Design of Iteration on Hash Functions and its Cryptanalysis

MRIDUL NANDI

Applied Statistics Unit
Indian Statistical Institute
Kolkata - 700108
India.

Design of Iteration on Hash Functions and its Cryptanalysis

Mridul Nandi

A thesis submitted to the *Indian Statistical Institute*
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
May 2005

under the supervision of

Professor Bimal Roy
Applied Statistics Unit
Indian Statistical Institute
Kolkata - 700108
India.

Dedicated to my parents

Preface

Cryptology consists of two complementary fields of research. One is cryptography, where the development of new schemes or algorithms are concerned and the other one is cryptanalysis which provides an attack algorithm to break the cryptographic algorithm. In the modern age, cryptology is a highly studied area in computer science.

The cryptographic hash function is popular in many digital applications, like digital signature, digital time-stamping, message authentication, PKI or public key infrastructure etc. The main reasons of having huge applications of hash function is that an output of a hash function can be considered as "compact", "unambiguous", "one-way" or random transformation of the message. So proper design of hash function is an important research area.

Till now there are many proposed hash functions. Most of them are constructed from scratch and rest of them are designed by using some other primitives of cryptography, like Block Ciphers, Stream Ciphers etc. Many researches have been made in designing hash functions or making cryptanalysis on hash functions. Unfortunately, most of the proposed hash functions are not secure, that is, there exists cryptanalysis on these hash functions. One of the major component of designing a hash function is domain extension of hash functions. The classical approach of extending hash functions is being scrutinized by many people. There are many drawbacks in the classical approach. Nowadays many researches to extend the range of hash function also have been made.

In this thesis, we study the both problems stated above, the domain extension and the range extension of hash function. We also study some type of cryptanalysis, mainly multicollision attack on hash function. Thus we provide many methods of extending domain and study the standard security properties. At the same time we also provide some attacks on some popularly designed hash functions. The author hopes that the research in this thesis would be helpful to advance the study in this significant field.

May, 2005
Mridul Nandi

Acknowledgement

The thesis would not have been possible without the help of many people, whom I would like to acknowledge here. First of all, I wish to express my sincere appreciation to my Professor Bimal Roy for supervising the thesis. I am deeply grateful to him for his heartily and earnest guidance which has encouraged me all the time. I am really greatly indebted to him for all academic and non-academic supports that I have received from him throughout my research period. I am also happy and impressed to meet such a great mind and personality of him.

I am profoundly grateful to Professor Douglas R. Stinson of Waterloo University. While I stayed with him in Waterloo, he made a great change in my way of intuition, in thinking process and also in writing skill. I am also grateful to Dr. Palash Sarkar, Professor S.B.Rao, ex-Director of Indian Statistical Institute, Professor Rana Barua and Dr. Subhamoy Maitra of our Institute who have continuously provided valuable comments and suggestions. I got lot of helps from Professor Sangjin Lee of Korea University, Professor Alfred Menezes of Waterloo University and Professor Kouichi Sakurai of Kyushu University. I got lot of academic as well as non-academic helps from my close friends Dr. Wonil Lee of Korea University.

I would like to thank my close friends Donghoon Chang, Souradyuti Paul and Jongsung Kim. I would also like to thank to my close friends of my Department Avishek, Debrup, Dalai, Ipsita, Kishan, Madhu, Pradeep, Sumanta, Saurav, Sanjit, Tanmay who helped me in many ways. I would also like to thank Ratna and Sanjeev who had spent their valuable time in reading my thesis. We had great discussions with Samrat, Tathagata, Arnab, Neo and Malapati. I had a great time with my neighbor friends Archan, Babu, Bappa, Debu, Raju, Rupal, Samrat, Sajal, Tapa and other close friends Dwaipayan, Debasis, Joy, Jumelia, Nandita, Omesh, Pratyay, Paromita, Penti and Sayan. I had a memorable time in our institute with my junior friends Brotish, Bantu, Bijit, Mithun, Riten, Seru, Som, Tuhin and many more. I am also grateful to all other teaching and non-teaching people in my department who have provided all sort of technical supports.

At last, but not the least, I would like to express my heartiest gratitude to my family members for their heartfelt cooperation and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A *hash function* $H(\cdot)$ is an easily computable function from the set $\{0,1\}^*$ of all finite length binary strings to a set $\{0,1\}^n$ of all binary strings of length $n > 0$. One can encode an English text or a message to a finite binary string and hence the set of all messages can be identified with $\{0,1\}^*$. Nowadays hash functions have been used in many practical applications, for example, digital signature [7, 20, 27, 34, 74], digital timestamp [4, 35], message authentication code (or MAC) [51, 109], public key encryption [17, 101] etc. The reason for applying hash function is that an output of a hash function can be considered as "compact", "unambiguous", "one-way" or random transformation of the message. For a message M, the hash-output, $H(M)$ can be used to identify M uniquely. This explains the term "unambiguous transformation". Apart from the unambiguity, the hash output should look like random. It is undesirable that, from a hash output of a message one can predict the message. This explains the term "one-way". The term "compact" refers the fact that for any finite length message, the output size of a hash function is fixed. So there is no need to take care about the variable length messages for designing other primitives since one can use the hash function as a preprocessor. While one designs a hash function several security requirements are kept in mind. These are *collision resistance*, *multicollision resistance*, *preimage resistance*, *target collision resistance* and 2^{nd} *preimage resistance*. Informal definitions of these properties for a keyed family of hash functions $\{H_k\}_{k\in\mathcal{K}}$ are given below. A detailed description of security properties are provided in Sect. 2.3.2.

1. ***Collision resistance* :** Given a random key k from key space $\mathcal{K}$, it should not be "easy to find" $M_1 \neq M_2$ such that $H_k(M_1) = H_k(M_2)$. The above event is known as *collision* and the pair of the messages (M_1, M_2) is known as a *collision*

pair. The term "easy to find" roughly means a polynomial time algorithm.

2. ***Multicollision resistance :*** This is a generalized notion of collision resistance. A hash family is said to be an r-way collision resistant or multicollision resistant hash family if, given a random key k, it would be hard to find an r-set $\{M_1, \cdots, M_r\}$ (known as r-way collision set or multicollision set) and an n-bit value z such that $H_k(M_i) = z$, $1 \leq i \leq r$. This event is known as multicollision.

3. **(2^{nd}) *Preimage resistance* :** In case of the preimage resistance, given a random output z and a random key k, it would not be easy to find a message M with $H_k(M) = z$. The message M is known as an *preimage* of z. In case of the 2^{nd} preimage resistance, given a fixed length random message M_1, it would not be easy to find a collision pair (M_1, M_2).

4. ***Target collision resistance* :** This is a variation of 2^{nd} preimage resistance for a hash family. This was previously known as Universal One-Way Hash Family or UOWHF introduced by Naor and Yung [74]. Later it is renamed under Target Collision Resistant by Bellare and Rogaway [7]. It says that it would not be easy to find a collision pair in the following game :
   - Commit a message M_1.
   - Given a random key $k \in \mathcal{K}$, find $M_2 \neq M_1$, $H_k(M_1) = H_k(M_2)$.

There are many constructions of hash functions designed from scratch. In those cases, it is not possible to prove the above resistance properties. But there were some evidences to resist some standard attacks. Now a list of some popularly known constructions of hash functions is given.

- **MD family** : MD2 [42, 66], MD4 [25, 82, 106, 110], MD5 [26, 43, 83, 94, 110], RIPEMD [23, 79, 110], HAVAL [86, 110, 114] etc.
- **SHA family** : SHA-0, SHA-1, SHA-256, SHA-512 [8, 75, 76] etc.
- **FFT family** : FFT-I [2, 95], FFT-II [96, 97, 105] etc.
- **Other Hash Functions** : TIGER [1], Whirlpool [3], PANAMA [18, 81] etc.

Apart from these hash functions, there are many block cipher based hash functions (see Sect. 2.5.2 and Sect. 2.5.4 for more details). To design a hash function, people

usually construct a *compression function* which has domain as a set of strings of fixed length, say $\{0,1\}^N$, and then iterate this compression function several times to make the domain $\{0,1\}^*$. The classical iteration is due to Merkle-Damgård [21, 61]. The method of iterations used in the classical iteration is very simple, efficient and easy to implement. Moreover, it preserves the collision resistance and preimage resistance properties. That is, the hash function is collision resistant and preimage resistant provided the underlying compression function is collision resistant and preimage resistant respectively. There are some other methods of iterations. For example, parallel iteration (which can be implemented in parallel) [62, 58, 88, 92, 88]. Thus, the construction of a hash function can be divided into two main steps. The first one is to construct a good compression function, $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$, for some positive integers $m, n > 0$. The second component considers how to iterate this compression function to define a hash function, $H : \{0,1\}^* \to \{0,1\}^n$. We call this *method of domain extension* or *design of iteration*. In Sect. 2.3.1, a detailed description of different design of iterations is given. The orientation of this thesis is mainly to study the design of iteration and cryptanalysis of hash functions.

1.1 Universal One Way Hash Family

Target collision resistance (*Universal One Way Hash Family or UOWHF*) was first introduced by Naor and Yung [74] in 1989. They constructed a UOWHF based on an one-way Permutation. Later, Rompel [87] showed that UOWHF exists under the assumption that one-way function exists. The target collision resistance has several importance over the collision resistance property for the following reasons;

1. **Theoretical existence :** UOWHF exists under the assumptions of existence of one-way function, but it is not known yet any theoretical construction of CRHF (collision resistant hash function) based on an one-way function is not yet known. Moreover, Simon [102] had shown that there is an oracle relative to which a UOWHF exists, but CRHF does not. But existence of collision resistant hash functions have been constructed under the assumption that discrete log problem [15, 60, 103] or factoring an integer [31] is hard.

2. **Level of security :** Target collision resistance is a weaker notion than the collision resistance property. In other words, a family of hash function is always UOWHF whenever it is CRHF, but the converse need not be true. Many attacks including the birthday attack for collision, which are applicable against collision resistance property, can not be applied in the case of the target collision resistance. Thus the security level of target collision resistance is expected to be more than that of collision resistance.

3. **Applications :** In many constructions, a CRHF can be replaced by a UOWHF, for example, in public key encryption [17, 100], in digital signature [7] etc. Naor and Yung [74] had designed a signature scheme by using a UOWHF. Bellare and Rogaway [7] had also designed a generic signature scheme based on UOWHF.

When a UOWHF is used in a signature scheme it is important to assume that the signer is honest, which might not be a practical assumption. Otherwise, a dishonest signer can choose key first and then he can choose a message which would to be signed. In this case, the security of the signature scheme depends on the collision resistant property, instead of the target collision resistant property. That is why UOWHF is not being used in any signature schemes in practice. However, in the light of the above discussions, UOWHF carries certainly some potential as well as theoretical importance.

One can design a UOWHF by constructing a compression function and then by applying some design of iterations. Unlike in collision resistance, the classical iteration (also tree based iteration) can not be applied here as it does not preserve the target collision resistance or "UOWHF" property [7]. Bellare and Rogaway [7] suggested a mask based parallel domain extension which preserves "UOWHF". A domain extension algorithm is said to be a UOWHF-preserving domain extension algorithm if it extends a compression function to a UOWHF provided the underlying compression function is a UOWHF (see Sect. 2.4 for more detail discussion). A *mask* is a part of the key. Here the key size (related to the number of masks) of the hash function grows as the domain of the hash function increases. This is another weak point of UOWHF. There are many other parallel and sequential designs. For more details, see Sect. 2.5.1. The signature scheme proposed by Bellare and Rogaway [7] uses the key k of the hash family as both input and output of the signature algorithm. Therefore, the shorter the key, better the signature scheme. These facts lead us to design UOWHF-preserving

domain extensions which have less number of mask.

Our Contribution

In Chapter 3 of this thesis, construction of two new parallel universal one way hash families are given. The first one, known as IBTC [53, 54, 67], is based on a complete binary tree and its key size is minimum among the class of all known complete binary tree based constructions [7, 89]. In fact, it is optimum in a wide subclass of binary tree based UOWHF-preserving domain extensions (also known as "*valid*" domain extension). A strong evidence [68] has been shown to claim that it is optimum in the class of all complete binary tree based valid domain extensions. The second construction, known as OPT [68, 69], has minimum key size and almost maximum parallelism among all mask based constructions. A sufficient condition has been provided to check "valid" property [69]. Surprisingly, we see that all known constructions [7, 53, 54, 67, 68, 69, 89, 91, 100] satisfy this sufficient condition.

1.2 PGV-Hash Families

A block-cipher is a family of permutations, $E : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$, which is used to design symmetric encryption schemes [19, 29]. An ideal block-cipher should behave like a random permutation (see Definition 2.4). If a block-cipher is indistinguishable with a random permutation then it is also called a "secure" block cipher. A secure block cipher can be used to construct a hash function (mainly the underlying compression function). We list some popular block cipher based compression functions, $f(h, x)$, where the key size is same as that of plain-text, i.e. $k = n$ and h, x are n-bit strings.

- Merkle's construction [61], Davis-Meyer [80, 111] compression function $(E_x(h) \oplus h)$, MMO [59], Miyaguchi *et al* [65] and Preneel [80] compression function $(E_x(h) \oplus x \oplus h)$ etc.
- Preneel, Govaerts, and Vandewalle [80] classified block cipher based compression functions, $E_a(b) \oplus c$, where $a, b, c \in \{h, x, h \oplus x, v\}$. Here, v is a fixed n-bit string. Note that, there sixty-four compression functions. These sixty-four compression functions are known as PGV-compression functions.

Black, Rogaway and Shrimpton [10] first analyzed the PGV hash function (based on PGV-compression functions) in the black-box model. The black box model is the model where an attack algorithm takes the underlying block cipher as a black box [98, 99]. They had shown [10] that twenty among sixty-four PGV-hash functions are collision and preimage resistant in the black box model. Since all PGV-hash functions are not hash families, the "UOWHF" property can not be studied here.

Our Contribution

In Chapter 4 of this thesis, the PGV-hash functions are generalized into generalized PGV-hash families by introducing a variable named, *key* of the hash family. We study the collision resistance, target collision resistance and preimage resistance properties and we see that forty-two among sixty-four generalized PGV-hash families are secure [55, 56]. We have also shown that security level of these secure hash functions are tight and hence can not be further improved.

1.3 Multicollision Attack

Multicollision security is a generalized notion of collision security. Instead of a pair, we have a set of different inputs whose hash values are same. The multicollision attack has several applications including Joux's attack [41] on a concatenated hash function. There are some other practical applications where multicollision secure hash functions are required. For example, the micro-payment scheme Micromint [84], the identification scheme of Girault and Stern [33], the signature scheme of Brickell *et al* [12] etc. Joux showed that the classical iteration is vulnerable to the multicollision attack. The same attack can be carried through in case of tree based hash functions. Recently there is an efficient second preimage attack on classical hash functions [44]. Thus, it is worthwhile to design a modified version of classical iteration and study the security properties.

Our Contribution

In the light of the above discussion it remains an open question how to find a good design of hash functions secure against multicollision attack. In Chapter 5 of this thesis,

some negative results to this question have been found. Here, a general definition of the classical hash function by hashing message blocks more than once has been studied. This generalization is a natural way to fix the classical iteration and it is therefore worthwhile to study this approach in detail. Unfortunately, there are efficient multicollision attack on generalized hash functions [73]. Also an 2^K-way collision attacks on n-bit generalized sequential hash functions with time complexity $O(nK^2 2^{n/2})$ is given, $K > 0$. This is a reasonable improvement over the birthday attack. A multicollision attack [73] with time complexity $O(nK^2 2^{n/2})$ on the class of generalized tree based hash functions (or a parallel design of hash functions) is given. Thus, a natural and a big class of design of hash functions in finding multicollision secure hash functions is being ruled out.

1.4 Double Length Compression or Hash Functions

Birthday attack is the most popular attack in hash function (see Sect. 2.3.4) [77, 104]. The birthday attack is generic in nature and hence can be applied to any hash function. The birthday attack on n-bit hash functions has time complexity $O(2^{n/2})$. For small values of n, the birthday attack can be practically feasible. In many block cipher based hash functions, the values of n are small (e.g. $n = 64$ or 128 [19, 45, 111]). One way to avoid these is by constructing a hash function from scratch where the output size is large. For example, SHA-256, SHA-512 [75, 76] etc. The other way is to design a hash function of size $2n$ from one or more than one underlying n-bit compression function(s). In this thesis, we are interested to study the latter and hence we consider the following problem;

Problem : *Given a "good" compression function, $f : \{0,1\}^{n+m} \to \{0,1\}^n$ (or s compression functions $f_1, \cdots, f_s : \{0,1\}^{n+m} \to \{0,1\}^n$), how to design a collision resistant or preimage resistant compression function $F : \{0,1\}^N \to \{0,1\}^{2n}$ and a hash function $H : \{0,1\}^* \to \{0,1\}^{2n}$, where $N > 2n$.*

The compression function, $F : \{0,1\}^N \to \{0,1\}^{2n}$, is termed as a *double length* compression function and the hash function $H : \{0,1\}^* \to \{0,1\}^{2n}$ is known as a *double*

length hash function. It may not be possible to design a collision resistant double length compression function (or hash function) by assuming only that the underlying compression function is collision resistant. Thus a stronger assumption that the underlying compression function is a random function [14] (also see Sect. 2.3.3) is needed. The term "good" compression function means that the function is a random function.

The most natural and efficient construction of a double length hash function is the concatenated hash function $H||G$, where H and G are two classical n bit hash functions based on a compression function $f(\cdot)$ with two different initial values. $H(\cdot)$ and $G(\cdot)$ can also be based on two different compression functions f_1 and f_2. Recently, A. Joux [41] showed that there is a collision attack on the concatenated hash function in time complexity $\mathrm{O}(n2^{n/2})$.

One can define a class of double length block cipher based hash functions as follows :

$$H_i = E_A(B) \oplus C \text{ and } G_i = E_D(E) \oplus F,$$

where $A, B, \cdots, F$ are some functions (e.g., linear combinations) of H_{i-1}, G_{i-1}, M_i. Here, M_i is the i^{th} message block. Some popularly known double length hash functions are MDC-2 [11], MDC-4 [63], Yi-Lam [112, 113] hash functions, LOKI [13], error-correcting codes based hash functions [49] etc. In [36, 48, 93, 107], it were shown that a wide class of block cipher based hash functions are not secure. Thus it is an interesting problem to design a secure double length hash function. Recently, Lucks [28, 57] and Hirose [37] designed secure double length hash functions, but the efficiency of their hash functions are poor and in some cases stronger assumptions are required.

Our Contribution

In Chapter 6 of the thesis, several double length compression functions [70, 71] and hash functions with security analysis are given. We show most of them are maximally secure. We design a simple double length compression function based on three independent compression functions [72] and prove that the security level is more than that of the underlying compression functions. We design an efficient double length hash function [71]. The efficiency of the construction is almost same as that of the insecure concatenated hash functions and the security level is almost the maximum security

level. Finally, we propose the best known efficient double length hash function and we leave the security of this hash function as a research problem.

1.5 Organization of The Thesis

This thesis is based on our following papers [53, 54, 55, 56, 67, 68, 69, 71, 72, 73, 70]. In Chapter 1, we give an introduction of hash function including universal one way hash family, PGV-hash functions, double length hash functions and multicollision attack on hash function. We briefly describe motivation and contribution of the thesis. In Chapter 2, we built a background of the thesis. This includes a short note on standard mathematical terminologies, graph theory [24], theory of hash function, the black box or random oracle model [98, 99] and the behavior of adversary in random oracle model. We also provide a detailed description of literature survey which motivate our research work presented in Chapter 3 to Chapter 6. At the end of Chapter 2 we state our main results of the thesis.

Chapter 3 is based on our papers [53, 54, 67, 68, 69]. We prove a sufficient condition for UOWHF-preserving (or "valid") domain extensions. We also describe our two new parallel domain extensions and study the "valid" property and optimality of our constructions.

Chapter 4 is based on our papers [55, 56]. We generalize the definition of PGV hash functions and study their security properties.

Chapter 5 is based on our paper [73]. Here we first note some limitation of classical hash functions and then we consider a natural wide class of domain extensions, generalized sequential hash functions and generalized tree based hash function. Finally, we propose several attacks on generalized sequential hash functions and generalized tree based hash functions.

Chapter 6 is based on our papers [71, 72, 70]. In this chapter, we study different methods of double length hash functions. Then we propose a class of double length hash functions of rate 1/4, a rate 1/3 double length hash functions and finally a rate close to 1/2 (also a rate close to one) double length hash functions. Finally, we conclude and state future research work in Chapter 7.

Chapter 2

Preliminaries and Related Works

2.1 Basic Definitions and Notations

The mathematical object "set" is most important language in many scientific areas. The following notations from set theory are used in this thesis. A set, A, is called an l-set if the number of elements of A is l. We say that, the *size* (or *cardinality*) of the set A is l and it is denoted by $|A|$.

- $\mathbf{N} = \{1, 2, \cdots\}$: set of natural numbers.
- $\mathbf{Z} = \{0, 1, -1, 2, -2, \cdots\}$: set of integers.
- $[a, b] = \{a, a+1, \cdots, b\}$, where $a \leq b$ are integers.
- $\mathbf{Z}_l = [1, l]$, $l \in \mathbf{N}$.

Besides the standard set theoretic operations union "$\cup$", intersection "$\cap$", minus "$\setminus$", there are some other operations which are *unordered* or *ordered cartesian product*. Given two sets A and B, the unordered cartesian product is the set $\{\{x, y\} : x \in A, y \in B \text{ and } x \neq y\}$. We use the notation $V^{(2)}$ to denote the unordered cartesian product of V by itself i.e. $V^{(2)} = \{\{x, y\} : x, y \in V, x \neq y\}$. In other words, $V^{(2)}$ denotes the set of all 2-subsets of V. The ordered cartesian product is denoted as $A \times B = \{(x, y) : x \in A, y \in B\}$. For $n \in \mathbf{N}$, $V^n = V \times \cdots \times V$ (n-times).

An n-bit string (or binary string) is an element from $\{0, 1\}^n$. We use the notation $x = x_1 x_2 \cdots x_n$ to denote $(x_1, x_2, \cdots, x_n)$, where $x_i = 0$ or 1. The length of the string x is n and we write $|x| = n$. Let λ denote the *empty string*, the string of length

zero. We write $\{0,1\}^0 = \{\lambda\}$, $\{0,1\}^* = \cup_{i=0}^{\infty}\{0,1\}^i$ and $\{0,1\}^{\leq N} = \cup_{i=0}^{N}\{0,1\}^i$. Let $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_m$ be n and m bit strings respectively. Then the *concatenation* of x and y, denoted by $x||y$ (or xy only), is the $(n+m)$ bit string $x_1\cdots x_ny_1\cdots y_m$. We use x^i to denote the string $x\cdots x$ (i times), where $x = 0$ or 1.

A binary relation $\prec$ on a set A is a totally order if the following holds :

1. for any pair $a \neq b \in A$ either $a \prec b$ or $b \prec a$.
2. if $a \prec b$, $b \prec c$ then $a \prec c$ (transitivity).

Let $\prec$ be totally order on an n-set A. Then we have $a_1 \prec a_2 \prec \cdots \prec a_n$, where $A = \{a_1, a_2, \cdots, a_n\}$. Let $\{x_a : a \in A\}$ be a set of finite binary strings. For a fixed totally order $\prec$ on A as above, define x_A by the string $x_{a_1}||\cdots||x_{a_n}$.

We use some complexity theoretic notations, namely, $\mathrm{O}(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$ and $\mathrm{o}(\cdot)$. Let $f, g : \mathbf{N} \to \mathbf{N}$ be two functions on a set of natural numbers. We write

- $f(n) \in \mathrm{O}(g(n))$ if there exists $C > 0$ such that $\frac{f(n)}{g(n)} \leq C$, for all $n > 0$.
- $f(n) \in \Omega(g(n))$ if there exists $C > 0$ such that $\frac{g(n)}{f(n)} \leq C$, for all $n > 0$.
- $f(n) \in \Theta(g(n))$ if $f(n) \in \mathrm{O}(g(n))$ and $g(n) \in \mathrm{O}(f(n))$.
- $f(n) \in \mathrm{o}(g(n))$ if $\lim_{n\to\infty}\frac{f(n)}{g(n)} = 0$.

We use the notations $f(n) = \mathrm{O}(g(n))$ instead of $f(n) \in \mathrm{O}(g(n))$. Similarly, $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$ and $f(n) = \mathrm{o}(g(n))$. See [60, 103] for more details.

2.2 Notes on Graph Theory

A graph, G, is a pair (V, E), where $E \subset V^{(2)}$. V is known as the *set of vertices* and E is known as the *set of edges*. A directed graph, G, is a pair (V, E), where $E \subset V^2 \setminus \{(x, x) : x \in V\}$. Here, E is known as the set of *arcs*. Given a directed graph, we can obtain an *induced graph* by replacing arcs by edges. Thus, for a directed graph $G = (V, E)$, the induced graph is $G_1 = (V, E_1)$, where $E_1 = \{\{x, y\} : (x, y) \in E\}$. Sometimes, we use uv (or $u \to v$) to denote an edge (or arc) instead of $\{u, v\}$ (or (u, v)), where $u, v \in V$.

