the attacks, and note that it is not the same as the message complexity in most of the cases.

We first quote the analysis of Kelsey and Kohno, which gives us an intuitive idea for the construction of the diamond structure. We give a brief overview of random graphs in Section 4.1. Using the concepts from the theory of random graphs and the analysis of Kelsey and Kohno, we give a revised and rigorous analysis of the construction of the diamond structure in Section 4.2 and study the impact of the revised analysis in Section 4.3.

Analysis of Kelsey and Kohno

We begin by recalling the analysis provided by Kelsey and Kohno [43]. They gave some intuition to compute the message complexity of the construction of the diamond structure. Before we begin, we note that this analysis implies that $n > k$. Kelsey and Kohno argued as follows:

The work done to build the diamond structure is based on how many messages must be tried from each of 2^k *starting values, before each has collided with at least one other value. Intuitively, we can make the following argument, which matches experimental data for small parameters: When we try* $2^{n/2+k/2+1/2}$ *messages spread out from* 2^k *starting hash values (lines), we get* $2^{n/2+k/2+1/2-k}$ *messages per line, and thus between any pair of these starting hash values, we expect about* $(2^{n/2+k/2+1/2-k})^2 \times 2^{-n} = 2^{n+k+1-2k-n} = 2^{-k+1}$ *collisions. We thus expect about* $2^{-k+k+1} = 2$ *other hash values to collide with any given starting hash value.*

We claim that this line of reasoning does not guarantee the diamond structure. In particular, we show that by using the given number of messages, we do not have the guarantee that 2^k nodes can be paired up to give 2^{k-1} collisions. We view the diamond structure in the graph theoretic setting. We translate the problem of constructing diamond structure to the problem of having a random graph with certain properties.

To continue further, we first give a brief exposition of the theory of random graph that is required to understand our approach. We refer the readers to standard references such as [10] for details.

4.1 Random Graph Model

A *random graph* $\mathcal{G}(\nu, p)$ is a graph on ν labelled vertices obtained by selecting each pair of vertices to be an edge randomly and independently with a fixed probability p. This model is commonly known as the *Erdös-Rényi model*. An alternate model is a random graph, $\mathcal{G}(\nu, \mathcal{E})$. In the $\mathcal{G}(\nu, \mathcal{E})$ model, a graph is chosen uniformly at random from the collection of all graphs which have ν nodes and $\mathcal{E}$ edges. The behavior of random graphs is usually studied in the case when ν tends to infinity. p and $\mathcal{E}$ can be fixed or some function of ν.

It is easy to show that the two models are closely related [10]. In our analysis, we will use the $\mathcal{G}(\nu, p)$ model.

The probability p has a very important role, as it can be seen as a parameter which determines the sparsity or density of the graph. As p ranges from 0 to 1, the graph becomes more and more dense on average. Moreover, many natural monotone graph-theoretic properties become true within a very small range of values of p. More precisely, given a monotone graph-theoretic property, there is typically a value of p (which will be a function of ν, the number of vertices) called the *threshold function*. The given property holds in the model $\mathcal{G}(\nu, p)$ with probability close to 0 for p less than the threshold, and the property holds with probability close to 1 for p greater than the threshold. Formally, the notion of a threshold function is defined as follows:

Definition 4.1. *A function $t(\nu)$ is a* threshold function *for a monotone property $\mathcal{P}$ if, whenever $p \ll t(\nu)$, then $\mathcal{G}(\nu, p)$ does not satisfy $\mathcal{P}$ almost always, and whenever $p \gg t(\nu)$, then $\mathcal{G}(\nu, p)$ satisfies $\mathcal{P}$ almost always.*

In random graph theory, threshold functions play a very important role. However, it should be noted that a threshold function is not unique for a particular graph property. For example, $1/\nu + 1/\nu^2$ and $10^{10}\nu^{-1}$ are both threshold functions for the property of containing a triangle.

We next show how we can reduce the problem of constructing a diamond structure to a problem of constructing a random graph with certain properties.

Reduction to a random graph

We construct a graph, $\mathcal{G}$, corresponding to a 2^k-diamond structure. The graph $\mathcal{G}$ has k components, $\mathcal{G}_1, \ldots, \mathcal{G}_k$, where component $\mathcal{G}_i$ corresponds to a level $(k-i)$ of the diamond structure. We only show how to construct the component corresponding to the level $(k-\ell)$.

Suppose we label the nodes at the level $(k-\ell)$ as $1, 2, \ldots, 2^\ell$. We construct a component, $\mathcal{G}_\ell = (\mathcal{V}_\ell, \mathcal{E}_\ell)$ of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in the following manner. The vertex set is $\mathcal{V}_\ell = \{v_1, \ldots, v_{2^\ell}\}$ and $(v_i, v_j) \in \mathcal{E}_\ell$ if the nodes i and j collide at the next level of the diamond structure. It is easy to see that for the construction of the diamond structure, each component should have a perfect matching.

To summarise, we require the graph to have following two properties:

- The graph consists of k components, where component i contains 2^i vertices.
- Each component has a perfect matching.

Note that the second property implies that the graph $\mathcal{G}$ has a perfect matching.

For reference, Figure 4.1(a) shows the list of the hash values (shown as the label of the nodes in the chain) obtained for different messages tried for every node at the level $k-3$ of a diamond structure, and Figure 4.1(b) shows the corresponding component of the graph (a perfect matching is shown by the bold edges). Every edge of the graph corresponds to a collision. For example, vertex 1 and 2 are connected by an edge in the graph because Node 1 and Node 2 collides at a hash value 1324.

Based on the analysis given by Kelsey and Kohno [43], we see that the component $\mathcal{G}_\ell$ is precisely a random graph in $\mathcal{G}_\ell(2^\ell, 2^{-\ell+1})$. Unfortunately, random graph theory tells us that it is very unlikely that there is a perfect matching in a random graph in $\mathcal{G}_\ell(2^\ell, 2^{-\ell+1})$. This is because Erdös and Rényi [23] proved that the threshold function for having a perfect matching in $\mathcal{G}(\nu, p)$ is roughly $\ln \nu/\nu$. The class of random graphs $\mathcal{G}_\ell(2^\ell, 2^{-\ell+1})$ has $p = 2/\nu$, which is well below the required threshold.

More precisely, the threshold function for having a perfect matching is any function having the form

$$t(\nu) = \frac{\ln \nu + f(\nu)}{\nu} \tag{4.1}$$

for any $f(\nu)$ such that $\lim_{\nu\to\infty} f(\nu) = \infty$.

Note also that the threshold function for not having an isolated vertex is $\ln \nu/\nu$ [22], and a graph that contains an isolated vertex obviously cannot contain a perfect matching.

4.2 Correct analysis

In our analysis, in order to simplify the assumptions and algebra, we assume that a random graph in $\mathcal{G}(\nu, \ln \nu/\nu)$ will have a perfect matching. This is perhaps a tiny bit optimistic in view of equation (4.1), but it is close enough for our purposes. To be completely rigorous, we could construct a random graph in $\mathcal{G}(\nu, c \ln \nu/\nu)$, for any constant $c > 1$. However, this would have no impact on our asymptotic computations.

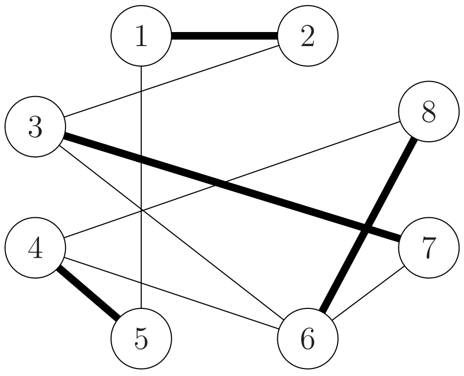
4.2.1 Message complexity

We proceed in a similar manner to [43]. Suppose we construct $\nu = 2^k$ lists, each containing L messages. The probability that any two given messages collide under H is 2^{-n}. The probability that no message in one list collides with a message in another list can be estimated to be

$$\left(1 - \frac{1}{2^n}\right)^{L^2} \approx 1 - \frac{L^2}{2^n}.$$

Node 1	**1324**	6534	**5634**	4323	6535	2346	6435
Node 2	2314	2354	7324	**1324**	**5242**	6452	1246
Node 3	4323	**5242**	6234	**4252**	5223	6253	4212
Node 4	**6534**	**4524**	6235	2462	6423	**5634**	6234
Node 5	5432	**5634**	6234	**3423**	6324	1352	4242
Node 6	2454	**6223**	6246	6235	**6534**	**4252**	**6223**
Node 7	4237	**4252**	6234	6231	6445	6623	6236
Node 8	7345	6424	6234	7364	**4524**	6115	**5623**

(a)



(b)

Figure 4.1: Level $k-3$ of a 2^k-diamond structure

The probability (which we denote by p) that there is at least one collision between two given lists is

$$p \approx \frac{L^2}{2^n}. \tag{4.2}$$

In view of our analysis above, we want

$$p \approx \frac{\ln \nu}{\nu}.$$

Recalling that $\nu = 2^k$, it is clear that we need to take

$$L \approx \sqrt{k \ln 2} \times 2^{(n-k)/2} \approx 0.83 \times \sqrt{k} \times 2^{(n-k)/2}. \tag{4.3}$$

The *message complexity* (i.e., the number of hash computations) at level 0 is therefore

$$2^k L \approx 0.83 \times \sqrt{k} \times 2^{(n+k)/2}. \tag{4.4}$$

From now on, we assume $n > ck$ for some constant $c > 1$ (as we will see later, a typical choice of c for the most optimal attack is approximately 3).

Theorem 4.2. [9] *The total message complexity required to construct a 2^k-diamond structure using the algorithm of Kelsey and Kohno is $\Theta(\sqrt{k} \times 2^{(n+k)/2})$.*

Proof. The entire diamond structure requires finding collisions at levels $0, 1, \ldots, k-1$. The above analysis shows that, at level ℓ, the message complexity is about $0.83 \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2}$. Therefore the total message complexity is

$$\begin{aligned}
\sum_{\ell=0}^{k-1} 0.83 \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2} &= \sum_{i=1}^{k} 0.83 \times \sqrt{i} \times 2^{(n+i)/2} \\
&< 0.83 \times \sqrt{k} \times 2^{n/2} \times \sum_{i=1}^{k} 2^{i/2} \\
&= 0.83 \times \sqrt{k} \times 2^{n/2} \times \frac{2^{1/2}(2^{(k+1)/2}-1)}{2^{1/2}-1} \\
&= O(\sqrt{k} \times 2^{(n+k)/2}).
\end{aligned} \tag{4.5}$$

For the lower bound, we have

$$\begin{aligned}
\sum_{\ell=0}^{k-1} 0.83 \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2} &= \sum_{i=1}^{k} 0.83 \times \sqrt{i} \times 2^{(n+i)/2} \\
&= 0.83 \times 2^{n/2} \sum_{i=1}^{k} \sqrt{i} \times 2^{i/2} \\
&> 0.83 \times \sqrt{\frac{k}{2}} \times 2^{n/2} \times \sum_{i=k/2}^{k} 2^{i/2} \\
&= 0.83 \times \sqrt{\frac{k}{2}} \times 2^{n/2} \times \frac{2^{k/4}(2^{(k/2+1)/2}-1)}{\sqrt{2}-1} \\
&= \Omega(\sqrt{k} \times 2^{(n+k)/2}).
\end{aligned}$$

The result follows. □

4.2.2 Computational complexity

In this section, we look at the computational complexity. The computational complexity of constructing a diamond structure has not been considered until now in the literature. However, this clearly is an important and relevant issue if these structures are ever going to be implemented in a real attack.

There are three main steps required to proceed from one level of the diamond structure to the next. As before, we start by analyzing the work done to go from level 0 to level 1 (the work done at other levels can be analyzed in the same way).

1. Compute $L = 0.83 \times \sqrt{k} \times 2^{(n-k)/2}$ hash values for each of the 2^k lists.
2. Construct the associated graph, i.e., for each pair of lists, determine if there is a common hash value.
3. Determine if the associated graph contains a perfect matching.

Under the assumption that each hash computation takes unit time, the complexity of step 1 is just the message complexity, which we have already computed to be $\Theta(2^k L)$ in equation (4.4).