The set of neighbors of a vertex v is $\{u \in V : uv \in E\}$ and it is denoted by $N(v)$. The degree $d(v)$ of a vertex v is $|N(v)|$. In case of a directed graph, we have two notions of degrees. The in-degree of a vertex v is $\text{indeg}(v) = |\{u \in V : u \to v\}|$ and the out-degree is $\text{outdeg}(v) = |\{u \in V : v \to u\}|$.

A labeled directed graph, $G = (V, E)$, where $E \subset V \times V \times L$, for some set L known as *label set*. A labeled arc $(u, v, x) \in E$ is also pictorially denoted by $u \to_x v$.

Definition 2.1 *$G = (V, E)$ is called a* tripartite *graph (see Fig. 2.1) if there are three disjoint nonempty sets of vertices A, B and C such that $A \cup B \cup C = V$ and $E \subset \{uv : u \in A, v \in B$ or $u \in A, v \in C$ or $u \in B, v \in C\}$.*

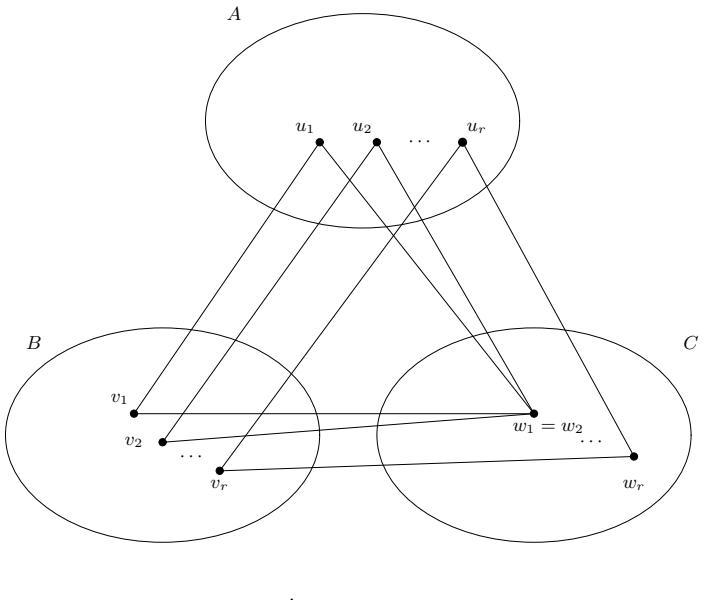

Figure 2.1: Tripartite Graph

A *subgraph* (*or directed subgraph*), $G_1 = (V_1, E_1)$, of $G = (V, E)$ is a graph (or directed subgraph) such that $V_1 \subset V$ and $E_1 \subset E$. For $k \geq 0$, a *path* P (from v_0 to v_k) of length k of a graph G is a subgraph (or directed subgraph) (V_1, E_1) of the form, $V_1 = \{v_0, v_1, \cdots, v_k\}$, $E_1 = \{v_0v_1, v_1v_2, \cdots, v_{k-1}v_k\}$ (or $E_1 = \{v_0 \to v_1, \cdots, v_{k-1} \to v_k\}$). We use the notation $v_0 \Rightarrow v_k$.

If there is a path P from v_0 to v_k, the vertices v_0 and v_k are said to be *connected* or connected by P. By convention, a vertex v is always connected by itself by a path of length zero. A graph is said to be connected if any two vertices are connected.

A *cycle* C, of a graph (or directed graph) G is a subgraph (V_1, E_1) of the form, $V_1 = \{v_0, v_1, \cdots, v_k\}$, $E_1 = \{v_0v_1, v_1v_2, \cdots, v_{k-1}v_k, v_kv_0\}$ (or $E_1 = \{v_0 \rightarrow v_1, \cdots, v_{k-1} \rightarrow v_k, v_k \rightarrow v_0\}$).

2.2.1 Rooted Directed l-ary Tree

A connected *acyclic* graph (which does not contain any cycle) is called a *tree*. A directed tree is a directed graph where the induced graph is a tree. We use the notation T for a tree.

A rooted directed tree $T = (V, E)$ is a directed tree in which, there is a special vertex q (known as *root*) and for any other vertex v, $v \Rightarrow q$.

For an integer $l \geq 2$, an l-ary rooted directed tree is a rooted directed tree so that $\text{indeg}(v) \leq l$ for all $v \in V$. The vertices with in-degree zero are known as leaves and all other vertices except root are known as intermediate nodes.

For any vertex $u \neq q$, there is a unique arc $u \rightarrow v$ which is denoted by e_u. The vertex v is known as *father* of the vertex u and u is *son* of v. $\text{son}(v) = \{u : u \rightarrow v\}$ is the set of sons of v. In case of binary tree, there are at most two sons. These are termed as left son and right son. Note that $\text{indeg}(v) = |\text{son}(v)|$.

Level of a vertex v, denoted as $l(v)$, is the number of vertices of the path from v to the root q. Thus $l(q) = 1$. *Height* of the tree $ht(T)$ or t (say) is the maximum level of any vertex. We define height of a vertex v by $t + 1 - l(v)$ and denote it by $ht(v)$.

The set of leaves of the tree T is denoted by $L[T]$ (or L only) and the set of intermediate nodes is denoted by $U = U[T] = V \setminus (L \cup \{q\})$. Given a vertex $v \in V$ in a rooted directed tree T, we can define an induced subtree rooted at v, $T[v] = (V[v], E[v])$, where $V[v] = \{u \in V : u \Rightarrow v\}$ and $E[v] = \{(u_1, u_2) \in E : u_1, u_2 \in V[v]\}$. We use $L[v]$ to denote $L[T[v]]$.

Examples.

1. A 2-ary tree is also known as a *binary* tree. A complete binary tree of height t is

$T = (V, E)$ where $V = [1, 2^t - 1]$ for some t and $E = \{\{i, \lfloor i/2 \rfloor\} : 2 \leq i \leq 2^t - 1\}$. A complete binary tree of height four is depicted in Fig. 2.6.

2. For $n \geq 1$, a path $\{v_1 \rightarrow v_2 \cdots \rightarrow v_n\}$ of length $n-1$ is also known as n-sequential tree with root v_n. We call the vertices v_1 and v_n end vertices.

3. **l-dim tree :** l-dim tree consists of several sequential tree in l many directions. For example, when $l = 2$, a 2-dim tree is shown in the Fig. 2.2. For integer t, suppose $a = \lceil t/2 \rceil$ and $b = t - a$. Let $T_t = (V_t, E_t)$, where $V_t = \{1, 2, \cdots, 2^t\}$ and $E_t = \{e_i : 2 \leq i \leq 2^t\}$. Here, $e_i = (i, i-1)$ for $2 \leq i \leq 2^a$, $e_i = (i, i - 2^a)$ for $2^a < i \leq 2^t$.

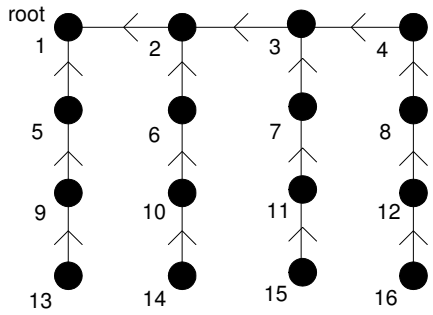

Figure 2.2: A 2-dim tree

2.2.2 Operation on Trees

Let $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ be two disjoint directed subtrees, i.e. $V_1 \cap V_2 = \emptyset$. A join of trees T_1 and T_2 is defined by $T_1 +_{(u_1,u_2)} T_2 = (V, E)$, where $V = V_1 \cup V_2$ and $E = E_1 \cup E_2 \cup \{(u_1, u_2)\}$ and $u_i \in V_i$ for $i = 1, 2$. Thus, T_1 and T_2 are joined by an arc (u_1, u_2). Similarly, we can join two un-directed trees T_1 and T_2 by an edge u_1u_2 and we denote it by $T_1 +_{u_1u_2} T_2$.

We can define *subtraction* by a subtree. If T_1 is a subtree of T then $T - T_1$ denotes the subtree of T by removing T_1 from T. Similarly we can define union of trees.

For $i \geq 2$, an i-λ-tree (see Fig. 2.3) is a join of an $(i-1)$-sequential tree and a binary tree, joined at the root of the binary tree with one end of the sequential tree. For $i = 1$, 1-λ-tree is nothing but a binary tree.

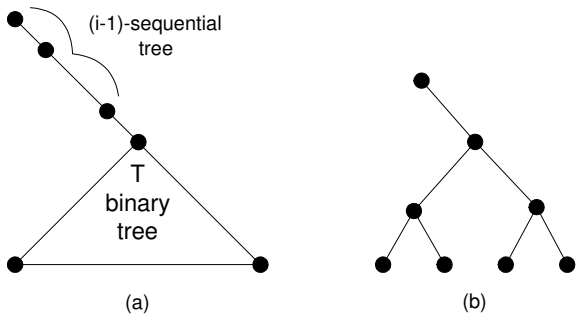

Figure 2.3: (a) An i-λ-tree or λ-tree, $i \geq 2$, (b) an example of 2-λ-tree

2.3 Hash Functions Methodology

2.3.1 Design of Iteration on Hash Function

A *hash function*, $H : \{0,1\}^* \rightarrow \{0,1\}^n$, is an easily computable function. But in practice, a function with domain $\{0,1\}^{\leq N}$, for large integer N, would suffice. A hash function $H : \{0,1\}^N \rightarrow \{0,1\}^n$ is also termed as (N,n) hash function. A family of (N,n) hash functions $\{H_k\}_{k\in\{0,1\}^\kappa}$ is termed as (N,n,K) hash family. $k \in \{0,1\}^K$ is known as a key of the hash function H_k.

Usually, a hash function is designed in two steps. In the first step, a small domain compression function, $f : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$, $m > 0$ is designed. Compression functions are either designed from scratch or it is based on some other primitives like block-cipher. After defining a compression function, we extend the domain to define a hash function. We call a method of domain extension by *design of iteration* as we iterate a compression function several times. Further, we divide the design of iteration in following two main steps:

1. **Padding Rule :** We append some unambiguous pad including a binary representation of the length of the message, for example, the Merkle-Damgård padding [61, 21]. Padding is essential as the domain of the underlying compression function is of the form $\{0,1\}^{n+m}$. It also helps to avoid some trivial

attacks on a hash function [60].

2. **Combining Rule :** After having a suitable sized padded message M, we try to iterate the compression function $f(\cdot)$ in a particular way. Combining rule says the rule of combination of the compression functions.

For example, let $f : \{0,1\}^{2n} \rightarrow \{0,1\}^n$ be a compression function. We define two extended functions, $F_1, F_2 : \{0,1\}^{4n} \rightarrow \{0,1\}^n$ as follows:

- $F_1(x_1||x_2||x_3||x_4) = f(f(f(x_1||x_2)||x_3)||x_4)$,
- $F_2(x_1||x_2||x_3||x_4) = f(f(x_1||x_2)||f(x_3||x_4))$,

where $|x_1| = |x_2| = |x_3| = |x_4| = n$. The functions F_1 and F_2 can be captured by trees in Fig. 2.4. We have placed the underlying compression function f on each node and through the arcs the outputs of f are flowing. These two directed trees are the underlying trees of the extended functions. In the case of F_1, the flow is sequential and hence the underlying tree is a sequential tree and in the case of F_2, the underlying tree is a binary tree since the input of the final invocation is the concatenation of the outputs of the previous invocations of f. Two functions F_1' and F_2' can be defined on $\{0,1\}^{\leq(4n-1)}$: $F_j'(X) = F_j(X||10^i)$, where $i = 4n - 1 - |X|$ and $j = 1$ or 2.

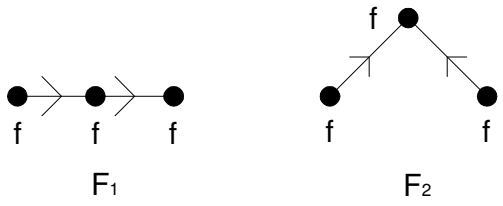

Figure 2.4: The Combining Rule of Iteration

In this thesis, we are mainly interested in the design of iteration (more precisely, in the combining rule) and study the security properties of hash functions. Thus given a good underlying compression function we define several good hash functions. Unless we mention, the underlying compression function is $f : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$.

1. Classical Iteration [21, 61]

Here we define a simplified version of Merkle-Damgård method. Let M be a message, $|M| < 2^m$ and let h_0 be an n-bit initial value. We choose smallest $i \geq 0$ such that $|M| + i + 1$ is multiple of m. Let $\langle |M| \rangle$ be the m-bit binary representation of $|M|$. Thus we write $M||10^i||\langle |M| \rangle = m_1||m_2 \cdots ||m_l$ for some positive integer l and $|m_i| = m$, $1 \leq i \leq l$. Now define $H(M)$ by the following method:

$$H(M) = h_l, \text{ where } h_i = f(h_{i-1}||m_i), \ 1 \leq i \leq l. \tag{2.1}$$

We also use the notation H^f to denote the classical hash function H based on the compression function f. The underlying tree of the classical iteration is a path (or sequential tree). This hash function is also known as a sequential hash function. Note that the domain of the hash function is $\{0,1\}^{\leq N}$, where $N = 2^m - 1$.

Consider a set of vertices $V = \{0,1\}^n$. We use the notation $h \rightarrow_x h'$ (a labeled arc) to mean $f(h,x) = h'$. Here, $|h| = |h'| = n$ and $|x| = m$. Thus, the computation of $H(M)$ can be represented by a labeled path from h_0 to h_l as follows:

$$h_0 \rightarrow_{m_1} h_1 \rightarrow_{m_2} h_2 \cdots h_{l-1} \rightarrow_{m_l} h_l \quad \text{or} \quad h_0 \Rightarrow_M h_l.$$

Thus, $h_0 \Rightarrow_M h_l$ if and only if $H(M) = h_l$. In Sect. 5.3, we generalize the classical iteration where the message block can be hashed more than once.

2. Parallel Design of Iteration

Next we discuss a general class of parallel methods for design of hash functions [58, 88]. Define a class of hash functions where a hash function H from the class behaves in the following manner;

1. It invokes f, a finite number of times.
2. The entire output of any intermediate invocation (not the final invocation) is fed into the input of other invocations of f.
3. Each bit of the message, M, is fed into at least one invocation of f.

4. The output of the final invocation is the output of the hash function, H.

Note that, the above class of hash functions includes all natural design of iteration. We make following reasonable assumptions to make the class simple and easy to study.

1. The input of f is of the form $h||x$ or x, where h is concatenation of output of previous invocations of f and x is concatenation of message blocks.

2. Output of any invocation of f (except the final invocation) is fed exactly once. Also each message block is hashed exactly once.

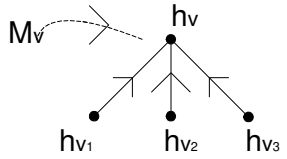

Figure 2.5: The Combining Rule of Iteration at node v

Thus, we have a rooted directed tree $T = (V, E, q)$. From now onwards we fix a totally order $\prec$ on V. Let $M = M_V = M_{v_1}||\cdots||M_{v_r}$ be a message, where $m + n - |M_v|$ is a non-negative multiple of n. In case of $m = kn$, for some integer k, M_v may be an empty string. M_v denotes the message block which is fed into f placed at the node v. For each node v, we recursively assign an n-bit string h_v (intermediate hash value) as follows (see Fig. 2.5);

$$h_v = f(s_v||M_v) \text{ if } v \in L, \text{ otherwise } h_v = f(h_{son(v)}||M_v). \tag{2.2}$$

We define, $H(M_V) = h_q$. Here s_v is some fixed initial value, L is the set of leaves of the tree T and q is the root of the tree. For simplicity assume that, either $|s_v| = n$ or $|s_v| = 0$. Here we need

$$|M_v| = n + m - |s_v| \text{ if } v \in L \text{ and } |M_v| = n + m - \text{indeg}(v) \times n \text{ otherwise.} \tag{2.3}$$

We need to assume that $m \geq |s_v| - n$ and $m \geq (\text{indeg}(v) - 1) \times n$, for all $v \in V$, so that the hash function is well defined. We apply similar padding rule as in the classical iteration in order to divide the message into several blocks (M_v's) which satisfy 2.3. We say that the hash function, H, is based on tree T and write $\mathcal{T}$ to denote the class of all such hash functions based on a tree. Note that a hash function is defined on a domain $\{0,1\}^{\leq N}$, for large integer N. If a function H is defined on $\{0,1\}^N$ then by applying padding 10^i we can define a function H' on $\{0,1\}^{\leq (N-1)}$. In Sect. 5.5, we generalize the tree based iteration where the message block can be hashed more than once.

3. Mask Based Parallel Iteration

Let $\{f_k\}_{k\in\{0,1\}^\kappa}$ be an $(n+m, n, K)$ hash family and $T = (V, E, q)$ be a rooted directed tree with $|V| = r$. Let $\psi : V \setminus \{q\} \rightarrow \mathbf{Z}_l$ be a function known as a *masking assignment* (also an l-masking assignment). We define an $(n + rm, n, P)$ hash family $\{H_p\}_{p\in\{0,1\}^P}$ where $P = K + nl$. Let $p = k||\mu_1|| \cdots ||\mu_l$ where $|k| = K$ and $|\mu_i| = n$, $1 \leq i \leq n$. μ_i's are known as masks. Like the parallel iteration, we recursively define n-bit strings h_v and z_v for each node v as follows:

$$z_v = h_v \oplus \mu_{\psi(v)} \text{ where } h_v = f_k(M_v) \text{ if } v \in L,$$
$$h_v = f_k(z_{son(v)}||M_v) \text{ if } v \notin L.$$

We define $H_p(M) = h_q$. Now, $|M| = \sum_{v\in V} |M_v| = n + r \times m$. This can be proved by using induction on r [90]. We can also include initial value and in that case $h_v = f(s_v||M_v)$ for $v \in L$. But the domain size of the hash function will be reduced and hence it is less efficient. For simplicity we do not include any initial value in the mask based iteration. We say that the hash function is based on the tree T and the masking assignment ψ and we denote the class by $\mathcal{M}$. The pair $\mathbf{A} = (T, \psi)$ is known as the *structure* of the hash function. Thus it is enough to describe a structure while we define a hash function. The hash family $\{f_k\}$ is known as *base hash family* and the hash family $\{H_p\}$ is known as the *extended hash family*. We also write $\mathbf{A} : \{f_k\} \rightarrow \{H_p\}$ or $\mathbf{A} : (n + m, n, K) \rightarrow (n + rm, n, K + nl)$. To define a hash function H_p on $\{0,1\}^{\leq N}$ for large N, we can choose $N = n + rm$ for large r and then we apply some standard padding rule, as stated earlier, to make the domain $\{0,1\}^{\leq (N-1)}$.

2.3.2 Types of attacks

We first list some standard attacks on hash function. Let $\{H_k\}_{k\in\mathcal{K}}$ be a hash family. Note that, $|\mathcal{K}|$ can be one and in this case it is a single function.

1. **(Multi-) Collision attack :** Given a random key $k \in \mathcal{K}$, find $M_1 \neq M_2$ such that $H_k(M_1) = H_k(M_2)$. This pair (M_1, M_2) is also known as a *collision pair* of the hash function H and the game is denoted by Coll. Similarly, given a random key k, find an r-set C and $z \in \{0,1\}^n$ such that for any $M \in C$, $H_k(M) = z$. This set C is known as a *r-way collision set* (or *multicollision set*) and z is known as the *collision value* of the set C.

2. **(2^{nd}) Preimage attack :** Given a random $z \in \{0,1\}^n$ and a random key k, find a binary string M such that $H_k(M) = z$. This M is known as the *preimage* of the n-bit string z and the game is denoted by Inv. In case of second preimage attack, given a random message M, find $M' \neq M$, such that $H(M) = H(M')$.

3. **Target collision attack :** There is one more attack which is a little modification of the 2^{nd} preimage attack. Find a collision pair (M_1, M_2) in the following game, called TColl.

   (a) Commit an $(n+m)$-bit string M_1 (guess stage).

   (b) Given a random key $k \in \{0,1\}^K$, find $M_2 \neq M_1$ such that $H_k(M_1) = H_k(M_2)$ (find stage).

The success probability or advantage of an attack algorithm $\mathcal{A}$ is defined as follows:

Definition 2.2 (Collision resistance, target collision resistance, and inversion resistance of an extended hash family '$\mathcal{H}$') *Let $\mathcal{H} = \{H_k\}_{k\in\mathcal{K}}$ be an extended hash family. Then the advantage of $\mathcal{A}$ with respect to (target) collision resistance and inversion resistance are the following real numbers.*

$$\begin{aligned}
\mathbf{Adv}^{Coll}_{\mathcal{H}}(\mathcal{A}) &= Pr[k \xleftarrow{R} \mathcal{K}; M, M' \leftarrow \mathcal{A} : M \neq M' \ \&\ H_k(M) = H_k(M')] \\
\mathbf{Adv}^{TColl}_{\mathcal{H}}(\mathcal{A}) &= Pr[M \leftarrow \mathcal{A}_{guess}; k \xleftarrow{R} \mathcal{K}; M' \leftarrow \mathcal{A}_{find}(M, k) :
\end{aligned}$$

$$M \neq M' \ \& \ H_k(M) = H_k(M')]$$
$$\mathbf{Adv}^{Inv}_{\mathcal{H}}(\mathcal{A}) = Pr[k \xleftarrow{R} \mathcal{K}; h^* \xleftarrow{R} \{0,1\}^n; M \leftarrow \mathcal{A} : H_k(M) = h^*]$$

Similarly we can define advantage for compression functions. If $\mathcal{A}$ is an adversary making some queries and $\mathbf{Adv}^{XXX}_{Y}(\mathcal{A})$ is a measure of adversarial advantage then we write $\mathbf{Adv}^{XXX}_{Y}(q)$ to mean the maximal value of $\mathbf{Adv}^{XXX}_{Y}(\mathcal{A})$ over all adversaries $\mathcal{A}$ that use queries bounded by the number q. Sometimes we also use q to denote the root of a binary tree. But from the context the meaning of q would be very much clear.

2.3.3 Random Function and Random Permutation

Design of a hash function secure against all possible attack algorithms is desirable. Thus, we need to formulate the definition of security in more detail. We also need to specify the power and model of the adversary trying to achieve some goals.

When proofs in the standard model are unappealing or provably impossible, we use some alternative model like random oracle model or black-box model. In practice this model does not exist. But when we are using this model there would be some suitable object to implement. This is known as instantiation of the random oracle model. Block-cipher and cryptographic hash functions are common building blocks for this. Till date, AES is appreciated as a good block-cipher and no properties of AES are known which might be a threat for the cryptographic protocol where it is being instantiated.

In this thesis, we mainly assume that the underlying compression function is a random function. We also design secure compression functions or hash functions based on secure block-ciphers. In this case, we assume the underlying block-cipher as a family of random permutations indexed by the key of the block cipher. There are several mathematical formulations of a random function or a random permutation. Let $\mathcal{F}^{D \to R}$ be a set of all functions from D to R. A randomly chosen function f from $\mathcal{F}^{D \to R}$ is known as a *random function* [14]. Similarly a randomly chosen permutation p from $\mathcal{P}^{D \to D}$ is known as a *random permutation* where $\mathcal{P}^{D \to D}$ is a set of all permutations on the set D. One can define a random function and a random permutation in the following equivalent way:

Definition 2.3 *A* random function *f from D to R taking values as a random variable such that for any $x \in D$, $f(x)$ has uniform distribution on R and for any k (> 0) distinct elements $x_1, \cdots x_k \in D$, the random variables $f(x_1), \cdots, f(x_k)$ are independently distributed.*

Definition 2.4 *A* random permutation *E on D taking values as a random variable such that for any $k > 0$ and k distinct elements $x_1, \cdots, x_k \in D$, the random variable $E(x_k)$ condition on $E(x_1) = y_1, \cdots, E(x_{k-1}) = y_{k-1}$ is uniformly distributed over the set $D - \{y_1, \cdots, y_{k-1}\}$.*

By abuse of notations, we write $f : D \to R$ and $E : D \to D$. We call $f_1, \cdots, f_s$ *independent random functions* if f_i's are chosen independently and randomly from the set $\mathcal{F}^{D \to R}$. Similarly for independent random permutations $E_1, \cdots, E_s$. We state an equivalent definition of independent random functions and permutations and state a proposition to generate a set of independent random functions.

Definition 2.5 *We say a family of functions $f_1, \cdots, f_s : D \to R$ are* independent *if for any s subsets $\{x_1^1, \cdots, x_{k_1}^1\}$, $\cdots$, $\{x_1^s, \cdots, x_{k_s}^s\}$, the random vectors $(f_1(x_1^1), \cdots, f_1(x_{k_1}^1)), \cdots, (f_s(x_1^s), \cdots, f_s(x_{k_s}^s))$ are independently distributed. We say $f_1, f_2, \cdots f_s : D \to R$ are* independent random functions *if they are random functions and independent too. Similarly, we say $E_1, E_2, \cdots, E_s : D \to D$ are* independent random permutations *if they are random permutations and independent too.*

Proposition 2.1 *If $f : \{0,1\}^{n+k} \to \{0,1\}^m$ is a random function then the family of functions $\{f_s\}_{s \in \{0,1\}^k}$, $f_s : \{0,1\}^n \to \{0,1\}^m$ defined by $f_s(x) = f(x||s)$, where $|x| = n$, are independent random functions. In particular, if $f : \{0,1\}^N \to \{0,1\}^n$ is a random function then $f_0, f_1 : \{0,1\}^{N-1} \to \{0,1\}^n$, $f_i(X) = f(i||X)$, $|X| = N - 1$ and $i = 0, 1$ are two independent random functions.*

The Adversary in the Random Oracle Model

In this thesis we consider the model of the adversary in the random oracle model which is due to Shanon [99]. When a hash function, $H(\cdot)$, is designed based on a random

compression function f an attack algorithm is an oracle algorithm $\mathcal{A}^f$ with the oracle f executing the following game:

- Adversary can ask several queries of f adaptively.
- Based on the query-response pairs adversary finally outputs.

Thus adversary can choose $x_1, \cdots x_q$ adaptively and gets responses $y_1, \cdots, y_q$, where $y_i = f(x_i)$. We can think y_i as a realization of the random variable $f(x_i)$ which is observed by the adversary. Define the complete list of query-response pairs $((x_1, y_1), \cdots, (x_q, y_q))$ by the *view* of the adversary. Any output produced by the adversary should only depend on the view. Moreover, if the adversary finds collision for a hash function, $H(\cdot)$, based on the compression function, $f(\cdot)$, and it outputs a pair of distinct messages $M \neq N$ then the values of $H(M)$ and $H(N)$ should be computed from the view. When we have two or more compression functions $f_1, f_2, \cdots$, we have a set of lists of pairs $\{((x_1^1, y_1^1), \cdots, (x_q^1, y_q^1)), ((x_1^2, y_1^2), \cdots, (x_q^2, y_q^2)), \cdots\}$ where the first member is the view due to the random compression function f_1 and so on. This set will be called a view of the adversary.

In case of block cipher based construction, an adversary, $\mathcal{A}^{E,E^{-1}}$, has access to oracles E and E^{-1} where E is the underlying block-cipher which is assumed to be a random permutations. Now the adversary can make two types of queries, E and E^{-1}. For E-query the adversary gives (a, x) and gets response y such that $E_a(x) = y$. Similarly for E^{-1} query. Here, the list of triples $((a_1, x_1, y_1), \cdots, (a_q, x_q, y_q))$ will be called a *view* of the adversary. Again we follow the similar conventions. Firstly, an adversary does not ask any oracle query on which the response is already known. Secondly, if M is one of the outputs produced by the adversary, then the adversary should make necessary E or E^{-1} queries to compute $H(M)$ during the whole query process. These assumptions are meaningful as any adversary $\mathcal{A}$ not obeying these conventions can easily be modified to obtain an adversary $\mathcal{A}'$ having similar computational complexity that obeys these conventions and has the same advantage as $\mathcal{A}$.

We define the complexity of an attack algorithm by size of the view to be required to have non negligible [5] probability of success or advantage. The minimum complexity of an attack algorithm is a measurement of security of the hash function.

In this thesis we use the terms random functions, black-box, and random oracle model. They all carry similar meanings. Also secure block cipher and random permutation

carry similar meaning. Depending on the context we will use the suitable terms.

2.3.4 Birthday Attack

A natural and most popular attack is the birthday attack. The following algorithm is the birthday attack for finding r-way collision of the function $g : D \to R$ with complexity q.

BirthDayAttack(g, q, r) **:**

1. Choose $x_1, \cdots, x_q$ randomly from the domain D of g and compute $y_i = g(x_i)$ for $1 \leq i \leq q$.

2. Return a subset (if any) $C \subseteq \{x_1, \cdots, x_q\}$ of size r such that C is an r-way multicollision subset for the function g. Otherwise return a string "failure".

The next Proposition gives an estimate of the complexity of the birthday attack in finding r-way collision with significant probability. Here we assume that the function, $g : D \to R$, is a *regular* function. A function g is called regular if $g(x)$ is uniformly distributed on R, whenever x is uniformly distributed over D. When g is not regular, the success probability of the birthday attack is more [6].

Proposition 2.2 (Complexity of the birthday attack) : *For any regular function $g : D \to \{0,1\}^n$ the birthday attack finds r-way collision with probability $O(q^r/2^{(r-1)n})$. Thus the birthday attack requires $\Omega(2^{n(r-1)/r})$ many queries to have an r-way collision with significant probability.*

Proof. Since $x_1, \cdots, x_q$ are randomly chosen from the domain D, $g(x_1), \cdots, g(x_q)$ are independent and uniformly distributed over R. Thus, for any $\{i_1, \cdots, i_r\} \subset [1, q]$ we have, $\Pr[g(x_{i_1}) = \cdots = g(x_{i_r})] = 1/2^{(r-1)n}$. Let $C_1, \cdots C_k$ be all r-subsets of $\{x_1, \cdots, x_q\}$ (the complete query list of the birthday attack) where $k = \binom{q}{r}$. Let E_i denote the event that C_i is an r-way multicollision set. Thus, the event corresponding to the existence of r-way collision in $\{x_1, \cdots, x_q\}$ is $\bigcup_i E_i$. Hence, the probability of having r-way collision set is

$$\Pr(\bigcup_i E_i) \leq \sum_i \Pr[E_i] = \frac{\binom{q}{r}}{2^{(r-1)n}} = \mathrm{O}(q^r/2^{(r-1)n}).$$

To make the probability significant q should be $\Omega(2^{n(r-1)/r})$. □

In case of the collision attack (or preimage or 2^{nd} preimage attack), the birthday attack for finding collision requires $\mathrm{O}(2^{n/2})$ (or $\mathrm{O}(2^n)$) queries to have significant success probability. See [60, 103] for more detail discussion. Note that the birthday attack for collision requires $\mathrm{O}(2^{n/2})$ many spaces to store all hash outputs. There is also a variant of birthday attack [77] which requires constant amount of space. To have a detail discussion on birthday attacks, one can see [6, 77, 103, 104, 108].

2.4 Universal One-Way Hash Family

UOWHF, Universal One Way Hash Family, was first introduced by Naor and Yung [74] in 1989. They constructed a UOWHF based on a one-way function. A UOWHF is an $(n+m, n, K)$ family of hash functions $\{f_k\}$, where the target collision attack (finding collision in "TColl" game, Sect. 2.3.2) is difficult. Thus target collision resistance hash family is known as Universal One-Way Hash Family or UOWHF. More precisely,

Definition 2.6 *$\{f_k\}_{k\in\mathcal{K}}$ is an (ϵ, t)-UOWHF, if every adversary with runtime at most t has at most ϵ success probability in target collision attack.*

Suppose a domain extension algorithm, $\mathbf{A}$ (see Sect. 2.3.1), extends an $(n+m, n, K)$ hash family $\{f_k\}_{k\in\mathcal{K}}$ to an (N, n, P) hash family, $\{H_p\}_{p\in\{0,1\}^P}$, $N > n+m$. In notation, $\mathbf{A} : (n+m, n, K) \to (N, n, P)$ or $\mathbf{A} : \{f_k\}_k \to \{H_p\}_p$.

Definition 2.7 *A domain extension algorithm, $\mathbf{A} = (T, \psi) \in \mathcal{M}$, $\mathbf{A} : (n+m, n, K) \to (N, n, P)$, is called* valid *or* UOWHF-preserving domain extension *if for any $(n+m, n, K)$ base hash family, $\{f_k\}_{k\in\mathcal{K}}$, which is (ϵ, t)-UOWHF, the extended $(n+rm, n, K+nl)$ hash family, $\{F_p\}_{p\in\mathcal{P}}$, is (ϵ', t')-UOWHF, where ϵ' and t' are constant multiples of ϵ and t respectively.*

In this thesis, we are interested in designing valid domain extensions where the *key expansion* i.e. $(P-K)$ is as small as possible. Also we try to reduce the time complexity

by constructing parallel domain extension algorithms. The domain extension from the class $\mathcal{T}$ (stated in Sec. 2.3.1) is not valid [7]. But there are several mask based valid domain extensions from $\mathcal{M}$. If we consider a weaker definition than UOWHF (providing some more power to the attacker) then it is possible to ignore masking up to some rounds [39].

2.4.1 Masking Assignment on a Tree

A masking assignment (or l-masking assignment) on a rooted directed tree $T = (V, E, q)$ is a function $\psi : V \setminus \{q\} \rightarrow \mathbf{Z}_l$, $l \geq 1$. We also define the masking assignment ψ on E, where $\psi(e_v) = \psi(v)$. Now we state some terminologies on a masking assignment.

Definition 2.8 ((strongly) even-free masking assignment)

A l-masking assignment ψ on $T = (V, E)$ is called even-free *masking assignment if for any non-trivial sub-tree $T_1 = (V_1, E_1)$ of T there exists $i \in \mathbf{Z}_l$ such that i appears odd many times in the sequence $\psi(E_1) = \langle \psi(e_1), \cdots, \psi(e_k) \rangle$ where $E_1 = \{e_1, \cdots, e_k\}$. If for every subtree T_1 there exists an i such that i appears exactly once in the sequence $\psi(E_1)$ then we say ψ is* strongly even-free. *This i is known as a* "single man" *of T_1.*

Proposition 2.3 (Properties of strongly even-free masking assignment)

1. *If ψ is strongly even-free on T then for any subtree T' the function ψ restricted on T' is also strongly even-free.*

2. *If ψ is a strongly even-free masking assignment on two disjoint subtrees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ such that $(u_1, u_2) \in E$ where $u_i \in V_i$, $i = 1, 2$ then ψ is also a strongly even-free masking assignment on the subtree $T_1 +_{(u_1,u_2)} T_2$ provided $\psi((u_1, u_2))$ does not appear in the multiset $\psi(E_1)$ and $\psi(E_2)$.*

Proof. If T'' is a subtree of T' then it is also a subtree of T and hence the first statement follows. Let T' be a subtree of $T = T_1 +_{(u_1,u_2)} T_2$. If it contains (u_1, u_2) then the mask assigned on (u_1, u_2) is a single-man. Otherwise it is a subtree of either T_1 or T_2. So, the second statement is proved. □

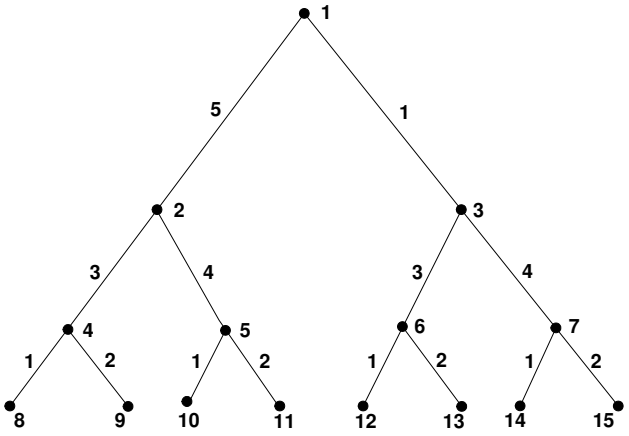

Figure 2.6: The tree T and the masking assignment ψ

Consider the masking assignment shown in Fig. 2.6. In this figure the masking assignments are described by the numbers written beside edges. Consider 2-λ-tree $T_1 = T[3] +_{31} \{1\}$. If a subtree of T_1 contains the edge $(6,3)$ or $(7,3)$, then either 3 or 4 is a single-man. Otherwise 1 is a single-man. Thus ψ on T_1 is strongly even-free. By the first part of Proposition 2.3, ψ is strongly even-free on $T_2 = T[2]$. Thus, it is strongly even-free on $T = T_2 +_{21} T_1$ as the mask 5 appears only once in the arc $(2,1)$ (see Proposition 2.3).

Definition 2.9 (level-uniform masking assignment)

A level-uniform masking assignment on a k-ary tree T is a masking assignment ψ such that for any two vertices u and v with $l(u) = l(v)$, $u_1 \prec \cdots \prec u_k$ and $v_1 \prec \cdots \prec v_k$, ($u_i$'s and v_i's are sons of u and v respectively) we have $\psi(u_i) = \psi(v_i)$ for all $1 \leq i \leq k$.

The masking assignment, given in Fig. 2.6, is level-uniform. Consider a complete binary tree $T = (V, E)$ of height t. Define $\alpha, \beta : [1, t-1] \to \mathbf{Z}_l$ by $\alpha(i) = \psi(2^{t-i})$ and $\beta(i) = \psi(2^{t-i} + 1)$. If ψ is level-uniform masking assignment then the function ψ is totally determined by α and β. An edge e_i is called an α-edge (or a β-edge) if i is even (or odd respectively). The mask on α-edge (or β-edge) is known as α-mask (or β-mask respectively). A level uniform masking assignment is illustrated in Fig. 2.7. We also say the mask α_i (or β_i) is α-mask (β-mask respectively) at height $i + 1$.

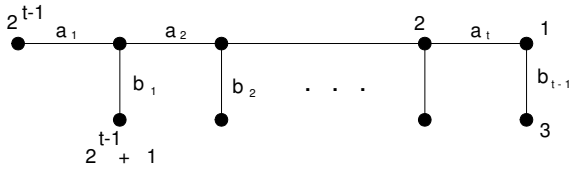

Figure 2.7: α and β functions in a level uniform masking assignment. Here, a denotes the α mask and b denotes the β mask

2.5 Literature Survey

2.5.1 Universal One Way Hash Family

Naor and Yung [74] first constructed a Universal One Way Hash Family or UOWHF based on a one-way permutation. Later, Rompel [87] showed that UOWHF exists under the assumption that one-way function exists. There are many literatures [30, 46] in efficiency of one-way function based universal one-way hash family. Naor and Yung also constructed a signature scheme where a UOWHF is sufficient to prove the security of the signature scheme. Bellare and Rogaway [7] constructed a generic signature scheme based on a Universal One Way Hash Family (UOWHF). It was shown that one-way functions are necessary and sufficient for signatures [87].

To construct a UOWHF on the arbitrary domain we start with a construction of UOWHF with smaller domain from scratch, call it by a compression function, and then extend it to the arbitrary domain. There is also other way to design UOWHF, for example, Impagliazzo *et al.* [40] constructed a UOWHF based on knapsack problem. Natural domain extension method is the MD construction which preserves the collision resistance property [21, 61]. Unfortunately, Bellare and Rogaway [7] showed that the MD construction would not work for UOWHF. They proposed a binary tree based construction with the notion of XOR-ing mask (a part of the key). Sarkar [90] generalized the class of domain extension algorithm $\mathcal{M}$ (see Sect. 2.3.1) which includes all known UOWHF-preserving domain extension algorithms. The domain extension algorithm from the class $\mathcal{M}$ is based on a rooted directed tree (Sect. 2.3.1) and a

masking assignment on it (Sect. 2.4.1 and Sect. 2.3.1). Sarkar [89] designed a binary tree based construction where the number of masks or key size is less than that of construction proposed by Bellare and Rogaway. Their constructions are based on a complete binary tree of height t and level-uniform masking assignments based on the functions α_t and β_t on $[1, t-1]$ as follows;

Bellare-Rogaway [7]: $\alpha_t(i) = i$ and $\beta_t(i) = t + i - 1$. The domain extension based on ψ_t is valid [7]. Here, we need $2(t-1)$ masks (see Fig. 2.8).

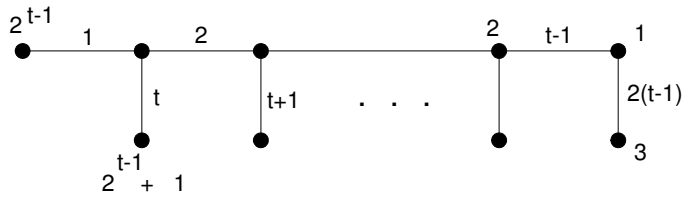

Figure 2.8: The masking assignment used by Bellare and Rogaway [7]

Sarkar [89]: $\alpha_t(i) = i$ and $\beta_t(i) = t + \nu_2(i-1)$, where $\nu_2(i) = k$ if 2^k divides i but 2^{k+1} does not. The domain extension based on ψ_t is valid and number of masks is $t + \lceil \log_2(t-1) \rceil - 1$ (see Fig. 2.9).

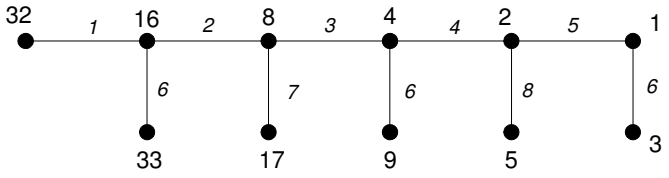

Figure 2.9: The masking assignment used by Sarkar [89] for $t = 6$

Shoup [100] constructed a sequential domain extension where the tree is a sequential tree. The masking assignment of Shoup's construction is $\psi(i) = \nu_2(i)$, $1 \leq i \leq r$. Thus it is the maximum power of two which divides the integer. Here, we need $\lceil \log_2 r \rceil$ masks.

Mironov [64] proved that it is optimum in key size among all sequential constructions. Let us denote the set of all sequential constructions by $\mathcal{S} \subset \mathcal{M}$.

Theorem 2.1 [64] *A mask based valid sequential domain extension should be based on an even-free masking assignment and any even-free masking assignment should contain* $\lceil log_2 r \rceil$ *masks, where* r *is the number of invocations of the base function.*

Thus Shoup's algorithm [100] is optimum in $\mathcal{S}$. Similar study has been made for any tree based domain extension. Sarkar [90] proved that any valid domain extension from $\mathcal{M}$ (Sect. 2.3.1) is based on a even-free masking assignment.

Theorem 2.2 [90] *A valid domain extension based on a directed tree with* t *vertices is based on an even-free masking assignment. Any even-free masking assignment needs at least* $\lceil log_2 s \rceil$ *masks, where* s *is the number of nodes.*

Using this necessary condition, Sarkar [90] proved the lower bound of the number of masks in the class $\mathcal{M}$ given in the above Theorem. In $\mathcal{S}$, both the bounds given by Sarkar and Mironov agree. Lee *et al* [53] constructed an optimum algorithm in the general class $\mathcal{M}$, but parallelism is much smaller than binary tree based algorithm. It is based on an l-dim tree (Sect. 2.2.1) with 2^t vertices. We define two functions, α_t and β_t as follows ((a, b) is defined in Sect. 2.2.1);

- $\alpha_t : [1, 2^a - 1] \rightarrow [1, a]$ is defined by $\alpha_t(i) = 1 + \nu_2(2^a - i)$.
- $\beta_t : [1, 2^b - 1] \rightarrow [a + 1, a + b]$ is defined by $\beta_t(i) = a + 1 + \nu_2(2^b - i)$.

$\psi_t(e_i)$ is defined as follow;

- $\psi_t(e_i) = \alpha_t(j)$ if $2 \leq i \leq 2^a$ and $j = i - 1$.
- $\psi_t(e_i) = \beta_t(j)$ if $2^a < i \leq 2^{a+b}$ and $j2^a < i \leq (j + 1)2^a$.

In case of $l = 2$, the tree and masking assignments are depicted in Fig. 2.5.1. It was shown that the domain extension based on the complete binary tree and the masking assignment ψ_t is valid [53, 54]. Here, we need t masks. Because of the above necessary condition, this construction is optimum (with respect to the number of masks) among all tree based domain extension in $\mathcal{M}$. Also Shoup's construction is optimum in $\mathcal{M}$.

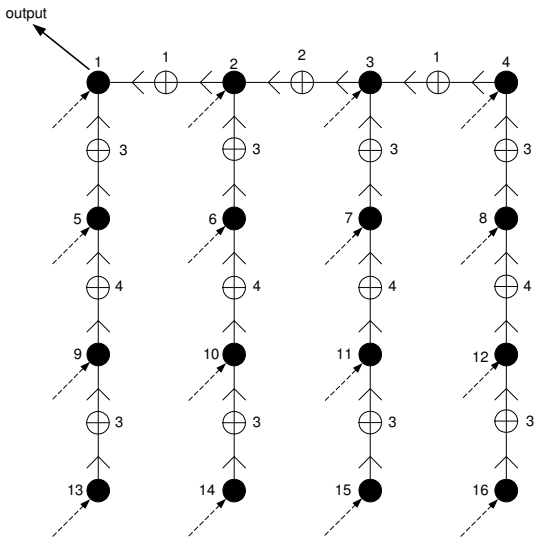

Figure 2.10: The 2-dim Valid domain extension

2.5.2 PGV-Hash Functions

Nowadays many people are designing hash functions based on the structure of block-ciphers, for example, TIGER [1]. There are many constructions based on block-ciphers. For a block cipher $E : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$, Davis-Meyer [80] compression function is $E_x(h) \oplus h$, where $|x| = k$ and $|h| = n$. Miyaguchi [65] and Preneel [80] designed compression function $E_x(h) \oplus x \oplus h$. MMO compression function was proposed by Matyas *et al* [59]. Let $s : \{0,1\}^n \rightarrow \{0,1\}^k$ be a mapping from the ciphertext space to the key space. The MMO compression function is defined as $E_{s(h)}(x) \oplus x$.

B. Preneel, R. Govaerts and J. Vandewalle [80] classified all block-cipher based hash functions. Let $E : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher. We have compression function $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ defined by $f(h,m) = E_a(b) \oplus c$, where $a, b, c \in \{h, m, h \oplus m, v\}$. Here, $|h| = n$, $|m| = n$ and v is some fixed n-bit constant. Note that there are sixty-four compression functions of the above form. These compression functions are known as PGV-compression functions and hash functions based on these compression functions are known as PGV-hash functions. In Chapter 4, we are mainly interested in PGV hash functions. Black, Rogaway and Shrimpton [10] proved that out of sixty four hash functions, only twelve PGV-compression functions and twenty PGV-hash functions are secure with respect to collision and preimage attacks. They

proved the security in the black box model. A parallel version of PGV-hash functions are also studied in [58]. A similar result like sequential PGV hash functions has been proved [58] where. twenty hash functions are proven secure in the black-box model.

In PGV-hash functions, the key of the block cipher depends on intermediate hash values and hence we have to make key-scheduling algorithm in each round. For most efficient construction we can fix keys independent of the message and intermediate hash values. Recently, Black, Cochran and Shrimpton [9] proved that there can not be any highly efficient block-cipher based hash function. They have proved that by using at most two queries one can find a collision of highly efficient block cipher based hash function (where we do not change the key of the block cipher). But this attack is not practical as it ignores the time and space complexity. It only points out some potential weaknesses of the definition.

2.5.3 Multicollision Attack

In a recent paper by Joux [41], it was shown that there is a 2^r-way collision attack for the classical iterated hash function based on a compression function, $f : \{0,1\}^{m+n} \rightarrow \{0,1\}^n$, in complexity $O(r2^{n/2})$, which is much less than $\Omega(2^{\frac{n(2^r-1)}{2^r}})$ (the complexity for birthday attack, see Proposition 2.2). The idea of the attack is to find first r successive collisions (see Figure 2.11) by performing r successive birthday attacks.

$$\begin{gathered} f(h_0, m_1) = f(h_0, n_1) = h_1 \text{ (say) }, \; m_1 \neq n_1 \\ f(h_1, m_2) = f(h_1, n_2) = h_2 \text{ (say) }, \; m_2 \neq n_2 \\ \vdots \\ f(h_{r-1}, m_r) = f(h_{r-1}, n_r) = h_r \text{ (say) }, \; m_r \neq n_r \end{gathered}$$

For $1 \leq i \leq r$, we apply BirthdayAttack$(f(h_{i-1}, \cdot), 2^{n/2}, 2)$ to find $m_i \neq n_i$ such that $f(h_{i-1}, m_i) = f(h_{i-1}, n_i)$. Thus the set $\{x_1 || \cdots || x_r : x_i = m_i \text{ or } n_i, 1 \leq i \leq r\}$ is a 2^r-way collision set and $H(x_1 || \cdots || x_r) = h_r$, where $x_i = n_i$ or m_i. Time complexity of the attack is $O(r2^{n/2})$.

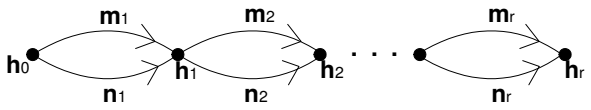

Figure 2.11: Graphical representation of Joux's Multicollision attack

Recently, S. Lucks designed a multicollision secure hash function [57]. He used the notion of wide compression function and twin pipe hash function. It has also been proposed by Finney in a mailing list [28].