In step 2, we have to search every pair of lists for a repeated value. Asymptotically, we do not know how to do better than concatenating all the lists, sorting them, and then performing a single pass through the sorted list to detect duplicates. The total time is therefore

$$O(2^k L \log(2^k L)).$$

Recall from equation (4.3) that $L = \Theta(\sqrt{k} \times 2^{(n-k)/2})$. Then we have

$$\begin{aligned}
2^k L \log(2^k L) &= \Theta(\sqrt{k} \times 2^{(n+k)/2} \times \log(\sqrt{k} \times 2^{(n+k)/2})) \\
&= \Theta\left(\sqrt{k} \times 2^{(n+k)/2} \times \left(\frac{n+k}{2} + \frac{1}{2}\log k\right)\right) \\
&= \Theta(\sqrt{k} \times 2^{(n+k)/2} \times n), \qquad (4.6)
\end{aligned}$$

since $k < n$.

In step 3, we need to find a perfect matching in a graph on 2^k vertices. Motwani [56] proved that finding a maximum matching in a graph on ν vertices, with $\mathcal{E}$ edges, and average degree at least $\ln \nu$ can be done in time $O\left((\mathcal{E} \log \nu)/(\log\log \nu)\right)$. The algorithm of Motwani is a randomized algorithm that finds a maximal matching with high probability. Alternatively, we can use the algorithm due to Micali and Vazirani [55], whose running time is $O(\mathcal{E}\sqrt{\nu})$, and does not require any specific property in the graph. However, for our purposes, the randomized algorithm of Motwani suffices. In our case, we have a graph that almost surely contains a perfect matching and the expected number of edges is $k \times 2^k$, so an algorithm that finds a maximum matching will in fact find a perfect matching. This will take time

$$O\left(\frac{k^2 2^k}{\log k}\right).$$

Combining the three steps, we see that the total time required at level 0 is

$$O\left(2^k L + 2^k L \log(2^k L) + \frac{k^2 2^k}{\log k}\right) = O\left(2^k L \log(2^k L) + \frac{k^2 2^k}{\log k}\right).$$

From (4.3), we have $L = \Theta(\sqrt{k} \times 2^{(n-k)/2})$. Now the total computation time is the summation of the time for performing all these steps at each level. From equation (4.6), we can evaluate the total computational time to be,

$$O\left(\sum_{\ell=0}^{k-1}\left(n \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2} + \frac{(k-\ell)^2 2^{k-\ell}}{\log(k-\ell)}\right)\right). \qquad (4.7)$$

The first part of the sum is just the message complexity multiplied by n. For the second part, note that

$$\sum_{\ell=0}^{k-2} \frac{(k-\ell)^2 \times 2^{k-\ell}}{\log(k-\ell)} < \sum_{\ell=0}^{k-2} \left((k-\ell)^2 \times 2^{k-\ell}\right) = \underbrace{\sum_{i=1}^{k} \left(i^2 \times 2^i\right)}_{S} -2. \tag{4.8}$$

We now evaluate the sum S.

$$\begin{aligned} S &= 1^2 \times 2 + 2^2 \times 2^2 + 3^2 \times 2^3 + \cdots k^2 \times 2^k & (4.9) \\ 2S &= 1^2 \times 2^2 + 2^2 \times 2^3 + 3^2 \times 2^4 \cdots (k-1)^2 \times 2^k + k^2 \times 2^{k+1}. & (4.10) \end{aligned}$$

(4.10)−(4.9) yields

$$S = k^2 \times 2^{k+1} - \sum_{i=1}^{k} (i^2 - (i-1)^2) \times 2^i = O(k^2 \times 2^k). \tag{4.11}$$

Since, $n > ck$ and $c > 1$, combining equations (4.7) and (4.11), we have the total computation time as,

$$O(n \times \sqrt{k} \times 2^{(n+k)/2} + k^2 \times 2^k) = O(n \times \sqrt{k} \times 2^{(n+k)/2}).$$

We can summarize it as the following theorem.

Theorem 4.3. [9] *If each hash computation takes unit time, the computational complexity of constructing a 2^k-diamond structure by the algorithm of Kelsey and Kohno is*

$$O(n \times \sqrt{k} \times 2^{(n+k)/2}). \tag{4.12}$$

Therefore, the computational complexity is up to n times the message complexity.

Remark 4.4. *Before we proceed, we remark that the above analysis was straightforward but a slight simplification of the real picture. To be specific, the Erdös-Rényi model does not exactly capture the way in which the diamond structure is constructed. For more rigorous analysis, one has to reduce the problem of the construction of a diamond structure to the problem of the construction of a random intersection graph with the property $\mathcal{P}$. However, none of the results change when we carry out the analysis in the random intersection graph model. We refer the readers to [9] for details.*

Remark 4.5. *We can be more precise with our analysis by parameterizing the complexity of the hash computations. Let each hash computation take t elementary operations. Then the message complexity of the construction of a 2^k-diamond structure is $\Theta(t \times \sqrt{k} \times 2^{(n+k)/2})$*

and the computational complexity is

$$O(t \times \sqrt{k} \times 2^{(n+k)/2} + n \times \sqrt{k} \times 2^{(n+k)/2}). \tag{4.13}$$

However, we will use Theorem 4.12 to compute the computational complexity of the attacks based on a diamond structure. For more rigorous result, reader can use the equation (4.13).

4.3 Impact on previous attacks

Earlier we made a comment that the diamond structure is an important data structure in many generic attacks, and therefore it is important to have a correct complexity analysis of the diamond structure. In this section, we review the complexity of those attacks in the light of our corrected analysis and compare it with the estimate given in the respective works. We do not perform the analysis for the herding attack on a simple iterated hash function because it follows easily from the analysis of the diamond structure. We give a detailed analysis for the second preimage attack on the dithered hash function and hash twice constructions, and the herding attack on the hash twice construction.

4.3.1 Herding attack

On the hash twice construction

Andreeva *et al.* [2] proposed a herding attack on the hash twice function by building a diamond structure based on multicollisions. Figure 3.5 shows the attack. The attack constructs Joux's multicollisions on the first pass of the hash function, and then uses the messages that caused the multicollisions to build a diamond structure on the second pass. In total, the attack requires the construction of two diamond structures, one on each of the two passes. In fact, the adversary performs the following steps during the offline phase:

1. Construct a 2^k-diamond structure (let us say $\mathcal{D}_1$) for the first pass by randomly choosing 2^k hash values as the label of the leaves. Let the hash value at the root node of $\mathcal{D}_1$ be h'.
2. Construct 2^{n-k+r}-multicollisions (let us say $\mathcal{M}$) starting from the root node of $\mathcal{D}_1$, where the value of r is to be calculated later. Let the hash value at the end of multicollisions be h_1.
3. Construct a diamond structure (let us say $\mathcal{D}_2$) on the second pass by randomly choosing 2^k hash values as the label of the leaves, and using the messages from the last r blocks of messages in $\mathcal{M}$.

The adversary commits to the hash value at the root of $\mathcal{D}_2$.

In the online phase, once the adversary is given a prefix P, she does the following:

1. Calculate $H(IV, P)$ and then find a linking message, m that links $H(IV, P)$ to one of the leaves of $\mathcal{D}_1$.
2. Find the path inside $\mathcal{D}_1$ to the root of $\mathcal{D}_1$. Let the message formed by concatenating the messages on the edge of the path be σ.
3. Calculate $H(h_1, P\|m\|\sigma)$. She uses the first 2^{n-k}-multicollisions of $\mathcal{M}$ to find the linking message, m^* that links $H(h_1, P\|m\|\sigma)$ to one of the leaves (say N) of $\mathcal{D}_2$.
4. Find the path from N to the root node of $\mathcal{D}_2$.

The adversary outputs the concatenation of m, $\sigma\|m^*$, and the edge labellings of the path through $\mathcal{D}_2$.

Andreeva *et al.* [2] calculated the total message complexity of the attack to be $O(2^{n-k} + 2^{(n+k)/2})$. We perform a more detailed analysis of the attack and compute the message complexity and the computational complexity of the attack.

Message complexity

Step 1 is the construction of a simple diamond structure for which we gave a detailed analysis in Section 3.1.2. To calculate the complexity of Step 2, we need to calculate the value of r, the number of blocks of messages required to construct the second diamond structure. Since we need $0.83 \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2}$ messages to herd from level ℓ to level $(\ell+1)$, the total number of multicollisions required to construct the diamond structure is

$$C = \sum_{\ell=0}^{k-1} 0.83 \times \sqrt{k-\ell} \times 2^{(n+k-\ell)/2} = \Theta(\sqrt{k} \times 2^{(n+k)/2}).$$

The last equality is due to equation (4.5).

Since $n > k$ and $r = \log C$, the message complexity to find a 2^{n-k+r}-multicollisions is

$$\Theta\left((n-k+\log C)\, 2^{n/2}\right) = \Theta(n \times 2^{n/2}) \tag{4.14}$$

For Step 3, we calculate the message complexity to go from level 0 to level 1. The analysis of the work done for the other levels can be done analogously. Note that every message that needs to be hashed is in fact a message of block length $\log 2^k L$ and there are

$2^k L$ such messages, where $L \approx 0.83 \times \sqrt{k} \times 2^{(n-k)/2}$. From equation (4.6), the total hash computation is

$$2^k L \log 2^k L = \Theta(n \times \sqrt{k} \times 2^{(n+k)/2}).$$

Therefore, the message complexity to construct $\mathcal{D}_2$ is

$$\Theta\left(\sum_{\ell=0}^{k-1} \sqrt{k-\ell} \times 2^{(n+k-\ell)/2} \times n\right) = \Theta(n \times \sqrt{k} \times 2^{(n+k)/2}). \tag{4.15}$$

For the online phase, the total message complexity is the number of hash computations required to find the two linking messages (one to $\mathcal{D}_1$ and one to $\mathcal{D}_2$). The work done to find the linking message to $\mathcal{D}_1$ is simply 2^{n-k}. However, the message complexity to find the linking message to $\mathcal{D}_2$ is $O((n-k) \times 2^{n-k})$, because each message is $(n-k)$ blocks long. Therefore, the total message complexity in the online phase is

$$O((n-k) \times 2^{n-k}) = O(n \times 2^{n-k}) \tag{4.16}$$

Combining equations (4.5) and (4.14)−(4.16), the message complexity for the attack is

$$O\left(\sqrt{k} \times 2^{(n+k)/2} + n \times \left(2^{n/2} + \sqrt{k} \times 2^{(n+k)/2} + 2^{n-k}\right)\right) = O\left(n \times \left(\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k}\right)\right).$$

This corrects the result in [2].

Computational complexity

Since $\mathcal{D}_1$ is a simple diamond structure, we note that the time required to construct $\mathcal{D}_1$ is the same as equation (4.12). Also, Step 2 is just finding the required number of multicollisions. Therefore, its computational complexity equals the message complexity of finding the required number of multicollisions, which is equal to equation (4.14). For the final part of the attack, we analyze each step of Section 4.2.2 for the construction of $\mathcal{D}_2$ for level 0 (the other levels are analyzed analogously). Recall that to proceed from one level to the next in a diamond structure, we first hash certain lists of messages, construct an associated graph by creating an edge when a pair of lists has a common hash value, and then find a perfect matching on that graph.

For the construction of $\mathcal{D}_2$, we have L lists of messages for each of 2^k hash values, but now all the messages are $\log(2^k L)$ blocks long. Therefore, the total time required to hash the $2^k L$ lists is

$$2^k L \log(2^k L) = \Theta(\sqrt{k} \times 2^{(n+k)/2} \times n).$$

The time complexity to construct the associated graph and then to find a perfect matching in that graph for $\mathcal{D}_2$ is the same as in the construction of a simple diamond structure. This is because we need to sort the same number of hash values (which defines the complexity of the construction of the graph) and we have the same number of vertices, the same expected number of edges, and the same average degree (which determines the complexity of finding a perfect matching). From equation (4.6), the total time required at level 0 is,

$$O\left(2^k L\log(2^k L) + 2^k L\log(2^k L) + \frac{k^2 2^k}{\log k}\right) = O\left(2^k L\log(2^k L) + \frac{k^2 2^k}{\log k}\right).$$

Note that this is same as in the construction of simple diamond structure. Hence, the time complexity of the construction of $\mathcal{D}_2$ is the same as equation (4.12).

For the online phase, we need to find the message that links $H(P)$ to one of the leaf node of $\mathcal{D}_1$. Also, we need to find a linking message to the second diamond structure. Without loss of generality, we can assume that the hash values at the leaves of either of the two diamond structure is sorted. If this is not the case, we first sort the hash values, which takes a computational time of $O(k \times 2^k)$. This has no effect on the asymptotic computation as the computational cost for sorting get subsumed in the computational cost to find the linking message to $\mathcal{D}_2$. Therefore, we can perform a binary search on either of the two diamond structure for each candidate of linking message. Thus the computational complexity of the online phase is k times the message complexity of the online phase.

Therefore, the total time complexity of the attack (pre-computation and online phase) is

$$O\left(n \times \left(2^{n/2} + \sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k}\right)\right) = O\left(n \times \left(\sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k}\right)\right).$$

4.3.2 Second preimage attacks

Dithered hash function

The dithered hash construction was proposed by Rivest [64] to counter the second preimage attack of Kelsey and Schenier [44]. However, recently Andreeva *et al.* [3] proposed an attack based on the diamond structure. For a challenge message M of block length 2^ℓ, the adversary first finds a linking message that links the root of the diamond structure to one of intermediate hash values in the hash computation of M. She then finds a linking message that hashes to one of the intermediate hash values (say corresponding to the node N_0) in the diamond structure. The adversary then finds the path inside the diamond structure that leads from N_0 to the root.

The success of the attack depends on the fact that although dithering sequences are non-periodic, they have many factors of large size. The attack uses such a factor as the dithering sequence for the messages that are used in the construction of diamond structure. If $Fact_{\psi}(k)$ denote the number of factors of size k in the sequence ψ, then Andreeva *et al.* evaluated the message complexity of their attack as

$$O(Fact_{\psi}(k) \times 2^{n-\ell} + 2^{n-k} + 2^{(n+k)/2}).$$

They also stated following two lemmas about the two dithering sequences advocated by Rivest.

Lemma 4.6. *For $k \leq 85$, for the Keränen sequence* [45, 46] *ψ, we have*

$$Fact_{\psi}(k) \leq 8k + 332.$$

Lemma 4.7. *Let $\mathbf{c}$ denote the sequence obtained by diluting the Keränen sequence ψ with a 13-bit counter* [64]*. Then for every $k \in [0, 2^{13}]$, we have*

$$Fact_{\mathbf{c}}(k) = 8k + 32760.$$

In other words, if we represent the above dithering sequence by ψ, the number of factors of size ℓ is bounded by

$$Fact_{\psi}(k) = O(k).$$

Using Lemma 4.6 and Lemma 4.7, the message complexity evaluated by [3] for the second preimage of a 2^{l}-block message, using a 2^{k}-diamond structure is,

$$M(k) = O(k \times 2^{n-\ell} + 2^{n-k} + 2^{(n+k)/2}).$$

We review this analysis in the light of our analysis of diamond structure.

Message complexity

In the attack on the dithered hash function, Andreeva *et al.* chose a factor of the sequence that has an appropriate length. They used the same element of the factor to dither the message on all the edges at the same level of the diamond structure, and the last element of the factor to dither the message that connects the root of the diamond structure to M. Thus, for a 2^{k}-diamond structure, they need a factor of size $(k+1)$. In the worst case, when all the factors have same size in the dithering sequence, the probability that a randomly chosen factor of size $k+1$ in the sequence ψ is the one used in the construction of diamond structure is $(Fact_{\psi}(k+1))^{-1}$. Therefore, the message complexity to connect the root of the diamond structure to M is

$$O(Fact_{\psi}(k+1) \times 2^{n-\ell}) = O(k \times 2^{n-\ell}). \tag{4.17}$$

Suppose the root gets linked to the i^{th} iteration of the hashing of M.

We already calculated the message complexity of the construction of diamond structure in Theorem 4.2. To find the complexity of the attack, note that the attacker needs to hash a message (let us say M') of block length $i-2-k$, where k is the depth of diamond structure, in order to defy the Merkle-Damgård strengthening. Since $k < n$, we see that this has no impact on the asymptotic calculation as it get subsumed by the construction of the diamond structure (we use this fact later on and never explicitly mention it). The message complexity to find a linking message that links $H(M')$ to one of the leaves of the diamond structure is $O(2^{n-k})$. We calculate the message complexity of the second preimage attack on the dithered hash function as

$$O\left(2^k + \sqrt{k} \times 2^{(n+k)/2} + 2^{n-k} + k \times 2^{n-\ell}\right) = O\left(\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k} + k \times 2^{n-l}\right).$$

Under the assumption that $l \approx k$, it can be further simplified to

$$M(k) = O\left(\sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k}\right).$$

Computational complexity

We note that the attack uses the simple diamond structure as constructed in Section 4.2. Hence, the total time required to construct the diamond structure is as stated in Theorem 4.3. For the time required in the online phase, the attacker has to find two linking messages,

1. The message that links the root of the diamond structure to one of the intermediate hash values in the hash computation of M. We can safely assume that the intermediate hash values in the hash computation of M are sorted and hence we can perform binary search. If not, then we first sort them in the online phase, which takes $O(\ell \times 2^{\ell})$ time. Note that it does not effect the computational complexity, because it get subsumed by the construction of the diamond structure (we implicitly use this fact again in the analysis of second preimage attack on hash twice construction). Hence, the time taken is a multiplicative factor of ℓ more than the message complexity (equation (4.17)). This results in $O(\ell \times k \times 2^{n-\ell})$ time to link the message.

2. The message that links the source node of the diamond structure to one of the leaf node. Arguing in the similar fashion as above, the time complexity to find that leaf

is $O(k \times 2^{n-k})$.

Therefore, the total time required for the attack is given by

$$T(k) = O\left(k \times 2^{n-k} + k \times \ell \times 2^{n-\ell} + n \times \sqrt{k} \times 2^{(n+k)/2}\right).$$

Under the assumption that $k \approx \ell$, we have

$$\begin{aligned} M(k) &= O\left(\sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k}\right). \\ T(k) &= O\left(k^2 \times 2^{n-k} + n \times \sqrt{k} \times 2^{(n+k)/2}\right). \end{aligned}$$

It is easy to check that the message complexity is minimized when k is a solution of the equation

$$n - 3k + \log k = 0,$$

which is roughly $n/3$. It is easy to note that the message complexity of the attack is dominated by $O(k \times 2^{n-k})$ for $k < n/3$ and therefore it is the same as in [3]. However, for large values of k, a factor of $\sqrt{k}$ comes into the picture.

Also, the time complexity is minimized when k satisfies the following equation:

$$n - 3k + 3\log k - \log n = 0$$

We present a comparison of the computational complexity of the attack with the message complexity and compare it with the analysis of [3] in Table 4.1.

n	Example	$k = \lfloor n/3 \rfloor$			$k = \lfloor 2n/5 \rfloor$		
		[3]	Actual		[3]	Actual	
			Lower	Upper		Lower	Upper
128	MD5	2^{89}	2^{91}	$2^{91.5}$	2^{90}	2^{93}	$2^{93.5}$
160	SHA-1	2^{111}	2^{113}	$2^{113.5}$	2^{112}	2^{115}	$2^{115.5}$
192	Tiger	2^{132}	2^{134}	$2^{134.5}$	2^{134}	2^{138}	$2^{138.5}$
256	SHA-256	2^{175}	2^{177}	$2^{177.5}$	2^{179}	2^{183}	$2^{183.5}$
512	Whirlpool	2^{347}	2^{349}	$2^{349.5}$	2^{358}	2^{362}	$2^{362.5}$

Table 4.1: Message complexity of second preimage attack on the dithered hash.

Hash twice construction

Let M be a 2^l-block long challenge message for the adversary. The steps followed by the adversary are similar to the herding attack. The only additional step for the adversary is in the online phase when she creates the prefix of the required block length that links to one of the leaves of $\mathcal{D}_1$ and has to find a linking message that links the root of $\mathcal{D}_2$ to one of the intermediate hash values in the second pass of M. In [2], the authors computed the message complexity of the attack to be $O(2^{n-k} + 2^{(n+k)/2} + 2^{n-l})$. We perform a more detailed analysis and tabulate the effect of the difference in the Table 4.2.

Let M be a 2^{ℓ}-block challenge message for the adversary. The adversary constructs two diamond structures, $\mathcal{D}_1$ and $\mathcal{D}_2$, in a similar manner as in the herding attack on the hash twice construction. The additional steps for the adversary are in the online phase when she finds a linking message that links the root of $\mathcal{D}_2$ to one of the intermediate hash values in the second pass of M and when she finds a prefix of the required block length that links to one of the leaves of $\mathcal{D}_1$. The total hash computations to find the linking message to $\mathcal{D}_1$ is $O(2^{n-k})$, and to find the linking message from the root of $\mathcal{D}_2$ to one of the intermediate hash values in the second pass of M is $O(2^{n-l})$. However, every message that the adversary uses to link the source node of $\mathcal{D}_2$ to one of its leaf node is $n-k$ blocks long. Therefore, the number of hash computations the adversary has to perform is $O((n-k) \times 2^{n-k})$. Thus the total message complexity of the online phase is

$$O(2^{n-\ell} + (n-k) \times 2^{n-k} + 2^{n-k}) = O(2^{n-\ell} + n \times 2^{n-k}).$$

Using the above equation and equation (4.15), the message complexity is,

$$O\left(n \times (2^{n-k} + \sqrt{k} \times 2^{(n+k)/2}) + 2^{n-\ell}\right).$$

Arguing in a similar fashion for online phase as in Section 4.3.2, we can say that the time complexity for linking the message to the diamond structure is k times the message complexity, and the time complexity for linking the root of $\mathcal{D}_2$ to one of the intermediate hash value in the hash computation of M is l times its message complexity. Therefore, the time complexity is,

$$O\left(n \times (\sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k}) + \ell \times 2^{n-\ell}\right).$$

Remark 4.8. *We can analyze the time and the message complexity of the herding attack on concatenated hash functions of the form*

$$H^{f_1}(M) \| H^{f_2}(M),$$

where f_1 and f_2 can be the same or different compression functions, in a similar fashion as in Section 4.3.1. This is because the basic construction is the same as for the hash twice function. The only difference lies in the commit stage, where the adversary commits to the

n	Example	$k = \lfloor n/3 \rfloor$			$k = \lfloor 2n/5 \rfloor$		
		[2]	Actual		[2]	Actual	
			Lower	Upper		Lower	Upper
128	MD5	2^{85}	2^{95}	$2^{95.5}$	2^{90}	2^{100}	$2^{100.5}$
160	SHA-1	2^{107}	2^{117}	$2^{117.5}$	2^{112}	2^{122}	$2^{122.5}$
192	Tiger	2^{128}	2^{139}	$2^{139.5}$	2^{134}	2^{145}	$2^{145.5}$
256	SHA-256	2^{171}	2^{182}	$2^{182.5}$	2^{179}	2^{191}	$2^{191.5}$
512	Whirlpool	2^{341}	2^{354}	$2^{354.5}$	2^{358}	2^{371}	$2^{371.5}$

Table 4.2: Message complexity of second preimage attack on the hash twice construction.

$h_1 \| h_2$, where h_1 is the hash value as in Step 2 of the herding attack on hash twice, and h_2 is the hash value of the root of $\mathcal{D}_2$ constructed in the attack.

We list the impact of the revised analysis on the herding attacks and the second preimage attacks in Table 4.3 and Table 4.4.