2.5.4 Double Length Hash Functions

The most natural and efficient construction of a double length hash function is the concatenated hash function $H||G$, where H and G are two classical n bit hash functions based on the compression function, f, with two different initial values. H and G can be based on two different compression functions f_1 and f_2. Recently, A. Joux [41] showed that there is a collision attack on the concatenated hash function in time complexity $\Theta(n2^{n/2})$. There were several attempts to construct a secure block cipher based double length compression functions. Most of these have attack much better than the birthday attack. Let $E : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher. In the following example, H_i^1, H_i^2, M_i^1 and M_i^2 are n-bits. M_i^1 and M_i^2 represent the message blocks used in i^{th} round and H_i^1 and H_i^2 represent the chaining hash value in the classical iteration. MDC-2 [11] was developed by Brachtl *et al.*

$$T_i^j = E_{H_{i-1}^j}(M_i) \oplus M_i = LT_i^j||RT_i^j \ , \ j = 1, 2.$$

Define, $H_i^1 = LT_i^1||RT_i^2$ and $H_i^2 = LT_i^2||RT_i^1$, where LT and RT represent the left half and right half respectively. There are many other double length block cipher based hash functions, for example,

Parallel-DM [38]:

$$H_i^1 = E_{M_i^1 \oplus M_i^2}(H_{i-1}^1 \oplus M_i^1) \oplus H_{i-1}^1 \oplus M_i^1,$$
$$H_i^2 = E_{M_i^1}(H_{i-1}^2 \oplus M_i^2) \oplus H_{i-1}^2 \oplus M_i^2.$$

The LOKI DBH [13]:

$$H_i^1 = E_{H_{i-1}^2 \oplus M_i^2}(H_{i-1}^2 \oplus M_i^1) \oplus H_{i-1}^1 \oplus H_{i-1}^2 \oplus M_i^1,$$
$$H_i^2 = E_{H_{i-1}^1 \oplus M_i^1}(H_{i-1}^2 \oplus M_i^2 \oplus H_i^1) \oplus H_{i-1}^1 \oplus H_{i-1}^2 \oplus M_i^2.$$

We can generalize the above constructions in the following way. Define $F : \{0,1\}^{4n} \rightarrow \{0,1\}^{2n}$ as follows:

$$F(h^1, h^2, x^1, x^2) = E_{a_1}(b_1) \oplus c_1 || E_{a_2}(b_2) \oplus c_2,$$

where a_i, b_i and c_i's are linear combinations of h^1, h^2, x^1 and x^2. In [48], it was shown that most of the hash functions based on these compression functions, have collision attack with complexity $O(2^{n/2})$. The remaining constructions have collision attack with complexity $O(2^{3n/4})$. If we use a block cipher $E : \{0,1\}^n \times \{0,1\}^{2n} \rightarrow \{0,1\}^n$, then double length hash functions based on compression functions of the form

$$F(h^1, h^2, x^1, x^2) = E_{a_1||a_1'}(b_1) \oplus c_1 || E_{a_2||a_2'}(b_2) \oplus c_2,$$

also have collision attack with complexity $O(2^{3n/4})$ (see [48, 93]). Thus almost all efficient block cipher based double length hash functions are not secure. Knudesn and Preneel [49] designed multiple length hash functions based on error-correcting codes. The efficiency of their construction is less than that of all the above constructions. But in this paper we have a concrete security proof. S. Lucks [57, 28] deigned a double length compression function, $F(h', h', x) = f(h', h'', x)||f(h'', h', x)$, where $|h'| = |h''| = n$ and $|x| = m - n$. He designed this compression function to construct a multicollision secure hash function. But it can be proved that the compression function is maximally secure in the random oracle model. Almost same design was proposed by S. Hirose [37]. His construction is based on a secure block cipher $E : \{0,1\}^n \times \{0,1\}^{2n} \rightarrow \{0,1\}^n$.

2.6 Our Contribution

In this section we briefly describe our main contributions. We only state the main theorems. Detail discussion of these theorems, proofs and explanation of terminologies can be found in the appropriate chapters.

In this thesis we provide a sufficient condition for a valid or UOWHF-preserving domain extension from the class $\mathcal{M}$. In Sect. 3.2 we prove that the domain extension algorithm

is valid if the underlying masking assignment is strongly even-free (see Definition 2.8). More precisely, let $\mathcal{A}$ be an algorithm based on the structure (T, ψ), and $\mathcal{A} : \{h_k\} \rightarrow \{H_p\}$ then we have,

Theorem (as in Theorem 3.1) *For any (ϵ, t)-UOWHF $\{h_k\}$, the extended hash family $\{H_p\}$ is (ϵ', t')-UOWHF, where ϵ' and t' are constant multiple of ϵ and t. Thus, a domain extension algorithm based on a strongly even-free masking assignment ψ on T is a valid domain extension.*

Theorem (as in Theorem 3.2) *The domain extension algorithms [7, 53, 89, 100] presented in Sect. 2.5.1 are valid domain extensions. In fact, all these domain extension algorithms are based on strongly even-free masking assignments.*

In this thesis we propose two new methods of domain extension which are UOWHF-preserving or valid. The first one is known as **IBTC** or Improved Binary Tree based Construction and it is based on a complete binary tree of height t and a masking assignment ψ_t described in Sect. 3.3.1.

Theorem (as in Theorem 3.4) *The domain extension algorithm IBTC of height t is a valid domain extension. The key expansion of the algorithm is $m(t+ O(\log_2^* t))$.*

The second construction, **OPT**, is based on a binary tree with number of vertices 2^t and height t using t many masks. In Sect. 3.3.2 we prove the following Theorem :

Theorem (same with corollary of Theorem 3.5) *There exists a binary tree T with 2^t many vertices of height $t+1$ and there is a strongly even-free t-masking assignment on T. Thus the domain extension algorithm based on this masking assignment is valid and optimum in both key size and the number of parallel rounds.*

In Chapter 4, we study the security properties of generalized PGV-hash families (see Definition in Sect. 4.2). We show the following (target) collision bounds and inversion bounds (see Table 2.1).

In Chapter 5, we first generalize the class of classical hash functions and also the parallel hash functions. We prove the following theorems:

Theorem (as in Theorem 5.1) *Let H be a hash function based on the sequence $\langle \alpha^1, \alpha^2, \cdots \rangle$ with $\text{freq}(\alpha^l) \leq 2$ for every $l \geq 1$. Then we have a 2^r-way multicolli-*

Table 2.1: The case $l > 0$ is analyzed in this thesis.

$l > 0$	(T)Coll Bound	Inversion Bound
E1 (20 schemes)	$\Theta(q^2/2^n)$	$\Theta(q/2^l)$ or $\Theta(q/2^n)$ or $\Theta(q^2/2^n)$
E2 (4 schemes)	$\Theta(q/2^l)$	$\Theta(q/2^l)$
E3/E4/E5 (18 schemes)	$\Theta(q^2/2^l)$	$\Theta(q/2^l)$ or $\Theta(q^2/2^l)$ or $\Theta(q/2^n)$
E6 (22 schemes)	$\Theta(1)$	–

sion of H with the complexity $\mathrm{O}(r^2 n 2^{n/2})$.

Theorem (as in Theorem 5.3) *If* $\mathrm{freq}(G(H)) \leq 2$ *then we have a* 2^r*-way multicollision on tree based hash function with the complexity* $\mathrm{O}(r^2 n.2^{n/2})$.

In Chapter 6 we propose several double length hash functions. We first propose a class of double length hash functions based on permutations and then prove the security of compression functions f^p and hash functions H^{f^p} based on permutation in Sect. 6.3 and in Sect. 6.3.3.

Theorem (same with Theorem 6.1) *For any permutation p where $\mathcal{F}_p$ is the empty set, any attack algorithm finding collision of f^p requires $\Omega(2^n)$ many queries of f in the random oracle model.*

Theorem (same with Theorem 6.2) *The classical hash function, H^{f^p}, based on a good permutation and an initial value $H_0 \notin \mathcal{F}_p[2n]$ has collision security $\Omega(2^n)$ in the random oracle model.*

Here, $\mathcal{F}_p$ denotes the the set of fixed points of p, i.e. $\{X : p(X) = X\}$. The definition of "good permutation" is given in Definition 6.3.

Then we propose a rate 1/3 double length compression function, say F, based on three independent compression functions and we prove the following security property in Sect. 6.4.

Theorem (given in Sect. 6.4) *The compression function F has collision complexity $\Omega(2^{2n/3})$ and preimage complexity $\Omega(2^{4n/3})$.*

Finally, we propose a rate close to 1/2 double length compression function F based on two independent random functions f_1 and f_2 in Sect. 6.5. We have the security bound

as follows;

Theorem (same with Theorem 6.5) *If f_1 and f_2 are two independent random functions then the complexity for finding a collision on $F^{(i)}$ requires $\Omega(2^n/(in^{i-1}))$ queries.*

Chapter 3

Universal One-Way Hash Family

3.1 Introduction

Universal One-Way Hash Family or UOWHF is another name of Target Collision Resistant Hash Family (see Sect. 2.4). Naor and Yung [74] first introduced it and gave an application to a signature scheme. Later, Bellare and Rogaway [7] renamed it to Target Collision Resistant. Bellare and Rogaway [7] showed that the classical iteration is not UOWHF-preserving. They introduced a notion of mask based domain extension. Since then, there have been many literatures regarding UOWHF-preserving mask based domain extensions.

Organization of The Chapter

In this chapter, a non-trivial sufficient condition for UOWHF-preserving domain extension is being provided. We first prove that a domain extension algorithm based on a masking assignment ψ on a rooted directed tree is UOWHF-preserving if the masking assignment is strongly even-free (see Definition 2.8). Next we show that all known valid domain extension algorithms satisfy this sufficient condition. Then we design a valid Improved Binary Tree based Construction or IBTC. This is a reasonable improvement over the best known domain extension [89]. Next we design OPT or optimal valid domain extension based on a tree which uses optimum number of masks and almost optimum number of parallel rounds. Finally, we prove that the IBTC is also optimum among a wide subclass of complete binary tree based domain extensions.

3.2 Sufficient Condition for a Valid Domain Extension

In this section we prove that if a domain extension algorithm with structure $\mathbf{A} = (T, \psi)$ (see Sect. 2.3.1) is based on a strongly even-free masking assignment ψ, then it is valid. Now we briefly recall mask based domain extension.

Let $\mathbf{A} = (T, \psi)$ be a structure of an algorithm, where

- $T = (V, E, q)$ is a rooted directed tree with the root q and $|V| = r$,
- ψ is an l-masking assignment on T.

Let $M = M_V = M_{v_1}||\cdots||M_{v_r}$ where $\prec$ is some fixed totally order on V such that $v_1 \prec \cdots \prec v_r$. Now, we define a domain extension $\mathbf{A} : (n+m, n, K) \to (n+rm, n, K+nl)$. Thus, given a $(n+m, n, K)$-hash family $\{f_k\}$, $k \in K$ and $p = k||\mu_1||\cdots||\mu_l$, we define $(n+rm, n)$-hash function H_p where $|\mu_i| = n$, $1 \le i \le l$.

- If $v \in L$, define $h_v = f_k(M_v)$,
- otherwise $h_v = f_k(z_{son(v)}, M_v)$, where $z_v = h_v \oplus \mu_{\psi_v}$.
- Define $H_p(M) = h_q$.

If $v \in L$ then define $y_v = M_v$, otherwise $y_v = z_{son(v)}||M_v$. That is, y_v is the input of f_k placed at node v. Note that h_v, z_v and y_v depends on the message M, the underlying directed tree T, the masking assignment ψ and the key p. Recall that an adversary $\mathcal{A}$ in TColl game runs in two stages, namely $\mathcal{A}^{\text{find}}$ and $\mathcal{A}^{\text{guess}}$.

Theorem 3.1 *If a domain extension algorithm is based on a strongly even-free masking assignment ψ on T then it is a valid domain extension. More precisely, for any (ϵ, t)-strategy for $\{H_p\}$ there is an (ϵ', t')-strategy for $\{f_k\}$ where $\epsilon' = \epsilon/r$ and $t' = t + O(r)$, where $r = |V|$.*

Proof. Let $\mathcal{A}$ be an adversary with runtime at most t and success probability at least ϵ for $\{H_p\}$. Now we define an adversary $\mathcal{B}$ for $\{f_k\}$ with runtime at most t' and success probability at least ϵ', where ϵ' and t' are constant multiples of ϵ and t respectively. .

$\mathcal{B}^{\textbf{guess}}$ **:**

1. $(M_V, s') \leftarrow \mathcal{A}^{\text{guess}}$.

2. Choose $v \in_R V$ (v is chosen randomly from V). Let $\text{indeg}(v) = d$. Choose a random string R, $|R| = dn$.

3. Let $y = R||M_v$ and $s = (s', v, y)$. $\mathcal{B}^{\text{guess}} \rightarrow (y, s)$.

At this point the adversary is given a k which is chosen uniformly at random from the set $\mathcal{K} = \{0,1\}^K$. The adversary then runs $\mathcal{B}^{\text{find}}$ which is described below.

$\mathcal{B}^{\textbf{find}}(y, k, s)$ **:** (Note $s = (s', v, y)$)

1. $\mu_1||\ldots||\mu_l \leftarrow \text{MDef}(M, k, v, R, T, \psi)$ (We describe the algorithm MDef shortly after this proof).

2. $M' \leftarrow \mathcal{A}^{\text{find}}(M, p, s')$ where $p = k||\mu_1||\ldots||\mu_l$. Let y' be the input to processor at node v while computing $H_p(M')$ i.e. $y' = y'_v$.

3. $\mathcal{B}^{\text{find}} \rightarrow y'$.

Now we prove the following Lemma to prove the rest of the Theorem.

Lemma 3.1 *The above mask defining algorithm* MDef *satisfies the following :*

1. *The masks (output of the algorithm) $\mu_1, \cdots, \mu_l$ are uniformly distributed provided R is uniformly distributed.*

2. *If* $\text{MDef}(M, k, v, R, T, \psi) = \mu_1||\cdots||\mu_l$ *then the input of f_k at node v is $R||M_v$ while computing $H_p(M)$, where $p = k||\mu_1||\cdots||\mu_l$.*

Proof of the Lemma 3.1 : We can check easily that all the masks are random strings as either they are random strings or they are XOR of two strings one of which is a random string (see step-5 in the algorithm). So, we can prove the first part of the Lemma by using induction on size of the tree. The second part of the Lemma is also straightforward by induction on size of tree. □

Thus, by Lemma 3.1, input of node v while computing $H_p(M)$ will be y which is already committed in $\mathcal{B}^{\text{guess}}$. Also note that p is a randomly chosen key from the set $\mathcal{P}$ as both k and μ_i's are random strings. We now provide lower bound of the winning probability. Suppose M and M' collide for the function H_p. Then there must be a $i \in V$ such that at vertex i there is a collision for the function f_k. (Otherwise it is possible to prove by a backward induction that $M = M'$.) The probability that $i = v$ is $\frac{1}{|V|}$. Hence, if the winning probability of $\mathcal{A}$ is at least ϵ, then the winning probability of $\mathcal{B}$ is at least $\frac{\epsilon}{|V|}$, since two events $\{i = v\}$ and $\{\mathcal{A} \text{ wins}\}$ are independent. Also the number of invocation of f_k by $\mathcal{B}$ is equal to the number of invocation of f_k by $\mathcal{A}$ plus at most $2|V|$. The number $2|V|$ is coming from the fact that in MDef algorithm we need at most $|V|$ many invocations and we may need at most $|V|$ many invocation of f_k again to compute y'. □

Now we state the algorithm MDef and prove Lemma 3.1. Let $R = R_{son(v)}$ where $v_1 \prec \cdots \prec v_d$ be sons of the node v and $|R_{v_i}| = n$, $1 \leq i \leq d$.

Algorithm MDef(M, k, v, R, T, ψ)

1. Suppose $\psi(u) = l$ (say) is a single man for the subtree $T[v]$. Let $T_1 = T \setminus T[u]$. Obviously, $T_1 \neq T$ (in fact, $|T_1| < |T|$). Note that $u \neq v$. Let u' be the father of u and without loss of generality $son(u') = \{u_1, \cdots u_s\}$, where $u_1 \prec \cdots \prec u_s = u$.

2. Let ψ_1 be the masking assignment ψ restricted on T_1. Note that ψ_1 is an l'-masking assignment. So, $l' \leq l - 1$.

3. If $v \neq u'$, then define r as an n-bit random string, otherwise $r = R_u$. Define a message M' such that $M'_{u'} = r||M_{u'}$ and $M'_w = M_w$ for other node w in T_1.

4. If $u' = v$ then define $R_1 = R_{u_1} \parallel \cdots \parallel R_{u_{s-1}}$. Otherwise $R_1 = R$.

5. Run MDef$(M', k, v, R_1, T_1, \psi_1)$ to get $\mu_1, \cdots, \mu_{l'}$. Define all other masks except μ_l randomly. Define $\mu_l = r \oplus h'_u$, where h'_u represents the h value at node u while computing $H(M)$ based on the tree $T[u]$.

Remark 3.1 *In the proof of the above theorem we assume that u is a single man in the subtree $T[v]$ where u is the last son of its father (see step-1 in Algorithm MDef). One*

can remove this assumption by considering a more general class of domain extension. In that class we consider a permutation on the input of the function f_k placed at node v for all $v \in V$. So when we recursively call MDef, we only need to redefine the permutation at node u.

3.2.1 "Valid" Property of Known Constructions

Now we can use the above Theorem to prove the UOWHF-preserving property for the domain extension algorithms presented in Sect. 2.5.1. In fact, one can check that all other valid domain extensions are based on strongly even-free masking assignments.

Theorem 3.2 *The domain extension algorithms [7, 53, 89, 100] presented in Sect. 2.5.1 are valid domain extensions. In fact, all these domain extension algorithms are based on strongly even-free masking assignments.*

Proof. Because of Theorem 3.1, we only need to prove that all these domain extensions are based on strongly even-free masking assignments.

(1) **(Shoup [100]):** Any nontrivial sub-tree of a sequential tree is an interval $[a, b]$, where $1 \leq a < b \leq r$. Now there exists unique $c \in [a, b-1]$ such that $\nu_2(c)$ is maximum on that interval as there can not be more than one c's with maximum $\nu_2(c)$. If not, there are two integers $x = 2^i r_1$ and $y = 2^i r_2$, where $r_1 < r_2$ are odd integers and both x and y lie on the interval $[a, b-1]$. Then $2^i(r_1+1)$ lies on the same interval and $\nu_2(2^i(r_1+1)) \geq i+1 > \nu_2(x)$. Thus, $j = \nu_2(c) = \psi(c)$ is a single-man for the sub-tree $[a, b]$. So, it is a strongly even-free masking assignment.

(2) **(Bellare-Rogaway [7]) :** Let $T' = (V', E')$ be any sub-tree. Let i be the vertex with minimum level in T'. Again, there can not be more than one such i as the subtree is connected. Now either $\psi(2i)$ or $\psi(2i+1)$ is a single-man for T'.

(3) **(Sarkar [89]) :** Let $T' = (V', E')$ be any sub-tree. Consider the α edge $(2i, i)$ in E' so that i is minimum. If there is one such i then $\psi(2i)$ is a single-man. So, assume there is no α edge in E'. Then T' is a sequential tree consisting of β edges. But along β edges the masking assignment is same as that of Shoup's. So, in that case we also have a single man.

(4)**(Lee et al** $(l-dim)$**) [53] :** We prove it for 2-ary tree. The other cases follow similarly. Let $T' = (V', E')$ be any sub-tree. Note that, $[1, 2^a] \cap V'$ is an interval, say $[c, d]$. If $d > c$, then from the definition of the masking assignment it is clear that masking assignment on $[1, 2^a]$ is same as Shoup's assignment which is strongly even-free. Also note that the masks used on $[1, 2^a]$ are totally different with the masks used in other parts. So the single man on $[c, d]$ is also single man of T'. Now if $c = d$ or $[1, 2^a] \cap V' = \phi$, then T' is a sequential sub-tree of the tree induced by the vertices $\{i, i + 2^a, \ldots, 2^a(2^b - 1) + i\}$ for some $2 \leq i \leq 2^a$. Again along this tree the masking assignment is determined by β_t which is strongly even-free as it is same as Shoup's assignment. So the masking assignment is strongly even-free. □

3.3 Two New Efficient Valid Domain Extensions

3.3.1 Improved Binary Tree based Construction (IBTC)

Define a sequence $\{t_k\}_{k\geq 0}$ recursively as follows:

$$t_{k+1} = 2^{t_k+k} + t_k, \text{ where } t_0 = 2 \text{ and } k \geq 0$$

It is easy to check that, $t_0 = 2 < t_1 < \cdots < t_{k-1} < t_k < \cdots$. We define another sequence $\{m_t\}_{t\geq 2}$ based on $\{t_k\}_{k\geq 0}$ as follows;

$$m_2 = 2, \ m_t = t + k, \text{ if } t_{k-1} < t \leq t_k, t > 2.$$

Proposition 3.1 *$m_{t+1} = m_t + 2$, when $t = t_k$ for some k and $m_{t+1} = m_t + 1$ when $t_k < t < t_{k+1}$ for some k.*

Proof. When $t = t_k$, $m_t = t + k$ and $m_{t+1} = (t+1)+(k+1)$, since $t+1 \in [t_k+1, t_{k+1}]$. Thus, $m_{t+1} = m_t + 2$. Similarly, $m_{t+1} = m_t + 1$, when $t_k < t < t_{k+1}$. □

Definition 3.1 *For $x \geq 1$, define $\log_2^* x = k$, if $\log_2^{(k)}(x) \leq 1$ but, $\log_2^{(k-1)}(x) > 1$, $k \geq 1$. Here, $f^{(k)}(x)$ means the the function f applies k many times on x. $f^{(0)}(x) = x$.*

Proposition 3.2 *$m_t \leq t + \log_2^* t$, $t \geq 2$.*

Proof. For $t = 2$, proof of the statement is trivial. For $t > 2$, $m_t = t + k$, where $t \in [t_{k-1} + 1, t_k]$ and $k \geq 1$. Note that, $\log t_{k+1} > t_k$. Thus, $\log^* t_k \geq \log^* t_{k-1} + 1$ and so on. By continuing, we get $\log^* t_k \geq \log^* t_0 + k = k + 1$. Since $t > t_{k-1}$, we have $\log^* t \geq \log^* t_{k-1} \geq k$. So, $m_t = t + k \leq t + \log_2^* t$. □

We define a level uniform masking assignment on a complete binary tree of height $t \geq 2$ by using m_t many masks. Recall that, a level uniform masking assignment is completely determined by two functions $\alpha_t, \beta_t : [1, t-1] \to [1, m_t]$, $t \geq 2$. The masking assignment of IBTC is determined by the following two functions α_t and β_t which are defined recursively.

1. $\alpha_2(1) = 2$ and $\beta_2(1) = 1$.
2. For $t \geq 3$,
   (a) $\alpha_t(i) = \alpha_{t-1}(i)$ and $\beta_t(i) = \beta_{t-1}(i)$ whenever $1 \leq i \leq t-2$.
   (b) If $t = t_k + 1$ for some k then $\alpha_t(t-1) = \alpha_{t-1}(t-2) + 2$ and $\beta_t(t-1) = \alpha_{t-1}(t-2) + 1$ and if $t_k < t - 1 < t_{k+1}$ then $\alpha_t(t-1) = \alpha_{t-1}(t-2) + 1$ and $\beta_t(t-1) = \nu_2(t - 1 - t_k) + 1$.

For $t = 6$, the masking assignment is depicted in Fig. 3.1.

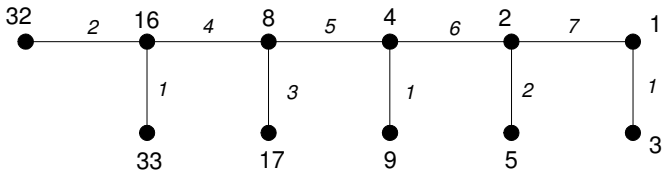

Figure 3.1: The masking assignment used in IBTC for $t = 6$

Proposition 3.3 *$\alpha_t([1, t-1]) \cup \beta_t([1, t-1]) = [1, m_t]$. For a subtree rooted at height $t_k + 1$, the masks of both α and β-edge at the height $t_k + 1$ are single man. For other height, t, only the mask of α-edge at height t is a single man.*

Proof. We first prove by induction that $\alpha_t(t-1) = m_t$, $t \geq 2$. We have, $\alpha_2(1) = m_2 = 2$. Now if $t = t_k + 1$, then $\alpha_t(t-1) = \alpha_t(t-2) + 2 = m_{t-1} + 2 = m_t$. Otherwise, $\alpha_t(t-1) = \alpha_t(t-2) + 1 = m_{t-1} + 1 = m_t$.