	Message Complexity
Herding attack on Merkle-Damgård	$O(\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k})$
Second Preimage on Dithered Hash	$O(\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k} + k \times 2^{n-\ell})$
Herding attack on Hash Twice	$O(n \times (\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k}))$
Second Preimage on Hash Twice	$O(n \times (\sqrt{k} \times 2^{(n+k)/2} + 2^{n-k}) + 2^{n-\ell})$

Table 4.3: Message complexity for various attacks

	Time Complexity
Herding attack on Merkle-Damgård	$O(n \times \sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k})$
Second Preimage on Dithered Hash	$O(n \times \sqrt{k} \times 2^{(n+k)/2} + k \times 2^{n-k} + \ell \times k \times 2^{n-\ell})$
Herding attack on Hash Twice	$O(n \times (k \times 2^{n-k} + \sqrt{k} \times 2^{(n+k)/2}))$
Second Preimage on Hash Twice	$O(n \times (k \times 2^{n-k} + \sqrt{k} \times 2^{(n+k)/2}) + \ell \times 2^{n-\ell})$

Table 4.4: Time complexity for various attacks

4.4 Conclusion

In this chapter, we pointed out that the analysis of the diamond structure proposed by Kelsey and Kohno is not complete and almost surely does not yield the desired structure.

We also gave a rigorous analysis of the construction of the diamond structure using concepts from the theory of random graphs. We further investigated the consequences of this analysis. In summary,

1. Our analysis showed that the message complexity of the construction of a 2^k-diamond structure is about $\sqrt{k}$ times more than what [43] claimed (Theorem 4.2).

2. We also showed that the computational complexity of the construction of a diamond structure is up to n times the message complexity (Theorem 4.3).

3. For the Rivest's construction, the ratio of the computational complexity and the message complexity of the second preimage attack is k if we choose the value of k to be strictly less than $n/3$, and n if we choose k to be larger than $n/3$.

4. For the herding attack and the second preimage attack on the hash twice construction and the concatenated hash function, the ratio of the computational and the message complexity is linear in k when k is strictly less than $n/3$, and it is constant for larger values of k.

Chapter 5

Diamond structure in multiple-herding attack

In Chapter 4, we performed a detailed analysis of the generic attacks based on a diamond structure. However, all the attacks find a single herding message (one suffix for every prefix given by the challenger) or a single second preimage. Moreover, the attacks are on hash functions with a specific form, like the hash twice, concatenated hash, and Zipper hash. In this chapter, we propose an attack that uses the diamond structure to find multiple herding messages on generalized sequential hash functions (GSHF), defined by Nandi and Stinson [59]. For this attack, we propose a *kite* like structure, which is essentially a diamond structure appended to the binary tree. Our generalization can be viewed as:

1. A generalization of a single herding attack of Andreeva *et al.* [2] to the case of multiple herding attack.
2. A generalization of the herding attack on specific hash constructions to a more general class of iterated hash functions.

Organization of the chapter

In this chapter, we propose 2^κ-way herding attack on GSHF proposed by Nandi and Stinson [59] (see Definition 2.14). Our attack uses a variant of the diamond structure, called a *kite* structure, which we introduce in Section 5.1. We break our attack on GSHF in following steps:

1. In the first step, we give the basic idea of our attack by extending the herding attack of Andreeva *et al.* on the hash twice construction (Section 5.2.1).

2. We tackle the more general problem in two stages (Section 5.2.2).
   (a) We first handle the simple case where every symbol is used exactly twice (Theorem 5.1).
   (b) In the second stage, we give the bound on the number of message blocks that makes GSHF susceptible to our attack (Theorem 5.2).

5.1 Kite structure

The multiple herding attack on GSHF and the multiple second preimage attack on the hash twice construction (see Appendix A) requires the construction of a variant of the diamond structure, which we call *kite* structure. The kite structure is parameterized by κ, which defines the size of the data structure.

A 2^{κ}-kite structure is comprised of two binary trees glued together so that the leaves of the two binary tree are common to both of them. It can also be viewed as a diamond structure with a unique path of length κ from the source node to every leaf of the diamond structure.

The source node of a kite structure is at level $(-\kappa)$ and the root node is at level κ. There are $2^{\kappa-|\ell|}$ nodes at level ℓ for $-\kappa \leq \ell \leq \kappa$, where $|.|$ denotes the absolute value. A kite structure is similar to a diamond structure from level 0 to level κ and only differs from level $-\kappa$ to level 0. Therefore we just define the labelling of the edges of the kite structure when the node ν is at level $\ell < 0$. For every level ℓ, we pick two strings σ_ℓ and σ_ℓ^* and label the edge directed to the left child of ν by σ_ℓ and the edge directed to the right child of ν by σ_ℓ^*. The nodes of the kite structure are labelled similarly as in the diamond structure.

Figure 5.1 shows a 2^3-kite structure. The source node is h_{-3} and the root node is h_3. The adversary commits to h_3.

5.2 Multiple herding attack on generalized family of hash functions

In this section, we show how we use the kite structure to find multiple herding messages on GSHF. Recall that the hash twice construction is a special case of GSHF. To give an idea of the attack, we start with the multiple herding attack on the hash twice construction.

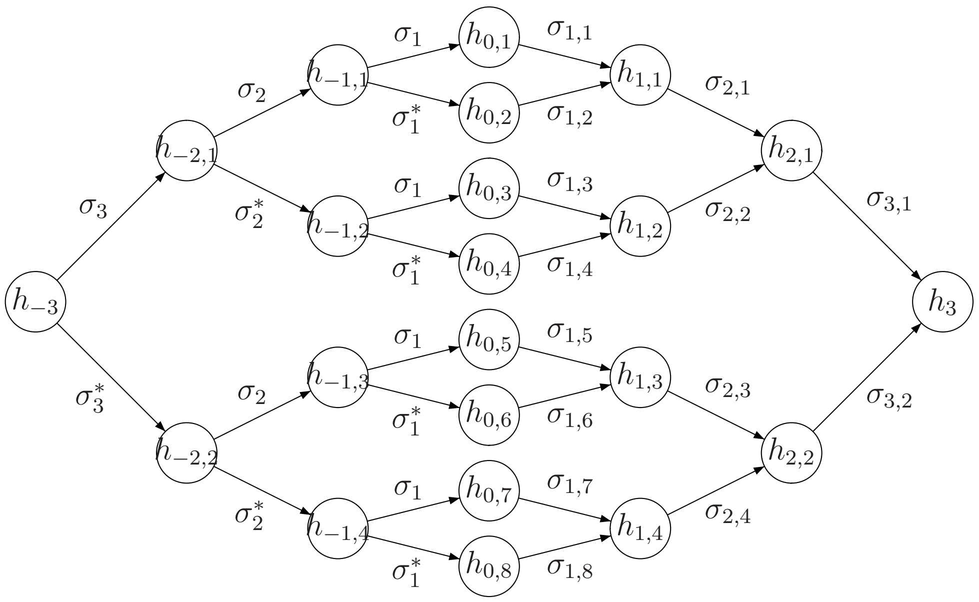


Figure 5.1: A 2^3-kite structure

5.2.1 Multiple herding attack on the hash twice hash function

In this section, we propose a 2^κ-way herding attack on the hash twice construction. Let the challenge prefix be P. On a high level, the attack is based on the herding attack of Andreeva *et al* [2] on the hash twice construction. In addition to their construction, we append a kite structure based on multicollisions. This allows us to have multiple suffixes. Now we give a more detailed description of the attack.

Offline phase

Let the underlying compression function be $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. We construct the following data structure for 2^κ-herding attack on the hash twice construction. In the context of the following construction, whenever we use the phrase "diamond structure based on multicollisions", we mean that the strings on the edges of the diamond structure are chosen from a set of multicollisions. We covered the construction of such a diamond structure in Section 4.3.1. We denote the number of message blocks required to construct such 2^k-diamond structure by $r(k)$. Note that in general, k is not equal to κ.

We define the steps followed for the construction of the underlying data structure that is used in the attack. We show the attack in Figure 5.2.

Construction of the data structure.

- On the first pass:

1. For some $k \geq \kappa$,randomly pick 2^k different hash values and using these hash values as the labels of nodes at level 0, construct a diamond structure $\mathcal{D}$. Let the label of the root node of $\mathcal{D}$ be $h_{1,1}$.
2. With $h_{1,1}$ as the initial hash value, construct a set of $2^{(n-k)}$-multicollisions. Let this set be $\mathcal{M}_1$. Construct another set of $2^{r(k)}$-multicollisions appended in front of $\mathcal{M}_1$, where $r(k)$ is to be calculated later. Let this set of multicollisions be $\mathcal{M}_2$. Let the common hash value of $\mathcal{M}_2$ be $h_{1,2}$.
3. With $h_{1,2}$ as the starting hash value, construct a set of 2^κ-multicollisions. Let this set of multicollisions be $\mathcal{M}_3 := \{(m_1, m_1^*) \times (m_2, m_2^*) \times \ldots \times (m_\kappa, m_\kappa^*)\}$ and the common hash value of this set by $h_{1,3}$.
4. With $h_{1,3}$ as the initial hash value, construct a set of $2^{r(\kappa)}$-multicollisions. Let the common hash value of this set be $h_{1,4}$ and the set be $\mathcal{M}_4$.

In summary, the adversary constructs *four* different sets of multicollisions. The adversary uses $\mathcal{M}_2$ and $\mathcal{M}_4$ to construct two different diamond structures, $\mathcal{D}_1$ and $\mathcal{D}_2$, in the second pass. She use $\mathcal{M}_1$ during the online phase to find the label of the edge between the source node of $\mathcal{D}_1$ and one of the leaf node of $\mathcal{D}_1$, and $\mathcal{M}_3$ to construct a complete binary tree, whose leaves are the nodes at level 0 of $\mathcal{D}_2$. We follow up with the details.

- On the second pass:

  1. Randomly pick 2^k hash values. With these hash values as the nodes at level 0 and using the multicollision set $\mathcal{M}_2$, construct a diamond structure $\mathcal{D}_1$. Let the label of the root node of $\mathcal{D}_1$ be $h_{2,2}$.
  2. With $h_{2,2}$ as the label of the source node, construct a kite structure as below:
     (a) Using $h_{2,2}$ as the label of the root node, construct a complete binary tree iteratively as follows.
     Label the left edge of any node at level i by m_{i+1} and the right edge by m_{i+1}^*. The labelling of the nodes at level $(i+1)$ is done in the following manner. If the label of a node at level i is h, then the left child of the node at level i is $f(h, m_{i+1})$ and the right child of the node at level i is $f(h, m_{i+1}^*)$. At the end of above steps, we get a complete binary tree $\mathcal{T}$, whose root is labelled by the hash value $h_{2,2}$, and the leaves are labelled by 2^κ different hash values. We denote this set of hash values of the leaf nodes by $\mathcal{H} := \{h_j : 1 \leq j \leq 2^\kappa\}$.
     (b) Using $\mathcal{H}$ as the labelling of the nodes at level 0, complete the kite structure by constructing a diamond structure $\mathcal{D}_2$ based on the multicollision set $\mathcal{M}_4$. Let the label of the root node be $h_{2,4}$.

The adversary commits to $h_{2,4}$.

Assume that we have a set of $2^{r(k)}$-multicollisions, where k is the parameter that defines the size of diamond structure. From equation (4.15), the message complexity of the construction of a diamond structure based on a multicollision set is

$$\Theta(n \times \sqrt{k} \times 2^{(n+k)/2}). \tag{5.1}$$

Therefore, we can calculate $r(k)$ as

$$r(k) = \Theta\left(\log(n \times \sqrt{k} \times 2^{(n+k)/2})\right) = \Theta(n),$$

for $n > k$.

Online phase

One can now use the standard method to find the herding messages. By using different elements from the set $\{(m_1, m_1^*) \times (m_2, m_2^*) \ldots \times (m_i, m_i^*) \times \ldots \times (m_\kappa, m_\kappa^*)\}$, the adversary can construct 2^κ distinct suffixes for P. The steps followed in the online stage is:

1. Using $H^f(IV, P)$ as the label of the source node of $\mathcal{D}$, find a message m^*, which acts as the labelling of the edge between the source node and one of the intermediate nodes, say ν, of $\mathcal{D}$. Let σ be the string formed by the concatenation of the labelling of the edges in the path from ν to the root of $\mathcal{D}$.
2. Compute $h_{2,1} = H(h_{1,4}, P\|m^*\|\sigma)$.
3. Find a linking message $m \in \mathcal{M}_1$ that links $h_{1,2}$ to one of the leaves of $\mathcal{D}_1$.
4. Find the path through $\mathcal{D}_1$ from the leaf node mentioned in Step 3 to the root node.

Using different elements from the set $\mathcal{M}_3$, the adversary can find 2^κ paths from the root node of $\mathcal{D}_1$ to the root node of $\mathcal{D}_2$. Each of these paths corresponds to a distinct suffix.

Analysis

We analyze the message and computational complexity of the attack separately for the two passes.

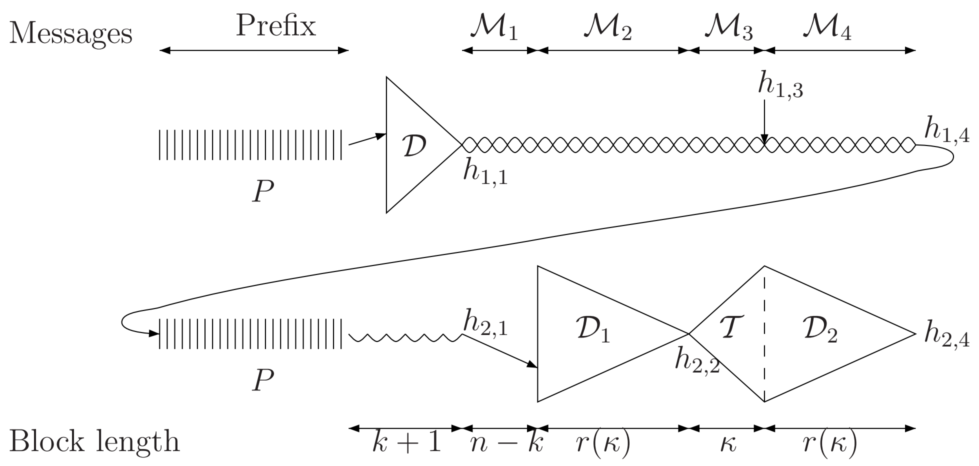


Figure 5.2: Herding attack on the hash twice construction

Message Complexity
We first calculate the message complexity of the offline phase. The first pass requires the construction of a diamond structure and four sets of multicollisions. From Theorem 4.2, the message complexity to construct the 2^k-diamond structure is $\Theta(\sqrt{k} \times 2^{(n+k)/2})$. The total number of multicollisions is

$$2^{n-k+r(k)+\kappa+r(\kappa)} = 2^{\Theta(n)}. \tag{5.2}$$

Therefore, the message complexity for the first pass is

$$O(\sqrt{k} \times 2^{(n+k)/2} + n \times 2^{n/2}).$$

The second pass requires construction of two diamond structures, $\mathcal{D}_1$ and $\mathcal{D}_2$, with k and κ as the parameter defining the size of the respective diamond structures. From equation (4.15), the message complexity for the construction of the two diamond structures is

$$O\left(n \times \sqrt{k} \times 2^{(n+k)/2} + n \times \sqrt{\kappa} \times 2^{(n+\kappa)/2}\right).$$

Since the message complexity for the construction of $\mathcal{T}$ is $O(2^\kappa)$, the message complexity of the offline phase is

$$O\left(n \times 2^{n/2} + n \times \sqrt{k} \times 2^{(n+k)/2} + n \times \sqrt{\kappa} \times 2^{(n+\kappa)/2} + 2^\kappa\right). \tag{5.3}$$

In the online phase, the adversary has to find two linking messages, m and m^*. The message complexity to find m^* is $O(2^{n-k})$. To find m, the adversary has to hash 2^{n-k} messages, each being $(n-k)$ blocks long. Hence, the message complexity of the online phase is,

$$O((n-k)\times 2^{n-k} + 2^{n-k}) = O((n-k)\times 2^{n-k}). \tag{5.4}$$

Combining equations (5.3) and (5.4), we calculate the message complexity as,

$$O\left(n \times (\sqrt{k}\times 2^{(n+k)/2} + \sqrt{\kappa}\times 2^{(n+\kappa)/2} + 2^{n-k})\right).$$

Note that under the previous assumption that $\kappa \leq k$, the message complexity of this attack is asymptotically equal to the message complexity of a single herding attack on the hash twice construction [9].

Computational Complexity
From Section 4.3.1, the computational complexity of the construction of a diamond structure based on a multicollision set is asymptotically equal to the message complexity. Therefore, the computational complexity of the offline phase is asymptotically equal to the message complexity of the offline phase. In the online phase, the adversary needs to find the linking messages and then has to find the linked node at level 0. We argue that we can safely assume that the labelling of the nodes at level 0 is sorted. If not, we sort the list, which takes a total time of $O(k \times 2^k)$ for the 2^k-diamond structure[1]. The adversary then performs a binary search to find the node. Thus, the computational complexity of finding the linking message, m and m^*, is k times more than the message complexity. Hence, the computational complexity of the attack is

$$O\left(n \times (\sqrt{k}\times 2^{(n+k)/2} + \sqrt{\kappa}\times 2^{(n+\kappa)/2} + k \times 2^{n-k})\right).$$

As in the case of the message complexity, under the assumption that $\kappa \leq k$, the computational complexity of this attack is also asymptotically equal to the computational complexity of a single herding attack on the hash twice construction [9].

5.2.2 Multiple herding attack on the generalized class of hash functions

In the previous section, we showed a 2^κ-way herding attack on the hash twice construction, which is a very special type of the GSHF. We next show an attack on a more general form of GSHF. Throughout this section, we denote the upper bound on the block length of the challenger's prefix by p. Let $H^{f,\psi}$ be a hash function based on the sequence ψ such that $|\psi| = \lambda$ and $alph(\psi) = [\ell]$. We start with the case, when $\lambda = 2\ell$ and $\psi[1,\ell] = \pi(\psi[\ell+1, 2\ell])$.

[1]As with all the analysis done in Section 4.3, the sorting time is subsumed by the computational complexity of finding the linking message.

In this case, every element from $[\ell]$ occurs exactly twice and the first half of the sequence is a permutation of the second half of the sequence.

Theorem 5.1. *Let Π_a denote the set of all possible permutations of any set of size a and ψ be a sequence of length 2ℓ with the entries from $[\ell]$. Let the challenge message be p-blocks long. For $R(\kappa) = c_1 n + c_2 \kappa$, where c_1 and c_2 are constants, the adversary can find a 2^κ-way herding attack on $\mathcal{H}^{f,\psi}$ if*

1. *$\psi[2\ell - R(\kappa), 2\ell] = \pi(\psi[\ell - R(\kappa), \ell])$ for some permutation $\pi \in \Pi_{R(\kappa)}$, and*
2. *$p + k + R(\kappa) < \ell$.*

Before giving a formal proof of the theorem, let us informally discuss the constraints in the statement of the proof. The first constraint on the structure of the sequence requires that the same block of messages is used in the last $R(\kappa)$ iterations of both the passes of the hash computation. The second constraint requires that the sequence is long enough for the attack to be successful. As we will see later, we exploit this nice structure of the sequence to find multiple herding messages.

Proof. It is easy to see that the construction in Section 5.2.1 can be used by an adversary to find 2^κ herding messages with $R(\kappa)$ equal to the total number of blocks of colliding messages in the four sets of multicollisions. This is because $\psi[2\ell - R(k), 2\ell] = \pi(\psi[\ell - R(\kappa), \ell])$ for some permutation π and in order to construct the two diamond structures, $\mathcal{D}_1$ and $\mathcal{D}_2$, the binary tree $\mathcal{T}$, and to find the linking message m, one does not necessarily require the multicollisions in the same order as in the first pass. The multicollisions only help in having a single hash value after the first pass, to provide with enough messages to construct the diamond structures and binary tree, and to find the linking message. We later show how to adapt the construction in Section 5.2.1 to construct a 2^κ-way herding attack on $\mathcal{H}^{f,\psi}$. Now $R(\kappa)$ can be upper bounded as below.

$$\begin{aligned}
R(k) &= n - k + r(k) + \kappa + r(\kappa) \\
&= n - k + \kappa + \log\Big(\sum_{i=1}^{k} 0.83 \times \sqrt{i} \times 2^{(n+i)/2}\Big) + \log\Big(\sum_{i=1}^{\kappa} 0.83 \times \sqrt{i} \times 2^{(n+i)/2}\Big) \\
&< n - k + \kappa + 2\log 0.83 + \frac{\log k + \log \kappa}{2} + \frac{n}{2} + \log\Big(\sum_{i=1}^{k} 2^{i/2}\Big) + \log\Big(\sum_{i=1}^{\kappa} 2^{i/2}\Big) + \frac{n}{2} \\
&= 2n - k + \kappa + 2\log 0.83 + \frac{\log k + \log \kappa}{2} + \frac{k + \kappa}{2} + 1.
\end{aligned}$$

As in Section 4.3, the message complexity and the computational complexity is minimum when $k \approx n/3$. Fixing $k = n/3$, and assuming that $\log k \ll k$ and $\log \kappa \ll \kappa$, we can approximate the value of $R(\kappa)$ as,

$$R(\kappa) \approx \frac{11}{6}n + \frac{3\kappa}{2}. \tag{5.5}$$

Note that for $\kappa < n$, equation (5.5) matches the asymptotic bound of equation (5.2).

We now give the details of how to adapt the construction in Section 5.2.1 to construct a 2^κ-way herding attack on $\mathcal{H}^{f,\psi}$. Let the last $R(\kappa)$ symbols of the sequence in the second pass be denoted as

$$(\alpha_1, \alpha_2, \ldots, \alpha_{n-k}, \beta_1, \beta_2, \ldots, \beta_{r(k)}, \gamma_1, \gamma_2, \ldots, \gamma_\kappa, \delta_1, \delta_2, \ldots, \delta_{r(\kappa)}) = \pi(\psi[\ell - R(\kappa), \ell]),$$

for some permutation $\pi \in \Pi_{R(\kappa)}$. It should be noted that the equality is not exact because we have an approximate estimate of $R(\kappa)$. However, for the sake of simplicity, we assume that we have an equality.

We use the same construction of Section 5.2.1 for the first pass. Let $M = M_1 \| M_2 \| \ldots \| M_\ell$ be the message from the first pass, such that the first p blocks are the challenge message, the next $(k+1)$ messages are to be found in the online phase, and the last $R(\kappa)$ blocks are the four multicollision sets constructed on the first pass. Let $\mathcal{M} = \{(m_{\ell-R(\kappa)}, m^*_{\ell-R(\kappa)}) \times \ldots \times (m_\ell, m^*_\ell)\}$ denote the set of $2^{R(\kappa)}$-multicollisions on the first pass.

1. Randomly pick 2^k hash values, and with these hash values as the nodes at level 0, construct a diamond structure $\mathcal{D}_1$ based on messages from the set $\{(m_{\beta_1}, m^*_{\beta_1}) \times (m_{\beta_2}, m^*_{\beta_2}) \ldots, \times (m_{\beta_{r(k)}}, m^*_{\beta_{r(k)}})\}$. Let the diamond structure results in the chaining value, $h_{2,2}$.

2. With $h_{2,2}$ as the label of source node, construct a kite structure as follows.

   (a) Using $h_{2,2}$ as the label of the root node, construct a complete binary tree iteratively as follows. Label the left edge of any node at level i by m_{γ_i} and the right edge by $m^*_{\gamma_i}$. The labelling of the nodes at level $(i+1)$ is done in the following manner. If the label of any node at level i is h, then the label of the left child of that node at level i is $f(h, m_{\gamma_i})$ and the label of the right child of the node at level i is $f(h, m^*_{\gamma_i})$.

   In the end, we get a complete binary tree $\mathcal{T}$, whose root is labelled by the hash value $h_{2,2}$ and the leaves are labelled by 2^κ different hash values. We denote the set of hash values at the leaf nodes by $\mathcal{H} := \{h_j : 1 \leq j \leq 2^\kappa\}$.

   (b) Using $\mathcal{H}$ as the labelling of nodes at level 0, complete the kite structure by constructing a diamond structure $\mathcal{D}_2$ based on messages from the set $\{(m_{\delta_1}, m^*_{\delta_1}) \times (m_{\delta_2}, m^*_{\delta_2}) \ldots, \times (m_{\delta_{r(\kappa)}}, m^*_{\delta_{r(\kappa)}})\}$. Let the label of the root node be $h_{2,4}$.