The first part of the proposition can be proved by using induction. The statement can be proved trivially for $t = 2$. Let $\alpha_t([1, t-2]) \cup \beta_t([1, t-2]) = [1, m_{t-1}]$. If $t - 1 = t_k$ then $m_t = m_{t-1} + 2$ and $\beta_t(t-1) = m_{t-1} + 1$ and $\alpha_t(t-1) = m_{t-1} + 2$. Otherwise, $m_t = m_{t-1} + 1$ and $\beta_t(t-1) < m_{t-1} + 1$ and $\alpha_t(t-1) = m_{t-1} + 1$. So, $\alpha_t([1, t-1]) \cup \beta_t([1, t-1]) = [1, m_t]$.

The second part is immediate from the definitions of α_t and β_t. □

Theorem 3.3 *The masking assignment ψ_t on a complete binary tree of height t, based on α_t and β_t defined as above, is a strongly even-free masking assignment.*

Proof. Let S be a sub-tree with root v, $l(v) = t'$. Let $t_{k+1} \geq t' \geq t_k + 1$. If there is an α-edge in S between height t_{k+1} to $t_k + 1$ then the mask of first such one (i.e. the α edge at maximum height) will be a single-man and we are done. So S can contain at most one β edge at height $t_k + 1$. If it contains that, then $\beta_t(t - t_k + 1)$ is a single-man for S. If S does not contain that β edge at height $t_k + 1$, then again it is a sequential sub-tree consists of only β-edges from height t' to at least height $t_k + 2$. But on that tree, masking assignment is defined by ν_2 function which is itself strongly even-free masking assignment. So, the above masking assignment is strongly even-free. □

Theorem 3.4 *The domain extension algorithm based on a complete binary tree of height t and the masking assignment ψ_t is a valid domain extension. The key expansion of the algorithm is $m(t+ O(\log_2^* t))$.*

Proof. This is an immediate corollary of Theorem 3.1, 3.3 and Proposition 3.3. □

3.3.2 An optimal binary tree based construction (OPT)

We know that if ψ is even-free or strongly even-free m-masking assignment, then $2^m \geq |V|$ [90]. An m-masking assignment ψ on $T = (V, E)$ is called optimal masking assignment if it is strongly even-free and $2^m \geq |V| > 2^{m-1}$. So, an optimal masking assignment is a strongly even-free masking assignment whose number of masks is

minimum possible. One such example is given by Shoup's [100] sequential construction. We say Shoup's masking assignment on n-sequential path n-sequential optimal masking assignment.

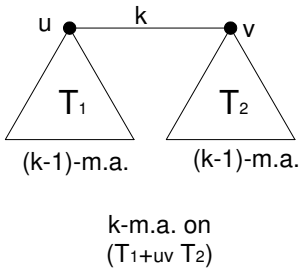

Figure 3.2: Concatenation of two masking assignments

Suppose ψ_i is a k-masking assignment on T_i for $i = 1, 2$. Then we can define ψ a $(k+1)$-masking assignment on $T_1 +_{uv} T_2$, where ψ on T_i is same as ψ_i on T_i and $\psi(uv) = k+1$ (see Fig. 3.2). We denote ψ as $\psi_1 +_{uv} \psi_2$ or $\psi_1 + \psi_2$. If both ψ_1 and ψ_2 are strongly even-free then so is ψ. It is easy to check that if both ψ_1 and ψ_2 are optimal then so is $\psi = \psi_1 + \psi_2$.

Definition 3.2 *An m-masking assignment is called* (m, l, i)**-optimal masking assignment** *if it is optimal masking assignment on a i-λ tree T such that $ht(T) = l$ and $|V| = 2^m$.*

Consider a set of i-λ tree T with $ht(T) = l$ and the number of nodes is 2^m. Call this class of trees by $\mathcal{T}$. If an (m, l, i)-optimal masking assignment exists then that masking assignment, say ψ, is optimum masking assignment among all masking assignments of valid domain extensions based on the class of trees $\mathcal{T}$. Because, for any even-free m-masking assignment the number of nodes is at most 2^m and even-free masking assignment is necessary for valid domain extension. In particular if we have $i = 1$ and $l = m + 1$ then the masking assignment ψ is optimum among all binary tree based valid domain extension where the height of the tree is $m+1$. But any binary tree with number of vertices 2^m must have height at least $m + 1$. Thus, in terms of the height

of the tree (or the number of rounds of the algorithm) the domain extension based on ψ is also optimum.

Now we illustrate some examples of optimal masking assignments. In Fig 3.3, we have $(3,4,1)$, $(3,5,2)$ and $(3,6,3)$-optimal masking assignments.

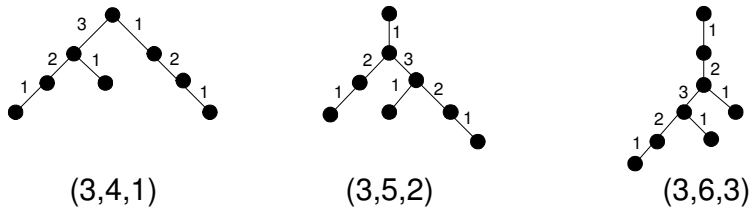

Figure 3.3: some optimal masking assignments (the numbers besides edges denote the values of masking assignment).

Theorem 3.5 *There exists an $(n, n+i, i)$-optimal masking assignment if $i \leq 2^n$.*

Proof. Let $f(k) = 2^k + k + 1$ for $k \geq 0$. It is strictly increasing function, So, given positive integers n and i there exists a unique $k \geq 1$ such that $f(k) > (n+i) \geq f(k-1)$. We prove the Theorem by induction on $n+i$. For small values of $n+i$ we have shown some examples in Fig. 3.3. Now given n and i we assume that the Theorem is true for any i_1 and n_1 such that $i_1 \leq 2^{n_1}$ and $i_1 + n_1 < i + n$. Choose k as above for these n and i.

First assume that, $f(k) > (n+i) > f(k-1)$. Let $j = (n+i) - f(k-1) \geq 1$. By induction hypothesis there is a $(k-1, k-1+j, j)$-optimal masking assignment ($2^{k-1} \geq j$ as $f(k) > n+i$). Call this by ψ_{k-1}. Now, ψ_{k-1} is a masking assignment on a j-binary tree $T_{k-1} = (V_{k-1}, E_{k-1}, v_1)$, where $\{v_1, \ldots, v_j\}$ is the i-path. Now take the sequential 2^{k-1}-optimal masking assignment ψ on $T = (V, E)$ and define $\psi_k = \psi_{k-1} +_{v_j v_{j+1}} \psi$, where $V = \{v_{j+1}, \ldots v_{j+2^{k-1}}\}$. Now we can add optimal masking assignment one by one with ψ_k at v_i's. More precisely, let ψ'_l be a $(l-1, 1, l)$-optimal masking assignment on $T_l = (V_l, E_l, u_l)$ for $k+1 \leq l \leq n$. Define $\psi_l = \psi_{l-1} +_{v_{l-k+1} u_l} \psi'_l$ recursively for $k+1 \leq l \leq n$. Now it can be checked easily that ψ_n is $(n, n+i, i)$-optimal masking assignment. This is explained diagrammatically in Fig. 3.4.

Now assume that, $n+i = f(k)$ for some k. In this case construct a 2^k-sequential optimal masking assignment ψ on a sequential tree $T_k = (V_k, E_k)$, where $V_k = \{v_1, \ldots, v_{2^k}\}$. For each l, $k \leq l \leq n-1$ we have $(l, l+1, 1)$-optimal masking assignment ψ_l. We can concatenate ψ_l with ψ one by one. Finally, we have $(n, n+i, i)$-optimal masking assignment. □

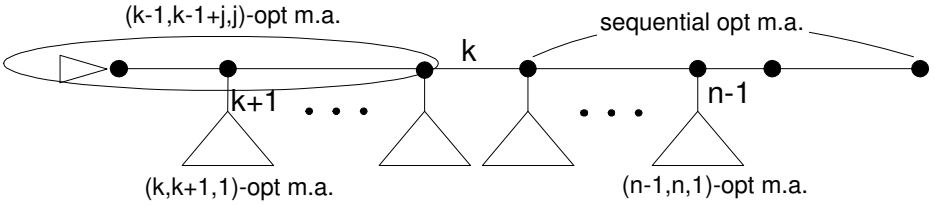

Figure 3.4: Construction of $(n, n+i, i)$-optimal masking assignment

One immediate corollary is given below which tells that we have a domain extension algorithm which needs t many masks and $t+1$ many rounds for 2^t many invocation of base hash function. Note that both number of rounds and number of keys are minimum possible.

Corollary 3.1 *There exists an $(m, m+1, 1)$-optimal masking assignment i.e. there exists a binary tree T of size 2^m with $ht(T) = m+1$ which is minimum possible (a complete binary tree of level m has size $2^m - 1$) so that an optimal masking assignment ψ on T exists.*

3.4 Optimality of IBTC

We fix a $T_t = (V_t, E_t)$, a full binary tree of height t and a level uniform masking assignment (see Definition 2.9) ψ on it which is based on two functions α_t and β_t on $[1, t-1]$. From now onwards, we write α_i or β_i instead of $\alpha_t(i-1)$ or $\beta_t(i-1)$ respectively, $2 \leq i \leq t$. Thus if v is a vertex at height $i \geq 2$ then $\psi(2v) = \alpha_i$ and $\psi(2v+1) = \beta_i$ (see Fig. 3.5).

Table 3.1: Specific comparison of domain extenders for UOWHF 1:seq/par, 2:message length, 3:number of invocation of f_k, 4:number of masks, 5:number of rounds, 6:speed-up, 7:rank in parallelism, 8:rank in key expansion

Param	Shoup [101]	l-DIM($l \geq 2$) [53]	Sarkar[89]	**IBTC**[67]	**OPT** [69]
1	sequential	parallel	parallel	parallel	parallel
2	$2^t n$ $-(2^t-1)m$	$2^t n$ $-(2^t-1)m$	$2^t n$ $-(2^t-1)m$	$(2^t-1)n$ $-(2^t-2)m$	$2^t n$ $-(2^t-1)m$
3	2^t	2^t	2^t-1	2^t-1	2^t
4	t	t	t+ $O(\log_2 t)$	t+ $O(\log_2^* t)$	t
5	2^t	$l2^{t/l}-l+1 (t \equiv 0 \bmod l)$	t	t	$t+1$
6	1	$\frac{2^t}{l2^{t/l}-l+1} (t \equiv 0 \bmod l)$	$\frac{2^t}{t+1}$	$\frac{2^t}{t+1}$	$\frac{2^t}{t+1}$
7	3	2	1	1	1
8	1	1	3	2	1

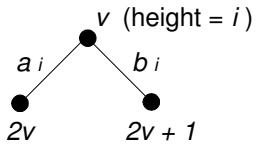

Figure 3.5: Definition of α_i (or a_i) and β_i (or b_i)

Let $M_1 = \emptyset$. For $t \geq j \geq i \geq 1$, define $M[i] = \{\alpha_i, \beta_i\}$ and $M[i,j] = \bigcup_{k=i}^{j} M[k]$. Define $\text{New}(i) = M[i] \setminus M[1, i-1]$, $i \geq 2$. $k \in \text{New}(i)$ is a new mask rooted at height i.

Let $m'_i = |M[2,i]|$, the number of masks used in the full binary sub-tree rooted at vertex of height i.

For any non-trivial sub-tree S, $\text{supp}(S) = \{i : vec_\psi(S)_i \text{ is odd } \} \subset [1,l]$ where $vec_\psi(S)_i$ = the number of times the mask i appears in the multi-set $\psi(S)$. In Fig. 2.6, $\text{supp}(T) = \{1,5\}$, where T is the complete tree. Note, if ψ is even-free (See Definition 2.8) then there is no non-trivial sub-tree S with empty $\text{supp}(S)$.

For each v define, $S(v) = \{A : A = \text{supp}(S)$ and S is a nontrivial sub-tree with root $v\}$. Define, $N(v) = |S(v)|$. If ψ is level-uniform then $S(v_1) = S(v_2)$ whenever $ht(v_1) = ht(v_2)$ and hence we write N_i instead of $N(v)$, where $i = ht(v)$.

Lemma 3.2 *Suppose ψ is any level-uniform masking assignment on a complete binary tree T_t. Then, for $i \geq 2$ and any subset $A \subset M[2,i]$ containing* New(i) *and for any $v \in V$ with height i, $A \in S(v)$.*

Proof. We prove the lemma by induction on i. For $i = 2$, $M[2,2] = \text{New}(2)$. Hence for any $A \supset \text{New}(2)$, we can always find a subtree S rooted at vertex of height 2 such that $\text{supp}(S) = A$.

Assume the result is true for $i-1 \geq 2$. Let v_1 and v_2 be two sons of v such that (v_1, v) is the α-edge and (v_2, v) is the β-edge. We have a subtree S_2 rooted at v_2 so that $\text{supp}(S_2) = \text{New}(i-1) \setminus A$. We consider three cases depending on the set New(i).

- Case(1) : Let $\text{New}(i) = \emptyset$ and $\alpha_i \in A$ but $\beta_i \notin A$ and $\alpha_i \notin \text{New}(i-1)$. $A' = (A \cup \{\beta_i\} \cup \text{New}(i-1)) \setminus \{\alpha_i\}$ contain New$(i-1)$ and hence by induction hypothesis there exists a subtree S_1 rooted at v_1 such that $\text{supp}(S_1) = A'$. Thus, $\text{supp}(S) = A$, where $S = S_1 \cup \{(v_1, v)\} \cup \{v_2, v\} \cup S_2$. Similarly we can prove the other cases.

- Case(2) : Let $\text{New}(i) = \{\alpha_i\}$. First assume that $\beta_i \notin A$ and $\beta_i \notin \text{New}(i-1)$. Let $A_1 = A \setminus \{\alpha_i\}$. Define $A' = A_1 \cup \text{New}(i-1) \cup \{\beta_i\}$. By induction hypothesis there is a subtree S_1 rooted at v_1 so that $\text{supp}(S_1) = A'$. Let $S = S_1 \cup \{(v_1, v)\} \cup \{v_2, v\} \cup S_2$, then $\text{supp}(S) = A$. If $\beta_i \in A$ and $\beta_i \notin \text{New}(i-1)$, then define $A' = (A_1 \cup \text{New}(i-1)) \setminus \beta_i$. By similar argument $\text{supp}(S) = A$. One can define A' for other two cases.

- Case(3) : Let $\text{New}(i) = \{\alpha_i, \beta_i\}$. Let $A_1 = A \setminus \text{New}(i)$. Then define $A' = A_1 \cup \text{New}(i-1)$. Now by induction hypothesis there exists a subtree S_1 rooted at v_1 so that $\text{supp}(S_1) = A'$. Then it is easy to check that $\text{supp}(S) = A$, where $S = S_1 \cup \{(v_1, v)\} \cup \{v_2, v\} \cup S_2$. □

In particular, if ψ is even-free then at each height there is a new mask. Otherwise we have a nontrivial tree whose support is empty which contradicts the assumption that ψ is even-free. Using the above Lemma we can get the following recursive inequality on N_i's.

Lemma 3.3 *If ψ is level-uniform even-free masking assignment, then for each i, $N_{i+1} = m(N_i + 1) + 2^{m'_i}$ when $|\text{New}(i)| = 1$ and $N_{i+1} = 3 \times 2^{m'_i}$ when $|\text{New}(i)| = 2$.*

Proof. When $\text{New}(i+1) = \{k\}$ for any subset A of $[1, m'_i]$, we have a non trivial sub-tree rooted at v of height $i+1$ such that $\text{supp}(S) = A \cup \{k\}$. Also there are $N_i + 1$ sets in $S(v)$ which does not contain k (by considering all sub-tree rooted at v not containing the edge which contains the new mask). So we have $(N_i + 1) + 2^{m'_i}$ many different sets in $S(v)$. Similarly we can prove when $|\text{New}(i)| = 2$. □

Observation : If ψ is even-free then, $|\text{New}(i)|$ is either 1 or 2 and more importantly, $2^{m'_i - 1} \leq N_i < 2^{m'_i}$. The first inequality can be proved by induction using the above Lemma. Now we have one of our main Theorem in this chapter:

Theorem 3.6 *If ψ is level-uniform even-free masking assignment then for any $l_k + 1 \leq i \leq l_{k+1}$, $N_i \geq 2^{m_i} - j \geq 2^{m_i - 1}$, where $j = l_{k+1} - i + 1$. In particular, $m'_i \geq m_i$. So, IBTC is optimal in all valid level-uniform domain extensions.*

Proof. By induction on i. Suppose the result is true for i. First assume $i = l_k$ then, $N_i \geq 2^{m_i} - 1$ (by induction hypothesis, here $j = 1$). Now consider two cases.

1. $m'_i \leq m_i$: $N_i \geq 2^{m'_i} - 1$ and hence $N_i = 2^{m'_i} - 1$ and $m_i = m'_i$. Now, $|\text{New}(i+1)| = 2$ as all non-empty subsets of $[1, m_i]$ occur in $S(v)$ where $ht(v) = i$. So we can apply the Lemma 3.3 for $|\text{New}(i+1)| = 2$. We have, $N_{i+1} = 3.2^{m'_i} = 3.2^{m_i} = 2^{m_i + 2} - 2^{m_i} = 2^{m_{i+1}} - 2^{m_i}$. Now note that $l_{k+1} - l_k = 2^{m_i}$. So we are done.

2. $m'_i > m_i$: Apply the corollary for both possible values of $|New(i+1)|$.

If $j > 1$ then $N_{i+1} \geq (N_i + 1) + 2^{m'_i} \geq 2^{m_i + 1} - j + 1$ (since $i \neq l_k$). For smaller values of i, say $i = 2$, proof of the result is trivial. □

So we have proved that we can not have level-uniform even-free masking assignment using less than m_t many masks. We prove now that minimum number of masks for strongly even-free masking assignment is m_t.

Theorem 3.7 *The minimum number of masks for a strongly even-free masking assignment on a complete binary tree of height t is m_t.*

Proof. Let $L_{i,j}(v)$ be a i-λ tree where the binary tree is the complete binary tree of height j with root v. We prove that if $\psi : L_{i,j} \to [1, l]$ is strongly even-free masking assignment then there exists $\psi' : L_{i,j} \to [1, l]$ which is level uniform even-free (i.e. it is level uniform on the complete binary tree). In particular, when $i = 1$, $L_{1,j}$ is the complete binary tree of height t. So, we can change a strongly even-free masking assignment to a level uniform even-free masking assignment on full binary tree using same number of masks and hence by Theorem 3.6, minimum number of masks for strongly even-free masking assignment is m_t and IBTC achieves this bound.

We prove the result by induction on $|L_{i,j}(v)|$. Suppose $e = (u_1, v_1)$ is an edge so that $\psi(e) = l$ (say) is a single-man for the subtree $L_{i,j}(v)$. If $e \in S_i$ then by removing e we have $L_{i',j}(u_1)$ as one component and the other component is a sequential tree. As $i' < i$, by induction hypothesis we can change the masking assignment ψ restricted at $L_{i',j}(u_1)$ to level uniform even-free masking assignment ψ' using same set of masks. On the rest of the tree ψ' is same as ψ. If $e \in T_j$ then $L_{i+1,j-1}(u)$ is a subtree of bigger component of $L_{i,j}(v) - \{e\}$ where u is a son of v. Again, change ψ on that part to level uniform even-free ψ' (by induction hypothesis). Then define $\psi'(u', v)$ by $\psi(e)$ and ψ' on the full binary tree rooted at u' is same as ψ on rooted subtree at u. It is easy to check that, ψ' is level uniform masking assignment. □

3.5 Conclusion

In this Chapter we have provided two parallel domain extenders, IBTC and OPT. Each of them has an important theoretical meaning in the study of efficient domain extension method for UOWHF.

IBTC was a reasonable improvement over the best known complete binary tree based extension [89]. It uses $m_t = O(t + \log_2^* t)$ many masks for a complete binary tree of height t. Previously, it was $(t + \log_2 t)$ [89]. We also proved that this is optimum among all complete binary tree based domain extension where the masking assignment is level-uniform. We can pose the following conjecture;

Conjecture : The minimum number of masks to be required to have a strongly even-free masking assignment on a complete binary tree is m_t.

On the other hand, OPT is optimum in both key size and the number of rounds which is

proportional to the time complexity provided we have sufficient number of processors. Till now it is unknown how efficiently OPT can be implemented in parallel. Even if it carries potential efficiency it is not clear how the optimum performance can be achieved. So it would be an interesting question how to implement OPT efficiently.

This chapter has concerned the efficient parallel construction. Of course, it would be very nice to have a parallel construction which has the optimal key length expansion and the same or more efficient parallelizability than complete tree based constructions simultaneously. But at this point, we do not have any such algorithm. Hence, in our opinion, we should separately consider both the key length expansion and the parallelizability with the same importance. And we would like to stress that the present work is important regarding both point of views.

We also study how to check UOWHF-preserving property of a domain extension algorithm by just verifying a simple property called strongly even-free. The sufficient condition makes easy to construct a UOWHF-preserving domain extension algorithm. It is very interesting to note that all known UOWHF-preserving domain extension algorithms satisfy the sufficient condition. So one can try to prove that the condition is a necessary condition.

Conjecture : A necessary condition for UOWHF-preserving domain extension algorithm is that the underlying masking assignment is strongly even-free.

Chapter 4

PGV-Hash Family

4.1 Introduction

Nowadays many people are designing hash functions based on block ciphers. Let $E : \{0,1\}^m \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher, where $\{0,1\}^m$ is the key space. Note that the permutation $E(K,\cdot)$ itself compresses the input considering K as a input, $|K| = m$. Moreover, an ideal block-cipher E should behave like a random permutation. Thus, it is likely that a good compression function can be constructed based on an ideal block-cipher E.

There are many constructions of compression functions $f : \{0,1\}^{n+m} \rightarrow \{0,1\}^n$ based on block-ciphers. For example, Davis-Meyer [80, 111] compression function, $f(x,h) = E_x(h) \oplus h$, $|x| = m$ and $|h| = n$, When $m = n$ we have, Miyaguchi [65] and Preneel [80] compression function, $f(x,h) = E_x(h) \oplus x \oplus h$, $|h| = |x| = n$. Later, B. Preneel, R. Govaerts and J. Vandewalle [80] classified block-cipher based hash functions. Let $E : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher. We have compression function $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ defined by $f(h,m) = E_a(b) \oplus c$, where $a, b, c \in \{h, m, h \oplus m, v\}$. Here, $|h| = n$, $|m| = n$ and v is some fixed n-bit constant. Note that there are sixty-four compression functions of the above form. These above compression functions are known as PGV-compression functions and hash functions based on these compression functions are known as PGV-hash functions.

In PGV-hash functions, key of the block ciphers depends on the message blocks or chaining hash values. Thus, we have to make key scheduling algorithm of the block-ciphers at each round. But it was shown [9] that there is no highly efficient (where

the keys of block cipher are independent of message blocks and chaining hash values) block cipher based secure hash function.

Organization of The Chapter

In this chapter, we are mainly interested in PGV hash functions. J. Black, P. Rogaway, and T. Shrimpton [10] proved that out of sixty four hash functions, only twelve PGV-compression functions and twenty PGV-hash functions are secure with respect to collision and preimage attacks. They proved the security in the black box model (see Section 2.3.3 for more details). In this chapter we generalize the definition of PGV hash function and study the collision resistance and other security properties in the black box model. We generalize the definition of a PGV hash function by making them into a hash family. We call these hash families by PGV hash families. We also study their security properties in the black-box model. We found many of the hash families (more than twenty) are collision and preimage secure with tight security bounds. The security bound is the maximum success probability of any attacker for finding collision or preimage using some number of queries. We define the security bound in Section 2.3.2 in more details. Also we study their "UOWHF" property as they are hash families unlike original PGV hash functions.

4.2 Generalized PGV-Hash Family

Let $0 \leq l < n$ and $E : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher. If $l = 0$, let $\{0,1\}^0 = \{\epsilon\}$, where ϵ is the empty string. Given two fixed n-bit constants $h_0, v \in \{0,1\}^n$ and $k \in \{0,1\}^l$, define $f^k : \{0,1\}^n \times \{0,1\}^{n-l} \rightarrow \{0,1\}^n$ as follows;

$$f^k(h,m) = E_a(b) \oplus c$$

where $a, b, c \in \{h, (m||k), h \oplus (m||k), v\}$. Note that $|h| = n$, $|k| = l$ and $|m| = n - l$. If $l = 0$ then $\mathcal{F} = \{f^k\}_{k \in \{0,1\}^0} = \{f^\epsilon\}$ is a singleton set and this corresponds to the original definition of PGV. In this case, we denote this $\mathcal{F}$ as just f without superscript ϵ. We call this f a (*block-cipher- based*) *compression function*. Similarly, we denote $\mathcal{H}$ as H without superscript ϵ. And we call this H an *extended hash function*.

This new generalized hash families are less efficient than the original hash functions. In fact, we hash $n-l$-bits of message per invocation of compression function. For example, consider $l = n/2$. In this case we hash $n/2$ message-bits. If we look at the Table 4.1, we see that the E2 schemes remain secure when $l = n/2$. The collision security bound is $\Theta(2^{n/2})$.