The adversary commits to $h_{2,4}$.

In the online phase, the adversary follows the same steps as in Section 5.2.1, except that the linking message m is chosen from the set $\{(m_{\alpha_1}, m^*_{\alpha_1}) \times (m_{\alpha_2}, m^*_{\alpha_2}) \ldots, \times (m_{\alpha_{n-k}}, m^*_{\alpha_{n-k}})\}$. By choosing different paths in $\mathcal{T}$, the adversary can find a 2^κ distinct suffixes, and hence a 2^κ-way herding attack. $\square$

We now turn our attention to a more general case when $\lambda \neq 2\ell$. For the general case, we restrict the freedom given to the challenger. We assume that

1. The adversary knows the upper bound on the block length of challenger's prefix in advance.

2. The adversary is allowed a permutation of the sequence on which the hash function is based.

Note that the two assumptions are not unrealistic. If we do not allow the first assumption to the adversary, then the challenger can provide arbitrary long challenge, and the adversary does not have enough number of message blocks to construct the required data structure. For the second assumption, if we do not allow the adversary the freedom to permute the sequence, the challenger can always win the game. The challenger just uses a sequence in which the last symbol of the sequence is the first element of the set on which the sequence is based. This will reduce the problem of finding the herding message to the problem of finding the second preimage. Note that the attack of Andreeva *et al* [2] on the Zipper hash also gave this freedom to the adversary, where the permuation is of a very special form.

For the rest of this section, let $m := \max\{R(\kappa), p + k + 1\}$, where k is the parameter of the diamond structure and $R(\kappa)$ is the number of message blocks required to construct the desired number of multicollsions.

Theorem 5.2. *Let Π_ℓ denote the set of all possible permutations of any set of size ℓ and ψ be a sequence of length λ, where $alph(\psi) = [\ell]$. Let the challenge message be at most p-blocks long. Then, for all $\ell \geq 64 \times m^2$, we can find a 2^κ-way herding attack on $\mathcal{H}^{f,\psi}$.*

Before we give a formal proof of the Theorem 5.2, we give an argument for the bound on l. For $l \geq 64 \times m^2$, Lemma 3.4 guarantees that there is a set of $8m$ symbols that occurs at most twice in the whole sequence such that if we restrict the sequence to these symbols, the resulting sequence is in the form of $t \leq 2$ successive permutations. This is because in the proof of the Lemma 3.4, Shamir and Hoch derived that

$$\ell \geq C_q \times x^{D_q},$$

where C_q and D_q are defined recursively as follows:

$$C_q = ((q-1) \times C_{q-1})^{D_{q-1}+1}, \ C_1 = x,$$

and

$$D_q = D_{q-1}^2 + 1, D_1 = x.$$

For $x = 8m$ and $q = 2$, we have $\ell \geq x^2 = 64 \times m^2$.

We construct our data structure in a form similar to the construction of Section 5.2.1 with minor changes on the blocks in the two successive permutations. We follow up with the details:

Proof. For notational simplicity, let $\mathcal{X} = alph(\psi)$. By Lemma 3.4, there exists a subset $\mathcal{Y} \subseteq \mathcal{X}$ that occurs at most twice in the whole sequence, such that if ψ is restricted to the symbols from $\mathcal{Y}$, then the resulting sequence is in the form of at most two successive permutations. Let t denote the number of times any element of $\mathcal{Y}$ occurs in the sequence.

If $t = 1$, the attack reduces to the attack of Kelsey and Kohno. The reason is that the adversary is allowed a permutation of ψ. The adversary follows the following steps. First, she finds the set of symbols $\mathcal{Y} \subseteq \mathcal{X}$ such that they occur just once in the whole sequence. Let the symbols be $(\alpha_1, \alpha_2, \ldots, \alpha_{8m})$. The adversary uses these symbols as the index of the message blocks of prefix, the linking message, and the messages used to construct the diamond structure in the proper order[2]. She also fixes the messages indexed by symbols not in $\mathcal{Y}$ by some fixed value, IV. Therefore, without loss of generality, we can assume that $t = 2$.

By Lemma 3.4, there exists a set of symbols $\mathcal{Y} \subseteq \mathcal{X}$, such that $|\mathcal{Y}| = 8m$ and $\psi|_{\mathcal{Y}}$ is in the form of two successive permutations. Let $\psi = \psi_1 \| \psi_2$ be such that $\psi_1|_{\mathcal{Y}}$ and $\psi_2|_{\mathcal{Y}}$ are permutations of each other. We further break $\psi_1 = \alpha_1 \| \beta_1$ and $\psi_2 = \alpha_2 \| \beta_2$ in two parts, such that $\alpha_1|_{\mathcal{Y}}, \alpha_2|_{\mathcal{Y}}, \beta_1|_{\mathcal{Y}},$ and $\beta_2|_{\mathcal{Y}}$ have the same cardinality. Let

$$\mathcal{B}_1 = \alpha_1|_{\mathcal{Y}}, \ \mathcal{B}_2 = \alpha_2|_{\mathcal{Y}},$$

$$\mathcal{C}_1 = \beta_1|_{\mathcal{Y}}, \text{ and } \mathcal{C}_2 = \beta_2|_{\mathcal{Y}}.$$

Note that $\psi_1|_{\mathcal{Y}}$ and $\psi_2|_{\mathcal{Y}}$ are two permutations of $\mathcal{Y}$. From Lemma 3.6, we have either of the two cases:

$$|\mathcal{B}_1 \cap \mathcal{B}_2| \geq m \text{ and } |\mathcal{C}_1 \cap \mathcal{C}_2| \geq m, \text{ or}$$

$$|\mathcal{B}_1 \cap \mathcal{C}_2| \geq m \text{ and } |\mathcal{B}_2 \cap \mathcal{C}_1| \geq m.$$

[2]The adversary uses a permutation to get the right order of indexing.

We analyse both the cases. For both the cases, the adversary permutes the sequence in such a way so that the symbols in the part of sequence that contains $\mathcal{D}_1$ contains the entire challenger's prefix.

We first describe the offline phase for the two cases separately.

1. Case 1: $|\mathcal{B}_1 \cap \mathcal{B}_2| \geq m$ and $|\mathcal{C}_1 \cap \mathcal{C}_2| \geq m$.

   We build the following structure for the attack:

   (a) Fix $M_i = IV$ for all $i \in \mathcal{X} \backslash (\mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{C}_1 \cup \mathcal{C}_2)$.

   (b) Build a simple diamond structure, $\mathcal{D}_1$ on the last k symbols of $\mathcal{B}_1$.

   (c) Construct a set of 2^m-multicollisions, say $\mathcal{M}$, for the messages indexed by the symbols in $\mathcal{C}_1$ and common to $\mathcal{C}_2$. The adversary uses $\mathcal{M}$ to find the linking message and to construct a diamond and a kite structure in $\mathcal{C}_2$. We can do this because of the argument used in the proof of Theorem 5.1.

   Figure 5.3 shows the description of the attack for case 1.

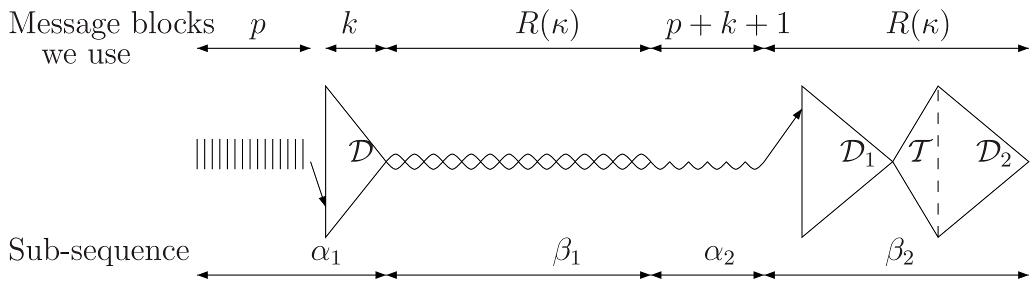


Figure 5.3: Herding attack for Case 1

2. Case 2: $|\mathcal{B}_1 \cap \mathcal{C}_2| \geq m$ and $|\mathcal{B}_2 \cap \mathcal{C}_1| \geq m$. We build the following structure for the attack:

   (a) Fix $M_i = IV$ for all $i \in \mathcal{X} \backslash (\mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{C}_1 \cup \mathcal{C}_2)$.

   (b) Construct a set of 2^m-multicollisions, say $\mathcal{M}$, for the messages indexed by the symbols in $\mathcal{B}_1$ and common to $\mathcal{C}_2$. The adversary uses $\mathcal{M}$ to find the linking message and to construct a diamond structure and a kite structure in $\mathcal{C}_2$. We can do this by the arguments in the proof of Theorem 5.1.

(c) Build a simple diamond structure, $\mathcal{D}_1$ on the last k alphabets of $\mathcal{B}_2$

Figure 5.4 shows the description of the attack for case 2.

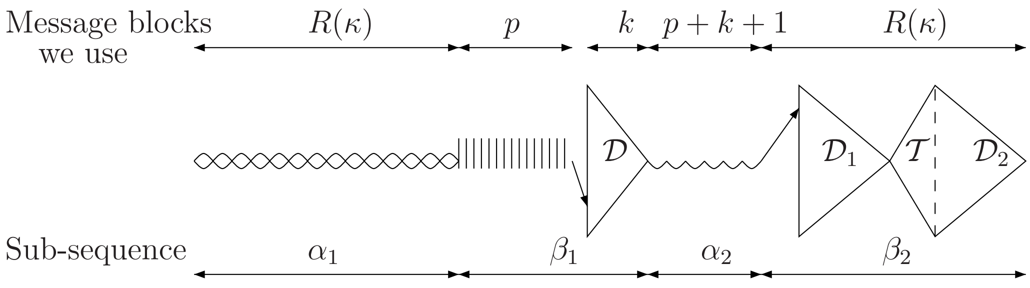


Figure 5.4: Herding attack for Case 2

In the online phase, the challenger gives a prefix P, such that the block-length of P is less than p. The adversary uses the blocks of P corresponding to the first p symbols in $\mathcal{B}$, where $\mathcal{B}$ is either $\mathcal{B}_1$ or $\mathcal{B}_2$, depending on the case listed above. Then the adversary finds a linking message that links the hash value to one of the leaves of $\mathcal{D}$. The rest of the steps are same as in the proof of Theorem 5.1. □

Finally, we need to describe how an adversary can efficiently find the subset $\mathcal{Y}$. It is easy to find the subset $\mathcal{Y}$ when $t = 1$. So, we will consider the case when $t = 2$. Let $(\psi, \prec)$ be a partial order such that $a \prec b$ holds if the last occurrence of a is before the first occurrence of b in the sequence ψ. Let $\mathcal{Y}$ be the subset of $\mathcal{X}$ such that $\psi|_{\mathcal{Y}}$ is in the form of two successive permutations. It is trivial to see that if a sequence reduced by a set is in the form of two successive permutations, then every element in the set are incomparable with respect to the partial order $\prec$. Therefore, $\mathcal{Y}$ forms an antichain with respect to the partial order $\prec$. By the equivalence of König's Theorem and Dilworth's theorem, we know that finding an antichain in a sequence is equivalent to finding a vertex cover in the bipartite graph[3] constructed in König's theorem. We next give a brief description of the method. For more details, we refer the reader to the lecture notes of Goemans [29].

[3]For a graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$, if we can partition the vertex set into two disjoint set $\mathcal{V}_1$ and $\mathcal{V}_2$, such that for every edge $(u, v) \in \mathcal{E}$, it is the case that either $u \in \mathcal{V}_1$ and $v \in \mathcal{V}_2$, or $u \in \mathcal{V}_2$ and $v \in \mathcal{V}_1$, then such graphs are called *bipartite graphs*.

Finding anti-chain of partially ordered set $(\psi, \prec)$

Let $(\psi, \prec)$ be a partially ordered set. We define a bipartite graph $\mathcal{G}$ with two vertices a_i, b_i for every $i \in \psi$, and the edge set $\mathcal{E}(\mathcal{G})$ is such that $(a_i, b_j) \in \mathcal{E}(\mathcal{G}) \Leftrightarrow i \prec j$. From the equivalence of the Dilworth's theorem and König's theorem, the minimum vertex cover of $\mathcal{G}$ is an antichain in ψ. Now finding a vertex cover in a bipartite graph is easy by using the construction in the König's theorem.

We start with the Edmonds-Gallai decomposition of $\mathcal{G}$ [29]. This decomposition can be found by using the Edmonds algorithm [21] to find the maximum matching of a graph. The decomposition partitions $\mathcal{V}(\mathcal{G})$ in three disjoint sets: $D(\mathcal{G})$ is the set of all vertices v such that there is some maximum matching that leaves v uncovered, $A(\mathcal{G})$ is the neighbor set of $D(\mathcal{G})$, and $C(\mathcal{G})$ is the set of all remaining vertices. Since we started with Edmonds-Gallai decomposition, every component, H, of the subgraph induced by $D(\mathcal{G})$ is factor critical, *i.e.*, the subgraph induced by removing any vertex u from H has a perfect matching. Also, $C(\mathcal{G})$ has a perfect matching. Using these facts, we find a vertex cover as follows:

1. Take all vertices of $A(\mathcal{G})$.
2. Since $C(\mathcal{G})$ is bipartite, take any one side of $C(\mathcal{G})$ in the vertex cover.
3. From each component of $D(\mathcal{G})$, delete one of the vertices which is adjacent to a vertex of $A(\mathcal{G})$. Then we take half of the elements from each component of $D(\mathcal{G})$.

An anti-chain is simply the elements of the sequence corresponding to the vertices in the vertex cover found in the above steps.

For the computational complexity of finding the antichain, we observe that it is exactly the same as the computational complexity of finding the maximum matching. We use Micali-Vazirani's algorithm [55]. We can also assume that the adversary perform this as a pre-computation. Finding a maximum matching on a graph with $\mathcal{V}$ vertices and $\mathcal{E}$ edges can be done in $O(|\mathcal{V}|\sqrt{|E|})$. Therefore, this does not have any impact on the overall computational complexity of the attack. Therefore, the message complexity and the computational complexity is the same as that in the attack of Theorem 5.1.

5.3 Conclusion

In the last four years, the diamond structure has been used as one of the important data structures for generic attacks on specific constructions of hash functions. Andreeva *et al.* [2] showed that it is easy to herd many other hash constructions, such as the hash twice and concatenated hash, which seems to be resistant to the then known generic attacks.

In this chapter, we showed that their attack can be generalized to a very wide variety of hash functions. We showed that the generalization of iterated hash functions defined by Nandi and Stinson [59] are also susceptible to the herding attack. We proved a bound on the size of symbol set that makes any hash function based on that symbol set susceptible to our attack. However, we are unable to translate the attack to the tree-based construction, which is possible for Nandi and Stinson's multicollisions and for Hoch and Shamir's multicollisions.

Chapter 6

Conclusion and future work

In this thesis, we computed the complexity of the construction of the diamond structure and the attacks based on it. We revisited the construction of Kelsey and Kohno and pointed out a subtle flaw in it (Chapter 4). We corrected their analysis by using concepts from the theory of random graphs, showing that the message complexity is asymototically $\sqrt{k}$ times the estimate by Kelsey and Kohno, where k is the parameter defining the size of the diamond structure. We also calculated the computational complexity of the construction, thereby pointing out the need of computing the computational complexity in all the attacks if the attacks are to be of any practical importance. We use this revised analysis in the analysis of our herding attack on a generalized class of hash functions. For the generalized class of hash functions, we gave a weaker version of the herding attack, in which we allow adversary with more power than the herding attack defined in the literature.

In summary, this thesis provides the following contributions:

1. Corrects the analysis of the diamond structure by giving a more concrete and detailed analysis (Chapter 4).

2. Advocates the need of computing the computational complexity of any generic attack (Chapter 4).

3. Extends the idea of Andreeva *et al.* [2] to show that finding multiple herding attacks on a generalized class of hash functions is not difficult under some weaker notion of CTFP resistance (Chapter 5).

4. Proves the lower bound on the number of message blocks for the weaker version of the herding attack (Chapter 5).

This work also leaves some open problems which might be of independent interest. We enumerate a few of them as follows:

1. In this work, we proposed a herding attack on a generalized class of hash functions in which every message block can be used at most twice. One of the future directions of work can be to find the feasibility of the herding attack on the family of hash functions in which every message can be used some fixed number of times. An approach can be to use Lemma 3.4 and Lemma 3.6 for the case when the message is used at most q times and then somehow use our method of case analysis.

2. There are several combining rules which are not covered in this work. Nandi [57] and Hirose [35] independently defined a family of such combining rule. In their construction, one uses the complete message twice, the second time as some permutation of the bits of the original message. However, this family of hash functions differs from the Definition 2.3.2 in a way that the permutation is done on the message bits and not on the message blocks. To our knowledge, no generic attack has been proposed on such constructions.

3. Gauravaram [25] introduced a new combining rule for hash functions, which he called *iterative halving*. In this scheme, the compression function is usually a block cipher, followed by truncation of the second half of the encrypted message. The user performs this iteration until the final encrypted message has required bit size. There has been no generic attack defined on such a construction. At first glance, this construction seems to be resistant even to Joux's attack.

4. It is an interesting problem whether we can have a tighter bound (in terms of constant) than the one achieved in the Theorem 5.2 for a generalized class of hash functions. The bound is asymptotically tight if we assume that $\Omega(m^2)$ is the lower bound on the required number of multicollisions [36, 59]. This is because finding a herding attack on iterated hash functions is at least as hard as finding a collision for iterated hash functions [43].

Bibliography

[1] J. P. Allouche and J. Shallit. *Automatic Sequences: Theory, Applications, Generalizations.* Cambridge University Press, 2003. 13, 24

[2] E. Andreeva, C. Bouillaguet, O. Dunkelman, and J. Kelsey. Herding, second preimage and Trojan message attacks beyond Merkle-Damgård. In *Selected Areas in Cryptography*, pages 393–414, 2009. 9, 10, 33, 36, 37, 39, 41, 51, 52, 53, 58, 59, 61, 63, 70, 74, 76, 84

[3] E. Andreeva, C. Bouillaguet, P. A. Fouque, J. J. Hoch, J. Kelsey, A. Shamir, and S. Zimmer. Second preimage attacks on dithered hash functions. In *EUROCRYPT*, pages 270–288, 2008. 9, 33, 35, 41, 54, 55, 57

[4] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *CRYPTO*, pages 1–15, 1996. 6

[5] M. Bellare and P. Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In *CRYPTO*, pages 470–484, 1997. 6

[6] E. Biham and R. Chen. Near-collisions of SHA-0. In *CRYPTO*, pages 290–305, 2004. 5

[7] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby. Collisions of SHA-0 and reduced SHA-1. In *EUROCRYPT*, pages 36–57, 2005. 6

[8] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. In *CRYPTO*, pages 2–21, 1990. 6

[9] S. R. Blackburn, D. R. Stinson, and J. Upadhyay. On the complexity of the herding attack and some related attacks on hash functions. Cryptology ePrint Archive, Report 2010/030, 2010. `http://eprint.iacr.org/`. 35, 36, 39, 47, 50, 67, 86

[10] B. Bollobas. *Random Graphs.* John Wiley and Sons Incorporation, Hoboken, New Jersey, U.S.A., 2008. 43, 44

[11] J. A. Bondy and U. S. R. Murty. *Graph Theory.* Springer, 2008. 15

[12] R. Canetti, R. L. Rivest, M. Sudan, L. Trevisan, S. P. Vadhan, and H. Wee. Amplifying collision resistance: A complexity-theoretic treatment. In *CRYPTO*, pages 264–283, 2007. 9

[13] F. Chabaud and A. Joux. Differential collisions in SHA-0. In *CRYPTO*, pages 56–71, 1998. 6

[14] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990. 13

[15] I. Damgård. Collision free hash functions and public key signature schemes. In *EUROCRYPT*, pages 203–216, 1987. 6

[16] I. Damgård. A design principle for hash functions. In *CRYPTO*, pages 416–427, 1989. 21

[17] R. D. Dean. *Formal Aspects of Mobile Code Security.* Ph.D Dissertation, Princeton University, Princeton, U.S.A, 1999. 5

[18] R. Diestel. *Graph Theory.* Springer Verlag, Heidelberg, 2005. 15

[19] W. Diffie and M. Hellman. New direction in cryptography. *IEEE Transactions on Information Theory*, IT-22(6). 1

[20] C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. *J. Cryptology*, 11(3):187–208, 1998. 6

[21] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. 74

[22] P. Erdös and A. Renyi. On the evolution of random graphs. In *Proceedings of the Hungarian Academy of Sciences*, volume 5, pages 17–61, 1960. 45

[23] P. Erdös and A. Rényi. On the existence of a factor of degree one of a connected random graph. *Acta Mathematica Academiae Scientiarum Hungaricae Tomus*, 17(3–4):359–368, 1966. 45

[24] M. Fischlin and A. Lehmann. Security-amplifying combiners for collision-resistant hash functions. In *CRYPTO*, pages 224–243, 2007. 9

[25] P. Gauravaram. *Cryptographic Hash Functions: Cryptanalysis, Design and Applications.* Ph.D Dissertation, Queensland University of Technology, 2007. 77

[26] P. Gauravaram and J. Kelsey. Linear-XOR and additive checksums don't protect Damgård-Merkle hashes from generic attacks. In *CT-RSA*, pages 36–51, 2008. 35

[27] P. Gauravaram and L. R. Knudsen. On randomizing hash functions to strengthen the security of digital signatures. In *EUROCRYPT*, pages 88–105, 2009. 6

[28] P. Gauravaram, W. Millan, E. Dawson, and K. Viswanathan. Constructing secure hash functions by enhancing Merkle-Damgård construction. In *ACISP*, pages 407–420, 2006. 35

[29] M. Goemans. Lecture notes on Topics in Combinatorial Optimization, 2004. `http://www-math.mit.edu/~goemans/18997-CO/topics-co.html`. 73, 74

[30] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. 6

[31] J. Guo, S. Ling, C. Rechberger, and H. Wang. Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. Cryptology ePrint Archive, Report 2010/016, 2010. `http://eprint.iacr.org/`. 6

[32] S. Haber and W. S. Stornetta. How to time-stamp a digital document. *J. Cryptology*, 3(2):99–111, 1991. 6

[33] S. Halevi and H. Krawczyk. Strengthening digital signatures via randomized hashing. In *CRYPTO*, pages 41–59, 2006. 6, 35

[34] K. Halunen, J. Kortelainen, and T. Kortelainen. Multicollision attacks and generalized iterated hash functions. Manuscript, 2010. 30, 31, 32

[35] S. Hirose. Provably secure double-block-length hash functions in a black-box model. In *ICISC*, pages 330–342, 2004. 77

[36] J. J. Hoch and A. Shamir. Breaking the ICE - finding multicollisions in iterated concatenated and expanded (ICE) hash functions. In *FSE*, pages 179–194, 2006. 10, 11, 30, 32, 77

[37] J. J. Hoch and A. Shamir. On the strength of the concatenated hash combiner when all the hash functions are weak. In *ICALP (2)*, pages 616–630, 2008. 9

[38] S Hohenberger and B. Waters. Realizing hash-and-sign signatures under standard assumptions. In *EUROCRYPT*, pages 333–350, 2009. 6

[39] A. Joux. Multicollisions in iterated hash functions. Application to cascaded constructions. In *CRYPTO*, pages 306–316, 2004. 4, 6, 9, 28

[40] A. Joux and T. Peyrin. Hash functions and the (amplified) Boomerang attack. In *CRYPTO*, pages 244–263, 2007. 6

[41] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987. 1

[42] J. Katz and Y. Lindell. *Introduction to Modern Cryptography.* Chapman and Hall, CRC Press, 2007. 17

[43] J. Kelsey and T. Kohno. Herding hash functions and the Nostradamus attack. In *EUROCRYPT*, pages 183–200, 2006. 3, 9, 19, 39, 41, 42, 43, 45, 60, 77