Table 4.1: The $l = 0$ is analyzed in [80] . The case $l > 0$ is analyzed in this thesis.

$l = 0$	(T)Coll Bound	Inversion Bound
E1 (20 schemes)	$\Theta(q^2/2^n)$	$\Theta(q/2^n)$ or $\Theta(q^2/2^n)$
E2/.../E6 (44 schemes)	$\Theta(1)$	–
$l > 0$	(T)Coll Bound	Inversion Bound
E1 (20 schemes)	$\Theta(q^2/2^n)$	$\Theta(q/2^l)$ or $\Theta(q/2^n)$ or $\Theta(q^2/2^n)$
E2 (4 schemes)	$\Theta(q/2^l)$	$\Theta(q/2^l)$
E3/E4/E5 (18 schemes)	$\Theta(q^2/2^l)$	$\Theta(q/2^l)$ or $\Theta(q^2/2^l)$ or $\Theta(q/2^n)$
E6 (22 schemes)	$\Theta(1)$	–

Here we fix E1 = $\{1, ..., 20\}$, E2 = $\{21, 22, 26, 28\}$, E3= $\{23, 24, 25, 31, 34, 35\}$, E4= $\{27, 29, 30, 32, 33, 36\}$, and E5 = $\{37, \cdots, 42\}$. Here, the numbers are corresponding to the numbers in the first column of Fig. 4.2 and 4.3. And E6 is the set of remaining extended hash families which are not represented in the first column of Fig. 4.2 and 4.3. So |E6| = 22. This classification is based on some properties of these hash families which are used in the proofs of security.

The black-box model is stated in Sect. 2.3.3. We define transcript or view of an adversary in the black box model in the following way;

Definition 4.1 Transcript *or* view *of an adversary, $\mathcal{A}$, is defined by the sequence of query-response quadruples $\{(s_i, x_i, y_i, \sigma_i)\}_{1 \le i \le q}$, where q is the maximum number of queries made by adversary, $s_i, x_i, y_i \in \{0,1\}^n$ and $\sigma_i = +1$ (in case of E-query) or -1 (in case of E^{-1}-query) and for all i, $E_{s_i}(x_i) = y_i$, where q is the maximum number of queries made by adversary,.*

The four-tuple $(s_i, x_i, y_i, \sigma_i)$ is called by i^{th} query-response quadruple (or q-r quadruple). Note that, if $\sigma_i = +1$ (respectively, -1) then y (resp. x) is a random string as

we assume that the block-cipher $E_s(\cdot)$ is a random permutation. Thus we have the following proposition which will be useful to find security bounds.

Proposition 4.1 *For fixed $x, y \in \{0,1\}^n$ and $A \subseteq \{0,1\}^n$, $Pr[y_i = y] \leq \frac{1}{2^n - i + 1}$ and $Pr[y_i \in A] \leq \frac{|A|}{2^n - i + 1}$ whenever $\sigma_i = +1$. Similarly, if $\sigma_i = -1$ then $Pr[x_i = x] \leq \frac{1}{2^n - i + 1}$ and $Pr[x_i \in A] \leq \frac{|A|}{2^n - i + 1}$*

Proof. Before i^{th} query at most $(i-1)$ outputs (or inputs) of a block-cipher with same key are known. Thus, the output (or input) of i^{th} query of E will be uniformly distributed over a set of size at least $2^n - (i-1)$ elements. □

4.3 Collision Resistance of Extended Hash Family

In this section we analyze the security of $\mathcal{H}_\imath$ for each $\imath \in [1, 42]$ defined in Figure 4.2 and Figure 4.3 in the notion of collision resistant. We consider any adversary $\mathcal{A}$ with respect to Coll. i.e. after having random key k he will try to find a collision pair (M_1, M_2) for $H_\imath^k$ i.e. $M_1 \neq M_2$, $H_\imath^k(M_1) = H_\imath^k(M_2)$. For that he will make some E/E^{-1} queries. Since a collision resistant function is also a target collision resistant function, it is sufficient to study the collision resistance property.

Definition 4.2 (useful arc set)

For any arbitrary hash family $\mathcal{F}_\imath$ for $\imath \in [1, 42]$ and a triple $\tau = (s, x, y)$ where, $s, x, y \in V$ define a set of labeled arcs $A(\tau)$, called useful arc set*; $A(\tau) = \{(h_1, h_2, m) \in V \times V \times L : f_\imath^k(h_1, m) = h_2 \Rightarrow E_s(x) = y\}$.*

Example 4.1 *For $\mathcal{F}_{21}$, $f_{21}^k(h_1, m) = E_{h_1}(m||k) \oplus h_1$. Thus, $f^k(h_1, m) = h_2 \Rightarrow E_{h_1}(m||k) \oplus h_1 = h_2$. So, $(h_1, h_2, m) \in A(\tau) \Rightarrow h_1 = s, h_2 = y \oplus h_1 = y \oplus s, m||k = x$. Hence, $A(\tau) = \{(s, s \oplus y, x[L])\}$ if $x[R] = k$ otherwise it is an empty set ($x = x[L]||x[R]$, where $|x[L]| = n - l$ and $|x[R]| = l$.*

Given a set of labeled arcs A we define induced arc set $A' = \{(h_1, h_2) : \exists m \in L, (h_1, h_2, m) \in A\}$. For a set of triple(s) $\tau = \{\tau_1 = (s_1, x_1, y_1), \ldots, \tau_a = (s_a, x_a, y_a)\}$ we

can define *labeled arc set* $A(\tau) = \bigcup_{i=1}^{a} A(\tau_i)$. It can be easily checked that $A'(\tau) = \bigcup_{i=1}^{a} A'(\tau_i)$. Every member of $A(\tau)$ (or $A'(\tau)$) will be called an *labeled arc* (or *arc*) *corresponding* to the set of triple(s) τ. Given a transcript $\{(s_i, x_i, y_i, \sigma_i)\}_{1 \le i \le q}$ of an adversary $\mathcal{A}$ let $\tau[i]$ denote the set of triples $\{\tau_1 = (s_1, x_1, y_1), \ldots, \tau_i = (s_i, x_i, y_i)\}$. For each i we have a labeled directed graph $T_i = T(\tau[i]) = (V, A(\tau[i]))$ and a directed graph $T_i' = (V, A'(\tau[i]))$. Define $T_0 = (V, \emptyset)$.

Lemma 4.1 *Adversary can compute $f_i^k(h_1, m) = h_2$ after i^{th} query if and only if for some $j \le i$, $E_{s_j}(x_j) = y_j \Rightarrow f_i^k(h_1, m) = h_2$ and hence $(h_1, h_2, m) \in A(\tau[i])$. Similarly, adversary can compute $H_i^k(m_1 || \cdots || m_a) = h_a$ after i^{th} query if and only if $h_0 \rightarrow_{m_1} h_1 \rightarrow_{m_2} \cdots \rightarrow_{m_a} h_a$ is a path in $A(\tau[i])$.*

Definition 4.3 *For each $1 \le i \le q$, we define some useful events related to collision. C_i : adversary gets a collision after i^{th} query. $\mathsf{PathColl}_i$: $\exists$ two distinct labeled paths P_1 and P_2 in T_i from h_0 to some h^*.*

Obviously the above events C_i and $\mathsf{PathColl}_i$ are equivalent. Thus to estimate the collision probability we only need to estimate the probability of existence of $\mathsf{PathColl}_q$, where q is the maximum number of queries. Before this we have to understand the graph T_i in more detail. The following proposition says about the structure of $A(\tau_i)$. Note that T_i is the union of all graphs $A(\tau_j)$ for $1 \le j \le i$.

Proposition 4.2 *If $A(\tau_i)$ is not empty then we have,*

1. *For $\imath \in$ E1 or E2, $A(\tau_i)$ is a singleton*
2. *For $\imath \in$ E3, $A'(\tau_i) = \{(h_1, h_2) : h_2[R] = u\}$ where, h_1 and u are fixed depending only on j and τ_i. So, the graph of the $A'(\tau_i)$ looks like an outward directed star and $|A'(\tau_i)| = 2^{n-l} = |A(\tau_i)|$.*
3. *For $\imath \in$ E4, $A'(\tau_i) = \{(h, h \oplus a) : h[R] = u\}$ where, a and u are fixed depending only on j and τ_i. So, the graph of the $A'(\tau_i)$ consists of 2^{n-l} parallel arcs and $|A'(\tau_i)| = 2^{n-l} = |A(\tau_i)|$.*

4. *For $\imath \in E5$, $A'(\tau_i) = \{(h_1, h_2) : h_1[R] = u\}$ where, h_2 and u are fixed depending only on j and τ_i. So, the graph of the $A'(\tau_i)$ looks like an inward directed star and $|A'(\tau_i)| = 2^{n-l} = |A(\tau_i)|$.*

Moreover, for each $(h_1, h_2) \in A'(\tau_i)$, there exists unique m such that $h_1 \rightarrow_m h_2$.

Proof. We prove the results for one hash function from each class. Other cases will be very similar and one can check analogously. Let $\tau_i = (s_i, x_i, y_i)$.

1. In case of $\mathcal{H}_1$, $f_1^k(h_1, m) := E_{h_1}(m||k) \oplus (m||k)$. So, $f^k(h_1, m) = h_2 \Rightarrow E_{h_1}(m||k) \oplus (m||k) = h_2 \Rightarrow (h_1 = s_i,\ h_2 = y_i \oplus (m||k), x_i = m||k)$. Hence, $A(\tau) = \{(s_i, y_i \oplus x_i, x_i[L])\}$ if $x_i[R] = k$ otherwise it is an empty set.

   In case of $\mathcal{H}_{21}$, we have shown in Example 4.1, that $A(\tau) = \{(s_i, s_i \oplus y_i, x_i[L])\}$ if $x_i[R] = k$ otherwise it is an empty set.

2. In case of $\mathcal{H}_{23}$, $f_{23}^k(h_1, m) := E_{h_1}(h_1) \oplus (m||k)$. So, $f^k(h_1, m) = h_2 \Rightarrow E_{h_1}(h_1) \oplus (m||k) = h_2 \Rightarrow (h_1 = s_i = x_i,\ h_2 = y_i \oplus (m||k))$. Hence, $A(\tau) = \{(s_i, h_2, m) : h_2[R] = y_i[R] \oplus k, m = h_2[R] \oplus y_i[R]\}$ if $x_i = s_i$ otherwise it is an empty set.

3. In case of $\mathcal{H}_{27}$, $f_{27}^k(h_1, m) := E_{w_1}(w_1) \oplus (m||k)$ where $w_1 = h_1 \oplus (m||k)$. So, $f^k(h_1, m) = h_2 \Rightarrow E_{w_1}(w_1) \oplus (m||k) = h_2 \Rightarrow (h_1 = s_i \oplus (m||k),\ h_2 = y_i \oplus (m||k) = h_1 \oplus (y_i \oplus s_i), s_i = x_i)$. Hence, $A(\tau) = \{(h_1, h_1 \oplus (s_i \oplus y_i), x_i[L] \oplus h_1[R])\}$ if $x_i = s_i$ otherwise it is an empty set.

4. In case of $\mathcal{H}_{36}$, we can prove similarly that $A(\tau_i) = \{(h_1, y_i \oplus v, m) : h_1[R] = s_i[R] \oplus k, m = h_1[L] \oplus s_i\}$ if $x_i = s_i$ otherwise it is an empty set. □

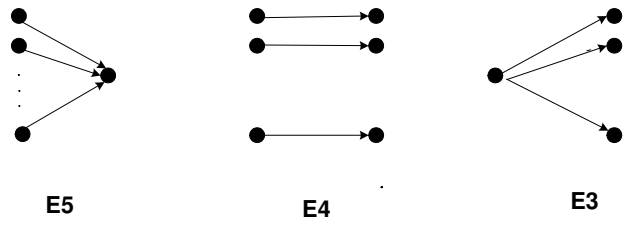

Figure 4.1: The Graphs of $A'(\tau)$ for E3/E4/E5 Hash Families

Definition 4.4 *For each hash function and* $0 \leq i \leq q$

1. *When* $\imath \in$ *E1, E2 or E3,* h *in* T_i *is* **old** *if* $deg(h) \geq 1$ *in* T_i *or* $h = h_0$.
2. *When* $\imath \in$ *E4 or E5,* h *in* T_i *is* **old** *if* $h = h_0$ *or there exists* h_1, $deg(h_1) \geq 1$ *in* T_i *and* $h[R] = h_1[R]$.

Remaining all other vertices are known as **new** *vertices. Denote the set of all* old *vertices in* T_i *by* O_i.

Proposition 4.3 1. *For* $\imath \in$ *E1 or E2, we have* $|O_i| \leq 2i + 1$.

2. *For* $\imath \in$ *E3, we have,* $|O_i| \leq (2i + 1)2^{n-l}$.
3. *For* $\imath \in$ *E4, we have* $|O_i| \leq (2i + 1)2^{n-l}$.
4. *For* $\imath \in$ *E5, we have* $|O_i| \leq (2i + 1)2^{n-l}$.

Proof. This is immediate from Proposition 4.2. □

Define an event Succ_i occurs if $\exists$ an arc $(h, h') \in A'(\tau_i)$ where both h and h' are old vertices in T_{i-1}.

Proposition 4.4 *For E1-E4 hash families, the event* $(\mathsf{Pathcoll}_i \mid \neg\mathsf{Pathcoll}_{i-1})$ *necessarily implies* Succ_i. *For E5,* $\mathsf{Pathcoll}_i$ *necessarily implies* $\mathsf{Succ}_{i'}$ *for some* $i' \leq i$.

Proof. Let P_1 and P_2 be two distinct labeled paths from h_0 to h^* in T'_i. Since $\mathsf{PathColl}_{i-1}$ is not true $\exists$ at least one arc in $P_1 \cup P_2$ which corresponds to τ_i. If Succ_i is not true then one of the vertices of an arc corresponding to τ_i should be new in T_{i-1} which implies $\exists$ two arcs either $(h_1, h_2), (h_2, h_3)$ or $(h_1, h_3), (h_2, h_3)$ corresponding to τ_i. But this is not possible by the structure of $A'(\tau_i)$ (see Proposition 4.3) in case of E1, E2, E3 and E4 hash families.

For E5 hash families, $A'(\tau_i)$ can not contain (g_1, h_1) and (g_2, h_2) with $h_1 \neq h_2$. Let P'_i be the corresponding non-labeled directed path of P_i, $i = 1, 2$. Thus for $P'_1 = P'_2$, the proof is trivial. Otherwise there exists g_1 and g_2 such that both (g_1, h) and (g_2, h) belongs to $A'(\tau_i)$ and g_1 and g_2 are old vertices. But g_1 and g_2 both can not become

old vertex after i_1^{th} query for $i_1 < i$. So there exists i' such that g_2 become old for the first time and g_1 is already old. Thus the events $\mathsf{Succ}_{i'}$ occurs. □

Since $\mathsf{C}_q \Rightarrow \bigcup_{i=1}^{q} \mathsf{Succ}_i$ (Proposition 4.4), we have $\Pr[\mathcal{A} \text{ gets a collision}] \leq \sum_{i=1}^{q} \Pr[\mathsf{Succ}_i]$. Thus it is enough to have an upper bound of $\Pr[\mathsf{Succ}_i]$ in all hash functions.

Theorem 4.1 *For each $1 \leq i \leq q$ we have*

1. *For E1 hash family, $Pr[\mathsf{Succ}_i] \leq (2i-1)/2^{n-1}$*
2. *For E2 hash family, $Pr[\mathsf{Succ}_i] \leq 2/(2^{l+1}-1)$ if $q \leq 2^{n-l-1}$.*
3. *For E3,E4 or E5 hash families, $Pr[\mathsf{Succ}_i] \leq (2i-1)/2^{l-1}$.*

Proof. Let $\mathcal{A}$ be an adversary attacking $\mathcal{H}_\iota$. Assume that $\mathcal{A}$ asks its oracles at most q total queries. Assume that the random key k is given. Let $(s_i, x_i, y_i, \sigma_i)$ be the i^{th} q-r quadruple.

Consider H_1^k in case of E1 hash family. For the other hash families in E1, the proof is analogous to the proof of 1. Let $\sigma_i = +1$. $\mathsf{Succ}_i \Rightarrow y_i \oplus x_i \in O_{i-1}$ (See Proposition 4.3). Hence, $\Pr[\mathsf{Succ}_i] \leq \Pr[y_i \in O_{i-1} \oplus x_i] \leq (2i-1)/(2^n - i + 1)$ (by Proposition 4.1 and 4.3). In case of, $\sigma_i = -1$, $\mathsf{Succ}_i \Rightarrow y_i \oplus x_i \in O_{i-1}$ (See Proposition 4.2). Hence, $\Pr[\mathsf{Succ}_i] \leq \Pr[x_i \in O_{i-1} \oplus y_i] \leq (2i-1)/(2^n - i + 1)$ (by Proposition 4.1 and 4.3). Therefore, $\Pr[\mathsf{Succ}_i] \leq (2i-1)/(2^n - i + 1) \leq (2i-1)/2^{n-1}$.

Consider H_{21}^k in case of E2 hash family. For the other hash families in E2, the proof is analogous to the proof of 21. If $\sigma_i = +1$, $\mathsf{Succ}_i \Rightarrow y_i \oplus s_i \in O_{i-1}$ (See Proposition 4.3). Hence, $\Pr[\mathsf{Succ}_i] \leq \Pr[y_i \in O_{i-1} \oplus s_i] \leq (2i-1)/(2^n - i + 1)$ (by Proposition 4.1 and 4.3). If $\sigma_i = -1$, $\mathsf{Succ}_i \Rightarrow x_i[R] = k$. Let $Q = \{x : x[R] = k\}$ then $|Q| = |2^{n-l}|$. Hence, $\Pr[\mathsf{Succ}_i] \leq \Pr[x_i \in Q] \leq 2^{n-l}/(2^n - i + 1)$ (by Proposition 4.1). Therefore, $\Pr[\mathsf{Succ}_i] \leq \max\{(2i-1)/(2^n - i + 1), 2^{n-l}/(2^n - i + 1)\}$. Since $q \leq 2^{n-l-1}$, $\Pr[\mathsf{Succ}_i] \leq 2^{n-l}/(2^n - i + 1) \leq 2/(2^{l+1} - 1)$.

Consider H_{23}^k in case of E3 hash family. For the other hash families in E3, the proof is analogous to the proof of 21. For E4/E5 hash functions the proof will be analogous to the proof of 23. If $\sigma_i = +1$, then Succ_i implies that $\exists$ an arc $(h, h') \in A(\tau_i)$ such that $h' \in O_{i-1}$. This implies that $\exists\ m$ such that $(y_i \oplus (m||k)) \in O_{i-1}$. By the Proposition 4.3 $(y_i \oplus (m||k)) \in O_{i-1} \Leftrightarrow (y_i \oplus (0||k)) \in O_{i-1} \Leftrightarrow y_i \in O_{i-1} \oplus (0||k)$.

Therefore, by the Proposition 4.1 and 4.3, $\Pr[\mathsf{Succ}_i] \leq 2^{n-l}(2i-1)/(2^n-i+1)$. If $\sigma_i = -1$, then Succ_i implies that $x_i = s_i$. Hence, $\Pr[\mathsf{Succ}_i] \leq \Pr[x_i = s_i]$. Hence, by the Proposition 4.1, $\Pr[\mathsf{Succ}_i] \leq \Pr[x_i = s_i] \leq 1/(2^n - i + 1)$. Therefore, $\Pr[\mathsf{Succ}_i] \leq \max\{2^{n-l}(2i-1)/(2^n-i+1), 1/(2^n-i+1)\} = 2^{n-l}(2i-1)/(2^n-i+1) \leq (2i-1)/2^{l-1}$. Since $q \leq 2^{n-l}$, $\Pr[\mathsf{Succ}_i] \leq 2^{n-l}(2i-1)/(2^n-i+1) = (2i-1)/(2^l - ((i-1)/2^{n-l})) \leq (2i-1)/(2^l-1)$. Therefore, we have $\mathbf{Adv}^{\mathrm{Coll}}_{\mathcal{H}_{23}}(q) \leq \Sigma_{i=1}^{q}\Pr[\mathsf{Succ}_i] \leq \Sigma_{i=1}^{q}(2i-1)/(2^l - 1) = q^2/(2^l-1)$. □

Theorem 4.2 *1.* $\mathbf{Adv}^{Coll}_{\mathcal{H}_\imath}(q) \leq q^2/2^{n-1}$ *for* $\imath \in$ *E1*

2. $\mathbf{Adv}^{Coll}_{\mathcal{H}_\imath}(q) \leq 2q/(2^{l+1}-1)$ *for all* $q \leq 2^{n-l-1}$ *and* $\imath \in$ *E2.*

3. $\mathbf{Adv}^{Coll}_{\mathcal{H}_\imath}(q) \leq q^2/2^{l-1}$ *for* $\imath \in$ *E3, E4 or E5.*

Theorem 4.3 *For all* $\imath \in [1, 42]$, $\mathbf{Adv}^{Coll}_{\mathcal{H}_\imath}(q) \leq \mathbf{Adv}^{Coll}_{H_\imath}(q)$

Proof. Suppose $\mathcal{A}$ is an adversary with respect to Coll for the hash family $\mathcal{H}_\imath$. We can construct an adversary $\mathcal{B}$ with respect to Coll for $H_\imath$ very easily. Choose k at random from $\{0,1\}^l$. Run $\mathcal{A}$ to get M_1 and M_2 where, $M_1 = m_1^1||\cdots||m_a^1$, $M_1 = m_1^2||\cdots||m_b^2$, $|m_i^j| = n - l$ and $j = 1$ or 2. $\mathcal{B}$ outputs (M_1', M_2') where $M_1' = (m_1^1||k)||\cdots||(m_a^1||k)$, and $M_2' = (m_1^2||k)||\cdots||(m_b^2||k)$. It is very easy to check that if (M_1, M_2) is a collision pair for $H_\imath^k$ then (M_1', M_2') is a collision pair for $H_\imath$. Note, whenever $\mathcal{A}$ asks for E-query/E^{-1}-query, $\mathcal{B}$ asks same query and output of the query is given to $\mathcal{A}$ as a response of the query made by $\mathcal{B}$. □

In [10] we know the followings :

1. For $\imath \in [1, 12]$, $\mathbf{Adv}^{\mathrm{Coll}}_{H_\imath}(q) \leq q(q+1)/2^n$
2. For $\imath \in [13, 20]$, $\mathbf{Adv}^{\mathrm{Coll}}_{H_\imath}(q) \leq 3q(q+1)/2^n$

So, we can conclude from Theorem 4.2 and 4.3 that,

Corollary 4.1 *For* $\imath \in [1, 12]$, $\mathbf{Adv}^{TColl}_{\mathcal{H}_\imath}(q) \leq \mathbf{Adv}^{Coll}_{\mathcal{H}_\imath}(q) \leq q(q+1)/2^n$.
For $\imath = [13, 20]$, $\mathbf{Adv}^{TColl}_{\mathcal{H}_\imath}(q) \leq \mathbf{Adv}^{Coll}_{\mathcal{H}_\imath}(q) \leq q^2/2^{n-1}$.

In fact we can have better bound like $q(q+1)/2^n$ for the hash family $\mathcal{H}_\imath$ where $\imath \in [13, 20]$. For that we need to change the definition of old vertex.

4.4 Target Collision Attack

The idea of attack : Here we give a generic attack for all $\mathcal{H}_j$ for the game TColl (See Section 2.3.2). This attack is not valid when the Merkle-Damgård strengthening is used. Commit $M_1 = (m_1||\ldots||m_q)$. we describe later how these m_i's will be chosen. Then given random key k compute $\mathcal{H}_j^k(M_1)$ by using q many queries. We obtain $h_1, \ldots h_q$ and $\mathcal{H}_j^k(M_1) = h_q$ where, $h_0 \rightarrow_{m_1} h_1 \rightarrow_{m_2} \ldots h_{q-1} \rightarrow_{m_q} h_q$. If we get one such $i < i'$ such that $h_i = h_{i'}$ then define $M_2 = m_1||\ldots||m_i||m_{i'+1}||\ldots m_q$. So, M_1 and M_2 will be a collision pair. Roughly h_i's are random string and the probability of success will be probability for birthday collision of h_i's which is $o(q^2/2^n)$. We choose m_i's such that the key for each query (i.e. s_i's) are different.

Choice of m_i's : If key of block cipher E is w in the definition of compression function then choose $m_i = 0$. So each w_i will be different as h_i's are different otherwise we get a collision. If key is h or m then choose $m_i = i$ and hence keys are different. If key is v then choose m_i's so that inputs of compression functions are different. In this case we study the lower bound separately.