[44] J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *EUROCRYPT*, pages 474–490, 2005. 4, 9, 33, 54

[45] V. Keränen. Abelian squares are avoidable on 4 letters. In *ICALP*, pages 41–52, 1992. 24, 55

[46] V. Keränen. On abelian square-free DT0L-languages over 4 letters. In *Proceedings of Conference on Combinatorics on Words*, pages 41–52, 2003. 24, 55

[47] V. Klima. Huge multicollisions and multipreimages of hash functions BLENDER-n. Cryptology ePrint Archive, Report 2009/006, 2009. `http://eprint.iacr.org/`. 6

[48] D. E. Knuth. *The Art of Computer Programming Volumes 1-3 Boxed Set.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998. 1

[49] E. Lee, J. Kim, D. Hong, C. Lee, J. Sung, S. Hong, and J. Lim. Weak-key classes of 7-round MISTY 1 and 2 for related-key amplified Boomerang attacks. *IEICE Transactions*, 91-A(2):642–649, 2008. 6

[50] M. Liskov. Constructing an ideal hash function from weak ideal compression functions. In *Selected Areas in Cryptography*, pages 358–375, 2006. 9, 29, 41

[51] U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *TCC*, pages 21–39, 2004. 9

[52] U. M. Maurer and S. Tessaro. Domain extension of public random functions: Beyond the birthday barrier. In *CRYPTO*, pages 187–204, 2007. 41

[53] A. J. Menezes, S. A. Vanstone, and P. C. van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1996. 17

[54] R. Merkle. One way hash functions and DES. In *CRYPTO*, pages 428–446, 1989. 21

[55] S. Micali and V. V. Vazirani. An $O(m\sqrt{n})$ algorithm for finding maximum matching in general graphs. In *FOCS*, pages 17–27, 1980. 49, 74

[56] R. Motwani. Average-case analysis of algorithms for matchings and related problems. *J. ACM*, 41(6):1329–1356, 1994. 49

[57] M. Nandi. *Design and Cryptanalysis of Iterated Hash Functions.* I.S.I. Kolkata, 2006. 77

[58] M. Nandi. Characterizing padding rules of MD hash functions preserving collision security. In *ACISP*, pages 171–184, 2009. 21

[59] M. Nandi and D. Stinson. Multicollision attacks on some generalized sequential hash functions. *IEEE Transactions on Information Theory*, 53(2):759–767, 2007. 10, 11, 29, 30, 61, 75, 77

[60] K. Nishimura and M. Sibuya. Probability to meet in the middle. *J. Cryptology*, 2(1):13–22, 1990. 5

[61] A. Numayama and K. Tanaka. On the weak ideal compression functions. In *ACISP*, pages 232–248, 2009. 9

[62] J. S. Park, M.S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. *SIGMOD Rec.*, 24(2):175–186, 1995. 1

[63] B. Preneel. *Analysis and Design of Cryptographic Hash Function.* Ph.D Dissertation, Katholieke Universiteit Leuven, 1993. 6, 18

[64] R. Rivest. Abelian square-free dithering for iterated hash functions. 2005. 13, 24, 54, 55

[65] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *FSE*, pages 371–388, 2004. 19

[66] Phillip Rogaway. Formalizing human ignorance. In *VIETCRYPT*, pages 211–228, 2006. 18

[67] B. Schneier. *Applied Cryptography.* John Wiley and Sons, 1996. 29

[68] J. Shallit. Personal communication, 2010. 30

[69] A. Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984. 8

[70] W. Stallings. *Cryptography and Network Security.* Prentice Hall, 2006. 17

[71] R. P. Stanley. *Enumerative combinatorics*. Wadsworth Publ. Co., Belmont, CA, USA, 1986. 14

[72] D. R. Stinson. *Cryptography: Theory and Practice.* Chapman & Hall/CRC Press Inc., 2006. 17

[73] K. Suzuki, D. Tonien, K. Kurosawa, and K. Toyota. Birthday paradox for multi-collisions. *IEICE Transactions*, 91-A(1):39–45, 2008. 28

[74] A. Thue. Über die gegenseitige lage gleicher teile gewisser zeichenreihen. *Norske vid. Selsk. Skr. Mat. Nat. Kl*, 1:1–67, 1912. 24

[75] David Wagner. The boomerang attack. In *FSE*, pages 156–170, 1999. 6

[76] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In *EUROCRYPT*, pages 1–18, 2005. 5

[77] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *CRYPTO*, pages 17–36, 2005. 5, 6

[78] X. Wang and H. Yu. How to break MD5 and other hash functions. In *EUROCRYPT*, pages 19–35, 2005. 5

[79] X. Wang, H. Yu, and Y. L. Yin. Efficient collision search attacks on SHA-0. In *CRYPTO*, pages 1–16, 2005. 5, 6

[80] M. N. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981. 6

[81] H. Yu, G. Wang, G. Zhang, and X. Wang. The second-preimage attack on MD4. In *CANS*, pages 1–12, 2005. 5

[82] H. Yu and X. Wang. Multi-collision attack on the compression functions of MD4 and 3-pass HAVAL. In *ICISC*, pages 206–226, 2007. 6

Appendix A

Multiple second preimage on hash twice construction

In this appendix, we propose a 2^{κ}-way second preimage attack on the hash twice construction, based on the compression function f. Let the challenge message be $M = M_1 \| M_2 \| \dots M_{2^{\ell}}$. On a high level, the attack is based on the second preimage attack by Andreeva *et al.* on the hash twice construction [2]. In addition to the construction proposed by Andreeva *et al.*, we append a kite structure. We next give a more detailed construction.

Offline phase

The steps followed in the construction of the data structure in the first pass follows the same steps as in the first pass as described in Section 5.2.1, except that instead of constructing $\mathcal{D}$, the adversary constructs a $(\ell, 2^{\ell} + \ell - 1)$-expandable message. By the end of the construction, we have $h_{1,4}$ as the chaining value of the first pass, and $h_{2,4}$ as chaining value of the second pass.

Online phase

We can now use the standard method to find second preimage attack. By using different elements from the set $\{(m_1, m_1^*) \times (m_2, m_2^*) \dots \times (m_i, m_i^*) \times \dots \times (m_{\kappa}, m_{\kappa}^*)\}$, we can construct 2^{κ}-distinct second preimage for M. The steps followed in the online stage are:

1. Find a linking message, m^*, that links $h_{2,4}$ to one of the intermediate hash value, h_i, in the second pass of M.

2. Compute $h_{1,c} = H(h_{1,4}, m \| M_{i+1} \| \ldots \| M_{2^l})$.
3. Compute $j = i - (n - k + r(k) + \kappa + r(\kappa))$
4. Find the message M_{expand} from the set of expandable message set, which is j-block long.
5. With $h_{1,c}$ as initial value, compute $h_{2,1} = H^f(h_{1,c}, M_{expand})$.
6. Find a linking message $m \in \mathcal{M}_1$ that links $h_{1,2}$ to one of the node at level 0 of $\mathcal{D}_1$.

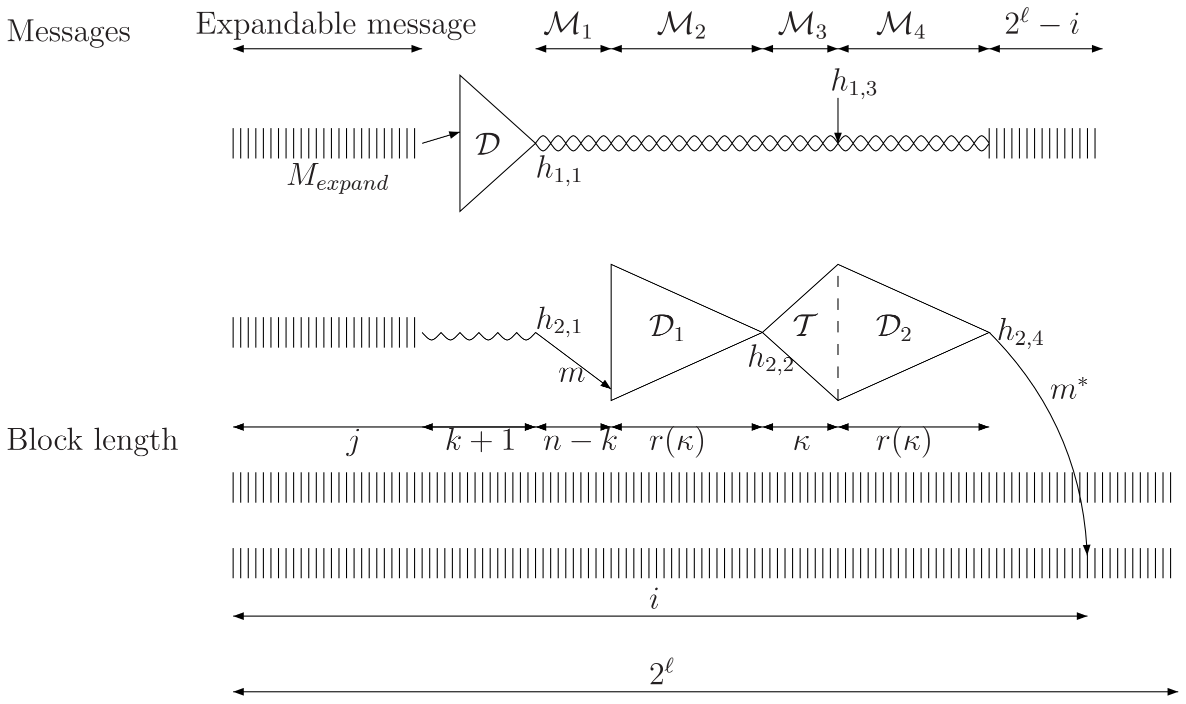


Figure A.1: Multiple Second preimage on the hash twice construction

Analysis

We analyze the message and the computational complexity of the attack for the two passes separately.

Message Complexity
We first calculate the message complexity of the offline phase. The first pass requires a construction of an $(\ell, \ell+2^\ell-1)$-expandable message. The message complexity to construct the $(\ell, \ell+2^\ell-1)$-expandable message is $O(\ell \times 2^{n/2})$. The rest of the analysis for the offline phase follows the same line as in Section 5.2.1. Therefore, the message complexity of the offline phase is

$$O\left(n \times 2^{n/2} + n \times \sqrt{k} \times 2^{(n+k)/2} + n \times \sqrt{\kappa} \times 2^{(n+\kappa)/2} + 2^\kappa\right) \tag{A.1}$$

In the online phase, the adversary needs to find two linking messages, m and m^*. The message complexity to find m^* is $O(2^{n-\ell})$. In case of m, there are 2^{n-k} messages to be hashed, each having $(n-k)$ blocks. Hence, the message complexity in the online phase is,

$$O((n-k) \times 2^{n-k} + 2^{n-l}) = O((n-k) \times 2^{n-k} + 2^{n-\ell}) \tag{A.2}$$

Combining equation (A.1) and (A.2), we calculate the message complexity as

$$O\left(n \times (\sqrt{k} \times 2^{(n+k)/2} + \sqrt{\kappa} \times 2^{(n+\kappa)/2} + 2^{n-k}) + 2^{n-\ell}\right),$$

which is the same as a single preimage attack on hash twice construction [9], under the assumption that $\kappa \leq k$.

Computational Complexity
Note that the computational complexity of the construction of diamond structure based on a set of multicollisions is the same as message complexity. Therefore, the computational complexity of the offline phase is the same as the message complexity of the offline phase. In the online phase, the adversary needs to find the linking messages m^* (and m respectively) and then has to actually find the node (and the chaining value respectively). As in Section 4.3, we can assume that the chaining values and the labelling of the nodes at level-0 of $\mathcal{D}$ and $\mathcal{D}_1$ is sorted. Therefore, the computational complexity of finding the linking message m^* (and m respectively) increases by a factor of ℓ (and k respectively). We thus calculate the computational complexity of the attack as,

$$O\left(n \times (\sqrt{k} \times 2^{(n+k)/2} + \sqrt{\kappa} \times 2^{(n+\kappa)/2} + k \times 2^{n-k}) + \ell \times 2^{n-\ell}\right),$$

which is same as a single preimage attack on hash twice construction [9], under the assumption that $k = \kappa$.