Theorem 4.4 $\mathbf{Adv}^{Coll}_{\mathcal{H}_\imath}(q) \geq \mathbf{Adv}^{TColl}_{\mathcal{H}_\imath}(q) \geq \frac{0.3q(q-1)}{2^n}$ *for each* $\imath \in [1, 42]$ *whenever key of* E *is not* v *in the definition of compression function.*

Proof. Define D_i by the event that no collision occurs after i^{th} query and D is the event that the above attack fails after all queries i.e. it is same as D_q. Define D_0 by a sure event. Now $Pr[D] = \prod_{i=1}^{q} Pr[\mathsf{D}_i|\mathsf{D}_{i-1}]$. If D_{i-1} is true then all $h_{i'}$'s are different for $i' < i$. Now $h_i = y_i \oplus \alpha_j$ (here α_j depends on h_{i-1}, m_i and v). Now D_i is true $\Leftrightarrow y_i \notin \{h_0, h_1, \ldots, h_{i-1}\} \oplus \alpha_j$. So, $Pr[\mathsf{D}_i|\mathsf{D}_{i-1}] = (1 - \frac{i}{2^n})$. So $\mathbf{Adv}^{\mathrm{TColl}}_{\mathcal{H}_\imath}(q) \geq 1 - \prod_{i=1}^{q}(1 - \frac{i}{2^n}) \geq \frac{\cdot 3q(q-1)}{2^n}$ (the last inequality is followed from Proposition 4.5 which is given at the end of this Section). □

For hash family E3/E4/E5 we can have better attack with lower bound $o(\frac{q^2}{2^l})$ if we just check whether $h_i[R] = h_{i'}[R]$ for $i < i'$. Construction of M_2 is given below where $h_i[R] = h_{i'}[R]$ for $i < i'$:

1. E3 : In E2 family if $h \rightarrow_m h'$ then $(h \oplus (a||0)) \rightarrow_{m \oplus a} (h' \oplus (a||0))$. So, define $M_2 = m_1||\ldots||m_{i'}||(m_{i+1} \oplus a)||\ldots||m_{i'} \oplus a||m_{i+1}||\ldots||m_q$. Here, $a = h_i[R] \oplus h_{i'}[R]$. This will give collision because $\mathcal{H}_j(m_1||\ldots||m_{i'}||(m_{i+1} \oplus a)||\ldots||(m_{i'} \oplus a) = h_i$.

2. E4 : By Proposition 4.3 we have some $m'_{i'}$ such that $h_{i'-1} \rightarrow_{m'_{i'}} h_{i'}$. So define $M_2 = m_1||\ldots||m_{i-1}||m'_{i'}||\ldots||m_q$. This will give a collision.

3. E5 : This case is very similar to E4 so we skip this.

Theorem 4.5 *Let $\imath \in$ E3 or E4 or E5. If v is not the key of E in the definition for compression function then* $\mathbf{Adv}^{Coll}_{\mathcal{H}_\imath}(q) \geq \mathbf{Adv}^{TColl}_{\mathcal{H}_\imath}(q) \geq \frac{0.3q(q-1)}{2^l}$. *In other cases* $\mathbf{Adv}^{Coll}_{\mathcal{H}_\imath}(q) \geq \mathbf{Adv}^{TColl}_{\mathcal{H}_\imath}(q) \geq \frac{0.3q(q-1)}{2^{l-1}}$.

Proof. We use same notations as above. If D_{i-1} is true then all $h_{i'}[R]$'s are different for $i' < i$. Now $h_i = y_i \oplus \alpha_j$ (here α_j depends on h_{i-1}, m_i and v). Now D_i is true $\Leftrightarrow$ $(y_i[R] \oplus \alpha =)$ $h_i[R] \notin \{h_0[R], h_1[R], \ldots, h_{i-1}[R]\}$. So, $y_i \notin A - \{y_1, \ldots, y_{i-1}\}$ where $A = \{x; x[R] \oplus a = h_{i'}[R], 0 \leq i' \leq i-1\}$ and $|A| = i.2^{n-l}$. Hence $Pr[\mathsf{D}_i|\mathsf{D}_{i-1}] = (1-\frac{i}{2^l})$. So $\mathbf{Adv}^{\mathrm{TColl}}_{\mathcal{H}_\imath}(q) \geq 1 - \prod_{i=1}^{q}(1 - \frac{i}{2^l}) \leq \frac{\cdot 3q(q-1)}{2^l}$ (the last inequality is followed from Proposition 4.5).

When key is same as v then everything is same as above except $Pr[\mathsf{D}_i|\mathsf{D}_{i-1}] = (1 - \frac{i2^{n-l}-i+1}{2^n-(i-1)})$ as y_i can not take previous $i-1$ outputs. So if $q \leq 2^{n-1}$, $Pr[\mathsf{D}_i|\mathsf{D}_{i-1}] \geq (1 - \frac{i}{2^{l-1}})$ and hence $\mathbf{Adv}^{\mathrm{TColl}}_{\mathcal{H}_\imath}(q) \geq \frac{\cdot 3q(q-1)}{2^{l-1}}$ □

Attack for E2 Hash Family : We consider $\mathcal{H}_{21}$ hash family from E2. Other cases are similar. Fix some integer $a > 0$ such that $(a+1)(a+2)/2 + a + 1 \geq q$. Let $m_1, \ldots m_a \in_R \{0,1\}^{n-l}$. Commit $M_1 = m_1||\ldots||m_a$ where m_i's are chosen like above (to make the keys different and note that in E2 there is no hash function with key v). Then given random key k compute $\mathcal{H}_{21}(M_1)$ using a queries (we have to do it by our convention). We obtain $h_0, h_1 \ldots, h_a = \mathcal{H}_{21}(M_1)$. If $h_i = h_{i'}$ for some $i < i'$ then $M_2 = m_1||\ldots m_i||m_{j+1}||m_a$. Output M_2. Otherwise run the loop below for $q-a$ many times.

For i, $j = 0$ to a $(j \neq i+1,\ i \leq j)$

Compute $E^{-1}_{h_i}(h_i \oplus h_j) = x$

If $x[R] = k$ then $M_2 = m_1||\ldots||m_i||m_{j+1}$ and output M_2.

Theorem 4.6 *For each $\imath \in$ E2,* $\mathbf{Adv}^{Coll}_{\mathcal{H}_\imath}(q) \geq \mathbf{Adv}^{TColl}_{\mathcal{H}_\imath}(q) \geq \cdot 3a(a+1)/2^n + (q-a)/2^l$ *for integer $a > 0$ such that $(a+1)(a+2)/2 + a + 1 \geq q$.*

Proof. (Sketch) Here we have two possibility to get collision. In first case success probability is at least $\cdot 3a(a+1)/2^n$ by similar argument as above. In the second case $Pr[x[R] = k] \geq 1/2^l$ for each loop. Altogether we have success probability is at least $(q-a)/2^l$. One can write down the proof in more details. □

Proposition 4.5 *[10]*

$1 - \prod_{i=1}^{q}(1 - \frac{i}{2^a}) \geq \frac{\cdot 3q(q-1)}{2^a}$ *for any integer* a.

4.5 Inversion Resistance of Extended Hash Family

4.5.1 Upper Bound

In the Inv game a random key k and a random h^* will be given where, $h^* \in \{0,1\}^n$. Then he will try to compute M in case of extended hash function or h, m in case of compression function such that $H_i^k(M) = h^*$ or $f_i^k(h,m) = h^*$. If he finds that then we say that adversary wins. As we study in black-box model adversary can query E/E^{-1} similar to other games like Coll or TColl. So, adversary has a transcript or sequence of query-response quadruples $\{(s_i, x_i, y_i, \sigma_i)\}_{1 \leq i \leq q}$. In this section we modify the definition of old vertices. In addition to the previous old vertices we also include h^* as an old vertex in each T_i. By the new definition of old vertex, size of O_i is one more than that of previous O_i. Definition of Succ_i is same as previous definition. Note that the definition of Succ_i involves old vertices. In that sense this definition is changed a little. Like C_i we define Inv_i which means that adversary gets inverse of h^* (i.e. adversary wins) after i^{th} query. It is very easy to check that $(\mathsf{Inv}_i | \neg \mathsf{Inv}_i)$ implies Succ_i. So for extended hash family we have one upper bound for probability of winning in the Inv game which will be same as that in Coll game. But we can have better bound for extended hash family using the theorem below.

Theorem 4.7 $\mathbf{Adv}_{\mathcal{H}_\imath}^{Inv}(q) \leq UB_\imath$ *for each* $\imath \in$ *E2/E3/E4/E5.*

Proof. (Sketch) Proof is very much similar to that collision resistant. We only have to change the definition of old namely we consider h^* as an old vertex where h^* is given randomly to the adversary for finding inverse of that. All propositions and lemma will

be valid in the new definition. For example $\mathsf{C}_i|\neg\mathsf{C}_{i-1} \Rightarrow \mathsf{Succ}_i$ is true even if Succ is changed by new definition of old. □

Theorem 4.8 $\mathbf{Adv}^{Inv}_{\mathcal{H}_\imath}(q) \leq \mathbf{Adv}^{Inv}_{\mathcal{F}_\imath}(q)$ *for each* $\imath \in [1, 42]$.

The proof for single hash function and single compression function is given in [10]. Same proof will carry forward for hash family and compression family also. Intuitively finding inverse for extended hash family is stronger than finding that for compression function.

Now we first study the security analysis of inversion resistance of compression functions. It can be easily observed that, for $\imath \in \{15, 17, 19, 20, 35, 36, 37\}$, the compression functions are not inversion resistance-secure. All other compression functions are inversion resistance-secure.

Theorem 4.9 $\mathbf{Adv}^{Inv}_{\mathcal{F}_\imath}(q) \leq q/2^{l-1}$ *for* $\imath \in [21, 34]$ *or* $\imath \in \{13, 14, 16, 18\}$.

Proof. Here we consider the hash family $\mathcal{H}_{23}$. Other cases will be very similar. A random key k and h^* are given to the adversary. The event $(\mathsf{Inv}_i|\neg\mathsf{Inv}_{i-1})$ implies the arc (h, h^*) corresponds to τ_i for some h. So, $E_{s_i}(x_i) = y_i \Leftrightarrow h \rightarrow_m h^*$ for some h and m. So $h^* = y_i \oplus (m||k)$ and $s_i = x_i$.

1. If $\sigma_i = +1$ then $\Pr[\mathsf{Inv}_i|\neg\mathsf{Inv}_{i-1}] \leq \Pr[y_i[R] = h^*[R] \oplus k \leq 2^{n-l}/(2^n - i + 1) \leq 1/2^{l-1}$ (assume $q \leq 2^{n-l}$ otherwise the bound is trivial).
2. If $\sigma_i = -1$ then $\Pr[\mathsf{Inv}_i|\neg\mathsf{Inv}_{i-1}] \leq 1/(2^n - i + 1) \leq 1/2^{n-1}$.

So, $\mathbf{Adv}^{\mathrm{Inv}}_{\mathcal{F}_\imath}(q) \leq \sum_{i=1}^{q} \Pr[\mathsf{Inv}_i|\neg\mathsf{Inv}_{i-1}] \leq q/2^{l-1}$. □

Theorem 4.10 $\mathbf{Adv}^{Inv}_{\mathcal{F}_\imath}(q) \leq q/2^{n-1}$ *for* $\imath \in [38, 42]$ *or* $[1, 12]$.

$\mathbf{Adv}^{Inv}_{\mathcal{F}_\imath}(q) \leq q/2^{l-1}$ *for* $\imath \in \{13, 14, 16, 18\}$.

Proof. Consider $\imath = 38$. Other cases will be similar. In fact, the idea of proof is same with the previous one. $\mathsf{Inv}_i|\neg\mathsf{Inv}_{i-1}$ implies $y_i = h^* \oplus v$ and $x_i = s_i$. So whenever

$i \leq 2^{n-1}$, $\Pr[\mathsf{Inv}_i | \neg \mathsf{Inv}_{i-1}] \leq 1/2^{n-1}$ (check for $\sigma_i = +1$ and -1). $\square$

For other cases $\imath \in \{35, 36, 37\}$ we can use the same technique used in proving the upper bound for Coll game. By the discussion made in beginning of the section we can have the following theorem.

Theorem 4.11 $\mathbf{Adv}^{Inv}_{\mathcal{H}_\imath}(q) \leq q^2/2^{l-1}$ *for* $\imath \in [35, 37]$ *and* $\mathbf{Adv}^{Inv}_{\mathcal{H}_\imath}(q) \leq \mathbf{Adv}^{Inv}_{H_\imath}(q) \leq 9(q+3)^2/2^n$ *for* $\imath \in \{15, 17, 19, 20\}$.

In 21/22/23/24/25/26/27/28/29/30/31/32/33/34 has upper bound $q/2^{l-1}$ and lower bound $q/2^{l+1}$ (some cases we have better lower bound) [35,37] compression families are not OWF-secure but extended hash family has same upper bound as Coll bound and lower bound $q^2/2^{l+1}$. 38/39/40/41/42 has upper bound $q/2^{n-1}$ and lower bound $q/2^n$.

4.5.2 Some attacks in Inv game for Lower Bound

Attack 1 : When $\imath \in \{15, 17, 19, 20, 35, 36, 37\}$ i.e. when the corresponding compression functions are not inversion resistance-secure we can perform meet-in-the-middle-attack [52]. Similar idea of our attack is given in [10]. Given h_0 and h^* we compute two sets F and B such that $h \to h_1$ for every $h_1 \in F$ and $h_2 \to h^*$ for every $h_2 \in B$. Note we can construct B as the compression functions are not inversion resistance-secure. If we get an element in $F \cap B$ say h then we have an inverse element of h^*. More precisely, if $h_0 \to_{m_1} h \to_{m_2} h^*$ for some m_1 and m_2 then $m_1 || m_2$ will be an inverse element of h^*. So we have the following lower bound which is similar to the bound given in [10] and hence we skip the proof.

Theorem 4.12 $\mathbf{Adv}^{Inv}_{\mathcal{H}_\imath}(q) \geq (0.15)q^2/2^n$ *for* $\imath \in \{15, 17, 19, 20\}$ *and* $\mathbf{Adv}^{Inv}_{\mathcal{H}_\imath}(q) \geq (0.15)q^2/2^l$ *for* $\imath \in [35, 37]$.

Consider $\mathcal{H}_{35}$. $f^k_{23}(h, m) = E_v(h) \oplus m$. Choose $q/2$ different values of h and compute $E_v(h)$ for those h. Call that output set by S. Roughly all elements of S have different right part. Define $S_1 = \{a : a[R] = b[R] \text{ for some } b \in S\}$. $|S_1| \approx q2^{n-l-1}$. Similarly choose $q/2$ different values of y such that $y[R] = k \oplus h^*[R]$. Compute $E_v^{-1}(y)$ and get

the set T_1. Now if $h_1 \in S_1 \cap T_1$ then we can compute the inverse of h^*. $\Pr[S_1 \cap T_1 \neq \emptyset] \approx q^2/2^{l+1}$.

Attack 2 : The attacking algorithm is same as the generic attack for target collision resistance described in Section 4.4. We choose $m_1, \cdots, m_q$ and then compute $h_1, \cdots, h_q$ and finally we look for some h_i such that $h_i = h^*$ (for $\imath \in [38, 42]$ or $[1, 12]$) or $h_i[R] = h^*[R]$ (for $\imath \in [21, 34]$). One can prove it exactly but this will be same as the proof for collision attack so we skip the details. In case of $\imath \in \{13, 14, 16, 18\}$ choose s_i different and compute $x_i = E_{s_i}^{-1}(h^* \oplus v)$.

Theorem 4.13 $\mathbf{Adv}_{\mathcal{H}_\imath}^{Inv}(q) \geq q/2^{l+1}$ *for* $\imath \in [21, 34]$ *and* $\mathbf{Adv}_{\mathcal{H}_\imath}^{Inv}(q) \geq q/2^n$ *for* $\imath \in [38, 42]$ *or* $[1, 12]$.

4.6 Conclusion

In this thesis we first generalized the definition of PGV-hash functions into a PGV-hash families. In the new definitions we have more secure hash family (forty-two hash families) with respect to collision resistant and one-wayness. Unlike the original definition of PGV hash functions, the new definition is a keyed family. Thus we can study other security notions like target collision resistant. In fact all these forty-two hash families become target collision resistant. Even if there are many secure block-ciphers, it would be better if we use a hash function which is based on secure compression function. As AES is considered to be a good candidate for a block cipher, we can implement these hash families using AES. Because of our results, the only attacks for these hash families should be based on some internal properties of AES. In other words, these hash families can be practically constructed using AES. The proof techniques used here are natural and apply directly to the security notions. So one can also study these proof techniques to get better ideas about using the black box model.

$\imath$	$\jmath$	$h_i =$	**(T)CR LB**	**(T)CR UB**	**IR LB**	**IR UB**
	1	$E_{x_i}(x_i) \oplus v$	1	1	–	–
22	2	$E_{h_{i-1}}(x_i) \oplus v$	$q/2^{l+1}$	$2q/2^{l+1} - 1$	$q/2^{l+1}$	$q/2^{l-1}$
13	3	$E_{w_i}(x_i) \oplus v$	$.3q(q-1)/2^n$	$q^2/2^{n-1}$	$q/2^l$	$q/2^{l-1}$
	4	$E_v(x_i) \oplus v$	1	1	–	–
	5	$E_{x_i}(x_i) \oplus x_i$	1	1	–	–
1	6	$E_{h_{i-1}}(x_i) \oplus x_i$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
9	7	$E_{w_i}(x_i) \oplus x_i$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
	8	$E_v(x_i) \oplus x_i$	1	1	–	–
	9	$E_{x_i}(x_i) \oplus h_{i-1}$	1	1	–	–
21	10	$E_{h_{i-1}}(x_i) \oplus h_{i-1}$	$q/2^{l+1}$	$2q/2^{l+1} - 1$	$q/2^{l+1}$	$q/2^{l-1}$
11	11	$E_{w_i}(x_i) \oplus h_{i-1}$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
	12	$E_v(x_i) \oplus h_{i-1}$	1	1	–	–
	13	$E_{x_i}(x_i) \oplus w_i$	1	1	–	–
3	14	$E_{h_{i-1}}(x_i) \oplus w_i$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
14	15	$E_{w_i}(x_i) \oplus w_i$	$.3q(q-1)/2^n$	$q^2/2^{n-1}$	$q/2^l$	$q/2^{l-1}$
	16	$E_v(x_i) \oplus w_i$	1	1	–	–
15	17	$E_{x_i}(h_{i-1}) \oplus v$	$.3q(q-1)/2^n$	$q^2/2^{n-1}$	$.15q^2/2^n$	$9(q+3)^2/2^n$
	18	$E_{h_{i-1}}(h_{i-1}) \oplus v$	1	1	–	–
16	19	$E_{w_i}(h_{i-1}) \oplus v$	$.3q(q-1)/2^n$	$q^2/2^{n-1}$	$q/2^l$	$q/2^{l-1}$
	20	$E_v(h_{i-1}) \oplus v$	1	1	–	–
17	21	$E_{x_i}(h_{i-1}) \oplus x_i$	$.3q(q-1)/2^n$	$q^2/2^{n-1}$	$.15q^2/2^n$	$9(q+3)^2/2^n$
23	22	$E_{h_{i-1}}(h_{i-1}) \oplus x_i$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^l$	$q/2^{l-1}$
12	23	$E_{w_i}(h_{i-1}) \oplus x_i$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
35	24	$E_v(h_{i-1}) \oplus x_i$	$.3q(q-1)/2^{l-1}$	$q^2/2^{l-1}$	$.15q^2/2^l$	$q^2/2^{l-1}$
5	25	$E_{m_i}(h_{i-1}) \oplus h_{i-1}$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
	26	$E_{h_{i-1}}(h_{i-1}) \oplus h_{i-1}$	1	1	–	–
10	27	$E_{w_i}(h_{i-1}) \oplus h_{i-1}$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
	28	$E_v(h_{i-1}) \oplus h_{i-1}$	1	1	–	–
7	29	$E_{x_i}(h_{i-1}) \oplus w_i$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
24	30	$E_{h_{i-1}}(h_{i-1}) \oplus w_i$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^l$	$q/2^{l-1}$
18	31	$E_{w_i}(h_{i-1}) \oplus w_i$	$.3q(q-1)/2^n$	$q^2/2^{n-1}$	$q/2^l$	$q/2^{l-1}$
25	32	$E_v(h_{i-1}) \oplus w_i$	$.3q(q-1)/2^{l-1}$	$q^2/2^{l-1}$	$q/2^l$	$q/2^{l-1}$

Figure 4.2: Summary of results about 64 extended hash families. Column 1 is our number $\imath$ for the function family (We write $\mathcal{F}_\imath$ for the compression function family and $\mathcal{H}_\imath$ for its induced extended hash family). Column 2 is the number from [10]. Column 3 defines $f_k(h_{i-1}, m_i)$ for some $k \in \{0,1\}^l$. We write x_i for $(m_i||k)$ and w_i for $x_i \oplus h_{i-1}$. Columns 4 and 5 give our (target) collision resistance bounds. Columns 6 and 7 give our inversion resistance bounds.

$\imath$	$\jmath$	$h_i =$	**(T)CR LB**	**(T)CR UB**	**IR LB**	**IR UB**
19	33	$E_{x_i}(w_i) \oplus v$	$.3q(q-1)/2^n$	$q^2/2^{n-1}$	$.15q^2/2^n$	$9(q+3)^2/2^n$
26	34	$E_{h_{i-1}}(w_i) \oplus v$	$q/2^{l+1}$	$2q/2^{l+1}-1$	$q/2^{l+1}$	$q/2^{l-1}$
38	35	$E_{w_i}(w_i) \oplus v$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^n$	$q/2^{n-1}$
37	36	$E_v(w_i) \oplus v$	$.3q(q-1)/2^{l-1}$	$q^2/2^{l-1}$	$.15q^2/2^l$	$q^2/2^{l-1}$
20	37	$E_{x_i}(w_i) \oplus x_i$	$.3q(q-1)/2^n$	$q^2/2^{n-1}$	$.15q^2/2^n$	$9(q+3)^2/2^n$
4	38	$E_{h_{i-1}}(w_i) \oplus x_i$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
27	39	$E_{w_i}(w_i) \oplus x_i$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^l$	$q/2^{l-1}$
36	40	$E_v(w_i) \oplus x_i$	$.3q(q-1)/2^{l-1}$	$q^2/2^{l-1}$	$.15q^2/2^l$	$q^2/2^{l-1}$
8	41	$E_{x_i}(w_i) \oplus h_{i-1}$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
28	42	$E_{h_{i-1}}(w_i) \oplus h_{i-1}$	$q/2^{l+1}$	$2q/2^{l+1}-1$	$q/2^{l+1}$	$q/2^{l-1}$
29	43	$E_{w_i}(w_i) \oplus h_{i-1}$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^l$	$q/2^{l-1}$
30	44	$E_v(w_i) \oplus h_{i-1}$	$.3q(q-1)/2^{l-1}$	$q^2/2^{l-1}$	$q/2^l$	$q/2^{l-1}$
6	45	$E_{x_i}(w_i) \oplus w_i$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
2	46	$E_{h_{i-1}}(w_i) \oplus w_i$	$.3q(q-1)/2^n$	$q(q+1)/2^n$	$.4q/2^n$	$2q/2^n$
39	47	$E_{w_i}(w_i) \oplus w_i$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^n$	$q/2^{n-1}$
40	48	$E_v(w_i) \oplus w_i$	$.3q(q-1)/2^{l-1}$	$q^2/2^{l-1}$	$q/2^n$	$q/2^{n-1}$
	49	$E_{x_i}(v) \oplus v$	1	1	–	–
	50	$E_{h_{i-1}}(v) \oplus v$	1	1	–	–
41	51	$E_{w_i}(v) \oplus v$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^n$	$q/2^{n-1}$
	52	$E_v(v) \oplus v$	1	1	–	–
	53	$E_{x_i}(v) \oplus x_i$	1	1	–	–
31	54	$E_{h_{i-1}}(v) \oplus x_i$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^l$	$q/2^{l-1}$
32	55	$E_{w_i}(v) \oplus x_i$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^l$	$q/2^{l-1}$
	56	$E_v(v) \oplus x_i$	1	1	–	–
	57	$E_{x_i}(v) \oplus h_{i-1}$	1	1	–	–
	58	$E_{h_{i-1}}(v) \oplus h_{i-1}$	1	1	–	–
33	59	$E_{w_i}(v) \oplus h_{i-1}$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^l$	$q/2^{l-1}$
	60	$E_v(v) \oplus h_{i-1}$	1	1	–	–
	61	$E_{x_i}(v) \oplus w_i$	1	1	–	–
34	62	$E_{h_{i-1}}(v) \oplus w_i$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^l$	$q/2^{l-1}$
42	63	$E_{w_i}(v) \oplus w_i$	$.3q(q-1)/2^l$	$q^2/2^{l-1}$	$q/2^n$	$q/2^{n-1}$
	64	$E_v(v) \oplus w_i$	1	1	–	–

Figure 4.3: Summary of results about 64 extended hash families, continued.

Chapter 5

Multicollision Attacks on a Class of Iterated Hash Functions

5.1 Introduction

In this chapter, we discuss multicollision attacks on certain general classes of hash functions. A multicollision is a generalized notion of collision on a function. A *collision* on a function $g : D \rightarrow R$ is a doubleton subset $\{x, y\}$ of D such that $g(x) = g(y)$. For an integer $r \geq 2$, an *r-way collision* (or *multicollision*) on a function, $g(\cdot)$, is an r-subset $\{x_1, \ldots, x_r\}$ of D such that $g(x_1) = g(x_2) = \ldots = g(x_r) = z$ (say). The common output value, z, is known as the *collision value* for this r-way collision set. The *birthday attack* for r-way collisions has time complexity $\Omega(|R|^{(r-1)/r})$.

Recently, A. Joux [41] found an algorithm to construct a 2^K-way multicollision set on a classical iterated hash function, having time complexity $O(K\, 2^{n/2})$, which is a considerable improvement over the birthday attack. This multicollision attack can be considered as a weakness of the iterated hash function design because of the following reasons:

1. Joux also showed how the multicollision attack can be used to construct a collision attack, which is better than the birthday attack, on the concatenated hash function $H \parallel G$ where H is the classical hash function and G is any hash function. This concatenated hash function has often been used when large output hash values are needed.

2. There are some other practical applications where multicollision secure hash func-

tions are required. These include the micropayment scheme Micromint [84], the identification scheme of Girault and Stern [33], the signature scheme of Brickell *et al* [12], etc.

3. Multicollision secure hash function design is an interesting fundamental question, because a function having an efficient multicollision attack gives evidence of the non-randomness of the function.

Organization of the Chapter

We first give a brief introduction of Joux's attack and its applications. Then we generalize the definition of classical iterated hash functions in a natural way. After defining this class of generalized iterated hash functions, we show 2^K-way collision attacks with complexity $\mathrm{O}(n\,K^2\,2^{n/2})$. We also find multicollision attacks on a certain class of generalized tree-based hash functions, having complexity $\mathrm{O}(n\,K^2\,2^{n/2})$. Tree-based hash functions can be viewed as a parallelization of the classical iterated hash functions (which are evaluated in a sequential manner). Finally, we conclude with a brief note on multipreimage attacks and possible future work.

5.2 Joux's Multicollision Attack

In a recent paper by Joux [41], it was shown that there is a 2^r-way collision attack for the classical iterated hash function based on a compression function, $f : \{0,1\}^{m+n} \to \{0,1\}^n$, where the attack has complexity $\mathrm{O}(r\,2^{n/2})$. This complexity is much less than $\Omega(2^{\frac{n(2^r-1)}{2^r}})$, which is the complexity for the birthday attack (see Proposition 2.2).

Consider a set of vertices $V = \{0,1\}^n$. We use the notation $h \to_M h'$ (a labeled arc) to mean $f(h, M) = h'$. Here, $|h| = |h'| = n$ and $|M| = m$. Recall that the classical hash function $H^f(m_1||\ldots||m_l) = h_l$

$$h_0 \to_{m_1} h_1 \to_{m_2} h_2 \cdots h_{l-1} \to_{m_l} h_l$$

Here, h_0 is the fixed initial value, the message (or padded message) $M = m_1||\cdots m_l$, $|m_i| = m$ and h_l is the final output.

The strategy of Joux's attack is to first find r successive collisions (see Figure 5.1) by performing r successive birthday attacks, as follows:

$$\begin{aligned} f(h_0, m_1) = f(h_0, n_1) = h_1 \text{ (say), where } m_1 \neq n_1 \\ f(h_1, m_2) = f(h_1, n_2) = h_2 \text{ (say), where } m_2 \neq n_2 \\ \vdots \\ f(h_{r-1}, m_r) = f(h_{r-1}, n_r) = h_r \text{ (say), where } m_r \neq n_r. \end{aligned}$$

For $1 \leq i \leq r$, we apply BirthdayAttack$(f(h_{i-1}, \cdot), 2^{n/2}, 2)$ to find $m_i \neq n_i$ such that $f(h_{i-1}, m_i) = f(h_{i-1}, n_i)$. Thus the set

$$\{x_1 \parallel \cdots \parallel x_r : x_i = m_i \text{ or } n_i, 1 \leq i \leq r\}$$

is a 2^r-way collision set. The complexity of the attack is $\mathrm{O}(r\, 2^{n/2})$.

Figure 5.1 is a diagram illustrating the attack.

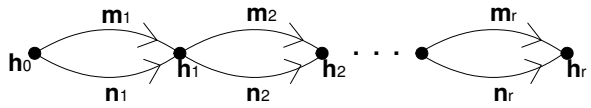

Figure 5.1: Graphical representation of Joux's multicollision attack

We summarize the above discussion as follows:

Lemma 5.1 *There is a multi-collision attack algorithm which requires $O(r2^{n/2})$ queries to find an 2^r-way collision on any n-bit classical hash function.*

Remark: Since the birthday attack is itself a probabilistic attack, there is some positive probability that Joux's attack fails. One can perform the birthday attack repeatedly until each required intermediate collision is found. If ϵ is the success probability of the birthday attack, then on the average we need r/ϵ calls of BirthdayAttack in total. In fact, if we make a total of $2r/\epsilon$ calls of BirthdayAttack, then we have an overall success probability exceeding $1/2$. This is because of the following fact which follows immediately from Markov's Inequality: if the expected value of a random variable, X, is μ, then $\Pr[X \leq 2\mu] \geq 1/2$.

5.2.1 Applications of Multicollision Attacks

The birthday attack is feasible for small sized output hash values. To make the birthday attack infeasible, one simply specifies a large output hash value. A natural and efficient approach to produce large output hash values is the concatenation of several smaller output hash values. For example, given two classical iterated hash functions, H and G, one can define a hash function $H(M) \parallel G(M)$. This concatenated hash function is efficient and simple to implement. However, due to the attacks of Joux [41], there exists a collision attack that is more efficient than the birthday attack. The complexity of the attack is roughly the maximum of the complexity of the birthday attack for collision on H and the complexity of the birthday attack for collision on G.

We briefly describe the attack (see [41] for more details). Let H and G have output hash values of n_H and n_G bits in length, respectively.

1. By using Joux's multicollision attack (see at the beginning of the Section 5.2), find $2^{n_G/2}$ messages which have common output hash value (say h^*) on H. The complexity of this step is $O(n_G 2^{n_H/2})$.

2. Find two messages, say M and N where $M \neq N$, which are members of the set of $2^{n_G/2}$ messages found in step 1, such that they have same output hash value (say g^*) on G. Note that we expect to be able to find a collision on an n_G-bit function from a set of $2^{n_G/2}$ messages using the standard birthday attack.

Thus, we have $H(M) \parallel G(M) = H(N) \parallel G(N) = h^* \parallel g^*$. The overall complexity of this attack is $\mathrm{O}(n_G\, 2^{n_H/2} + 2^{n_G/2})$. Note that we only assume that H is a classical iterated hash function; G can be any hash function at all.

Similarly, there is a preimage attack on the concatenated hash function $H \parallel G$. Let $y_1 \parallel y_2$ be a given random desired image. Then carry out the following operations.

1. First apply Joux's 2^{n_G}-way collision attack on the classical hash function H. Let h^* be the common hash value. Find a message block m such that $f(h^*, m) = y_1$ by using the birthday attack. Thus 2^{n_G} messages have common hash value y_1 for the hash function H.

2. Find a message M among these 2^{n_G} messages which have hash value y_2 for the n_G-bit hash function G. We expect a preimage of the function G from a set of queries of size 2^{n_G}. Thus, we have a message M where $H(M) \parallel G(M) = y_1 \parallel y_2$ in complexity $\mathrm{O}(n_G(2^{n_H/2} + 2^{n_G}))$.

Remark: As mentioned previously, we ignore the padding that includes the binary representation of the length of the inputs. Note that, even if we included the padding, it does not affect the above attack, as the multicollision sets consist of messages of equal length.

5.3 A General Class of Hash Functions

We have seen in Section 5.2 that the classical iterated hash function is vulnerable to a multicollision attack. Thus one cannot use the classical iterated hash function if multicollision secure hash functions are needed. There are some other disadvantages of using classical iterated hash functions. For example, very recently, Kelsey and Schneier [44] have found a generic second preimage attack that is better than the birthday attack.

To fix all these problems, one can try to use some suitable variant of the classical iterated hash function. We note that, recently, Lucks [57] designed a hash function that is secure against multicollision attack. In his construction a "wide" compression function is used. The hash function is proven to be secure if the compression function and the output transformation are both random oracles.

Alternatively, one might consider a modification of the classical iterated hash function where message blocks are used more than once. This idea was introduced by Davies and Price [22] while they proposed an DES-based message authentication scheme. They used message blocks more than once to avoid the "meet in the middle attack" [52]. Later, Coppersmith [16] and Girault et. al. [32] introduced a generalized birthday attack on the Davies and Price scheme.

Another approach is to use a parallel design, which is characterized by a directed tree; see [88]. One can also combine these two approaches.

These generalized hash functions could be considered as an alternative to the classical

iterated hash function since the Joux's attack cannot be applied. For example, the hash function $H'(M) = H(H(IV, M), M)$ uses each message block twice. Here H denotes the classical iterated hash function. We call this hash function by *doubly iterated hash function* as it uses the classical iteration twice. Obviously, Joux's attack can not be applied directly to this hash function. Thus it is worthwhile to study this class in more detail.

To begin with, we define a very general class of hash functions. Let $f : \{0,1\}^N \rightarrow \{0,1\}^n$ be a compression function. A hash function H from the class behaves in the following manner:

1. It invokes f a finite number of times.
2. The entire output of any intermediate invocation (not the final invocation) is fed into the input of other invocations of f.
3. Each bit of the message, M, is fed into at least one invocation of f.
4. The output of the final invocation is the output of the hash function, H.

We define a general class $\mathcal{D}$ of hash functions satisfying the above conditions. We will assume that our message has the form $M = m_1 \parallel \cdots \parallel m_l$, where each m_i is a message block that is an m-bit string. We also assume that we have a fixed set of *initial values*, denoted $v_1, v_2, \cdots$, each of which is an n-bit strings. Every input to f is of the form $h \parallel x$. Each h is the concatenation of r n-bit strings, each of which is a previously computed output of f or an initial value; and each x is the concatenation of q message blocks. We will require that $r \geq 0$, $q \geq 1$ and $nr + mq = N$.

Then we can specify the computation of the hash function by a list of triples

$$L = \{(h_i, x_i, y_i) : 1 \leq i \leq s\},$$

where the following conditions hold for all i:

$$\begin{aligned}
&f(h_i \parallel x_i) = y_i,\\
&h_i = h_i^1 \parallel \cdots \parallel h_i^r,\\
&h_i^j \in \{v_1, v_2, \cdots\} \cup \{y_1, \cdots, y_{i-1}\},\\
&x_i = x_i^1 \parallel \cdots \parallel x_i^q, \quad \text{and}\\
&x_i^j \in \{m_1, \cdots, m_l\}.
\end{aligned}$$

Each y_i is an intermediate hash value and y_s is the output hash value. Note that the values of r and q do not have to be constant; they may depend on i. However, $nr + mq = N$ must always hold.

In this chapter, we consider two special (but still quite general) subclasses of $\mathcal{D}$. They are termed *generalized sequential hash functions* (denoted $\mathcal{S}$) and *generalized binary tree based hash functions* (denoted $\mathcal{T}$). We show multicollision attacks on certain subclasses of $\mathcal{S}$ and $\mathcal{T}$.

Generalized Sequential Hash Functions

In the class $\mathcal{S}$ of generalized sequential hash functions, we have $r = q = 1$ for all i and $N = m + n$. Define a sequence $\alpha = \langle \alpha_1, \cdots, \alpha_s \rangle$ where $\alpha_i \in [1, l] = \{1, 2, \ldots, l\}$. Let $h_1 = v_1$ (an initial value), $h_i = y_{i-1}$ for all $i \geq 2$, and let $x_i = m_{\alpha_i}$ for all $i \geq 1$. Hence, we can express the computation in the form

$$h_{i+1} = f(h_i, m_{\alpha_i}), \, 1 \leq i \leq s,$$

where h_1 is an initial value and h_{s+1} is the final hash value.

We can present this hash function diagrammatically, as follows:

$$h_1 \rightarrow_{m_{\alpha_1}} h_2 \rightarrow_{m_{\alpha_2}} h_3 \rightarrow_{m_{\alpha_3}} \quad \cdots \quad \rightarrow_{m_{\alpha_{s-1}}} h_{s-1} \rightarrow_{m_{\alpha_s}} h_{s+1}.$$

Note that a message block can be used more than once. In the case of the classical iterated hash function, however, we have $\alpha_i = i$ for all i, and $s = l$. Also, in the classical iterated hash function, each message block is used exactly once.

Generalized Tree based Hash Functions

We also consider a class $\mathcal{T}$ of binary tree based hash functions in this section. We assume that $N = 2n$. so the compression function $f : \{0,1\}^{2n} \rightarrow \{0,1\}^n$.

We can specify the computation of the hash function on an l-block message $m_1 || \cdots || m_l$ ($|m_i| = n, 1 \leq i \leq l$) by a list of triples

$$L = \{(z_i^1, z_i^2, y_i) : 1 \leq i \leq s\},$$

where the following conditions hold:

$$f(z_i^1 \parallel z_i^2) = y_i, \quad \text{and}$$
$$z_i^1, z_i^2 \in \{v_1, v_2, \cdots\} \cup \{y_1, \cdots, y_{i-1}\} \cup \{m_1, \cdots, m_l\}.$$

We describe this class in more detail in Section 5.5.

5.4 Attacks on Generalized Sequential Hash Functions

5.4.1 Some Terminologies on Sequences

Consider a finite sequence $\alpha = \langle \alpha_1, \alpha_2, \cdots, \alpha_s \rangle$ of $[1, l]$. The *length* of the sequence is s and it is denoted by $|\alpha|$. The *index set* of the sequence is $[1, s]$. Given any subset $I = \{i_1, \cdots, i_t\}$ of the index set $[1, s]$, we have a *subsequence* $\alpha(I) = \langle \alpha_{i_1}, \cdots, \alpha_{i_t} \rangle$ where, $i_1 < \cdots < i_t$. Sometimes, we use the same notation $\alpha(I)$ to denote the multiset $\{\alpha_{i_1}, \cdots, \alpha_{i_t}\}$.

We say *subintervals* $I_1, \cdots, I_d$ form a *partition* of $I \subseteq [1, s]$ if there exists

$$1 = a_1 < b_1 + 1 = a_2 < b_2 + 1 \cdots b_{d-1} + 1 = a_d < b_d = s$$

such that $[a_i, b_i] \cap I = I_i$. The I_i's are disjoint subintervals of I and their union is the whole set I. Let $J = [a, b] \cap I$ and suppose that the minimum elements of I and J are the same. Then we say that J is a *left-end subinterval* of I.

Definition 5.1 (**Independent Elements in a Subsequence**)

Distinct elements $x_1, \ldots, x_d \in [1, l]$ are said to be independent *in the subsequence $\alpha(I)$ if there exist d subsets $I_1, \ldots, I_d$ which form a partition of I such that $x_i \in \alpha(I_i)$ but $x_i \notin \alpha(I_j)$ for $1 \leq j \neq i \leq d$.*

Observation: $x_1, \cdots, x_d$ are independent in α if and only if any subsequence containing $x_1, \cdots, x_d$ of α is of the form

$$\langle x_1, \cdots, x_1, x_2, \cdots, x_2, \cdots, x_d, \cdots, x_d \rangle.$$

$x_1, \cdots, x_t$ cannot be independent in α if there is a subsequence $\langle x_i, x_j, x_i \rangle$ of α for some $1 \leq i \neq j \leq t$; this is because any interval containing all occurrences of x_i also contains x_j.

We write

$$\mathcal{I}(\alpha(I)) := \max\{d : \exists\, d \text{ independent elements in the subsequence } \alpha(I)\}.$$

Thus, existence of d independent elements in $\alpha(I)$ implies $\mathcal{I}(\alpha(I)) \geq d$. Obviously, for any subsequence α_1, it holds that $\mathcal{I}(\alpha_1) \geq 1$. If there are k elements which appear only once in the sequence α, then $\mathcal{I}(\alpha) \geq k$.

We now consider some examples illustrating the above definition and terminologies. Later, we also show multicollision attacks on the generalized sequential hash functions based on these sequences.

Example 5.1 *Let $\vartheta_l^{(1)}$ (or simply $\vartheta^{(1)}$) $= \langle 1, 2, \ldots, l\rangle$ (the sequence for the classical iterated hash function). It is easy to note that $\mathcal{I}(\vartheta^{(1)}) = l$.*

Example 5.2 *Let $\vartheta_l^{(2)}$ (or simply $\vartheta^{(2)}$) $= \langle 1, 2, \ldots, l, 1, 2, \ldots, l\rangle$. The* doubly iterated hash function *is based on the sequence $\vartheta^{(2)}$. It is easy to observe that there are no two independent elements in the sequence $\vartheta^{(2)}$ and hence $\mathcal{I}(\vartheta^{(2)}) = 1$. But, $\mathcal{I}(\vartheta^{(2)}([1, l])) = \mathcal{I}(\vartheta^{(1)}) = l$.*

Example 5.3 *Let $\theta^2 = \langle 1, 2, 1, 2\rangle$, $\theta^3 = \langle 1, 2, 1, 3, 2, 3\rangle$ and $\theta^l = \langle\ 1, 2, 1, 3, 2, 4, 3, \cdots, l-1, l-2, l, l-1, l\rangle$, $l > 3$. Here, $\mathcal{I}(\theta^l) \geq \lfloor \frac{l+1}{2} \rfloor$ as $1, 3, \cdots, l$ (if l is odd) or $1, 3, \cdots, l-1$ (if l is even) are independent elements. To show this, consider the partition $[1, 3], [4, 7], \cdots [2l-2, 2l]$ of $[1, 2l]$. Assume that l is odd (the other case can be proved similarly). Now one can see that $1 \in \theta^l[1, 3]$, $3 \in \theta^l[4, 7]$, $\cdots$, $l \in \theta^l[2l-2, 2l]$. These elements are not appearing in other intervals. Thus $1, 3, \cdots, l$ are independent elements (see Definition 5.1) for the sequence θ^l.*

In fact, we have $\mathcal{I}(\theta^l) = \lfloor \frac{l+1}{2} \rfloor$. For any $\lfloor \frac{l+1}{2} \rfloor + 1$ elements from $[1, l]$ there are two consecutive elements (by applying the pigeonhole principle), say i and $i+1$, and hence there is a subsequence $\langle i, i+1, i\rangle$ of θ^l. Thus no $\lfloor \frac{l+1}{2} \rfloor + 1$ elements can be independent (see the observation above).

For a sequence α of $[1, l]$ and $x \in [1, l]$ we define

$$\text{freq}(x, \alpha) = |\{i : \alpha(i) = x\}|.$$

Sometimes we write this as freq(x) and we call it the *frequency of* x. This value denotes the number of times x appears in the sequence α. We also write freq(α) (*frequency of the sequence*) for the maximum frequency of any element from the sequence. More precisely,

$$\text{freq}(\alpha) = \max\{\text{freq(x)} : x \in [1, l]\}.$$

Note that, for all $1 \leq i \leq l$, $\text{freq}(i, \vartheta^{(2)}) = 2$ and $\text{freq}(i, \theta^l) = 2$. Thus $\text{freq}(\vartheta^{(2)}) = 2$ and $\text{freq}(\theta^l) = 2$. We show some multicollision attacks on sequential hash functions based on sequences whose *frequencies* are at most two.

5.4.2 Multicollision Attacks on Generalized Sequential Hash Function

For the sake of convenience, we slightly modify the notation used in the definition of the generalized sequential hash functions. Given a compression function $f : \{0,1\}^{n+m} \rightarrow \{0,1\}^n$, a fixed initial value h_0, and a sequence $\alpha = \langle \alpha_1, \cdots, \alpha_s \rangle$ on $[1, l]$, the generalized sequential hash function based on α of $[1, l]$ is defined to be $H(m_1 \parallel \cdots \parallel m_l) = h_s$, where $h_i = f(h_{i-1}, m_{\alpha_i})$ for all i, $1 \leq i \leq s$. The sequence α is called as the defining sequence of H on the set of l-block messages.

To define a hash function with arbitrary domain $(\{0,1\}^m)^*$, say $H : (\{0,1\}^m)^* \rightarrow \{0,1\}^n$, we have a sequence of sequences $\langle \alpha^1, \alpha^2, \cdots \rangle$ such that $H(M)$ is defined based on the sequence α^l, where α^l contains elements from $[1, l]$, whenever M is an l-block message.

We present a 2^r-way multicollision attack on the hash function based on a sequence α, where $\mathcal{I}(\alpha) = r$. The complexity of the attack is $O(s\, 2^{n/2})$ where $s = |\alpha|$. In case of the classical iterated hash function, the corresponding sequence is $\vartheta^{(1)}$ (see Example 5.1). Here we have $\mathcal{I}(\vartheta^{(1)}) = l$ and thus we have a 2^l-way multicollision attack with complexity $O(l\, 2^{n/2})$. This is the same as the complexity of Joux's attack. In fact, we will see that our attack is same as Joux's attack in the case of the classical iterated hash function.

The idea of the attack is to first identify some independent message blocks (the set of message blocks whose indices are independent elements in the defining sequence α), and then to find a sequence of intermediate collisions by varying only those message

blocks.

First, we illustrate our attack for the hash function based on the sequence $\theta^5 = \langle 1, 2, 1, 3, 2, 4, 3, 5, 4, 5\rangle$ (see Example 5.3). Here $1, 3, 5$ are independent elements in θ^5. The attack proceeds as follows:

1. We first fix the message blocks m_2 and m_4 arbitrarily, by defining their values to be equal to some string IV.

2. Then we find $m_1^1 \neq m_1^2$ such that

$$f(f(f(h_0, m_1^1), IV), m_1^1) = f(f(f(h_0, m_1^2), IV), m_1^2) = h_1.$$

3. Then, we find find $m_3^1 \neq m_3^2$ such that

$$f(f(f(f(h_1, m_3^1), IV), IV)m_3^1) = f(f(f(f(h_1, m_3^2), IV), IV)m_3^2) = h_2.$$

4. Finally, we find $m_5^1 \neq m_5^2$ such that

$$f(f(f(h_2, m_5^1), IV), m_5^1) = f(f(f(h_2, m_5^2), IV), m_5^2) = h_3.$$

5. Now it is easy to see that

$$C = \{m : m_i = m_i^1 \text{ or } m_i^2 \text{ for } i = 1, 3, 5, m_i = IV \text{ for } i = 2, 4\} \qquad (5.1)$$

   is a 2^3-way multicollision set with collision value h_3 (see Figure 5.2). The complexity of the attack is $\mathrm{O}(10 \times 2^{n/2})$, because $|\theta^5| = 10$.

The attack is depicted diagrammatically in Figure 5.2.

Before giving the proof of a general attack based on the ideas used in this example, we first want to introduce some additional notation and rewrite the above attack in terms of this notation. Referring to the previous attack, we introduce some notation relating to h_1, h_2 and h_3. These are the intermediate hash values where we are looking for intermediate collisions.

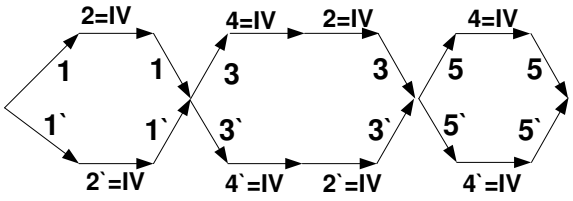

Figure 5.2: Graphical representation of Multicollision attack on the hash function based on the sequence θ^5

Consider the algorithm for computing a generalized sequential hash function. In the ith round, we have $h_i = f(h_{i-1}, m_{\alpha_i})$. We can define certain partial computations in that algorithm. Define $H(h^*, [a, b], M) = h_b$ when computing $H(M)$, given that $h_a = h^*$. Note that h_a and h_b are the intermediate hash values at rounds a and b, respectively. So, $H(h^*, [a, b], M)$ is the hash value at round b given that we start with the intermediate hash value h^* at round a.

Now, we rewrite the above-described attack in terms of the notation just defined. We have the partition $[1, 3], [4, 7]$ and $[8, 10]$ and independent elements 1, 3 and 5 (see Example 5.3). We have fixed the message blocks m_2 and m_4 to be equal to a certain string IV. Thus $H(h_0, [1, 3], M)$ depends only on the message block m_1 and hence we can write $H(h_0, [1, 3], m_1)$ instead of $H(h_0, [1, 3], M)$. Similarly, we write $H(h_1, [4, 7], m_3)$ and $H(h_2, [8, 10], m_5)$ instead of $H(h_1, [4, 7], M)$ and $H(h_2, [8, 10], M)$ respectively.

Then the 2^3-way collision attack can be described succinctly as follows :

1. Find $m_1^1 \neq m_1^2$ such that $H(h_0, [1, 3], m_1^1) = H(h_0, [1, 3], m_1^2) = h_1$ (say).
2. Find $m_2^1 \neq m_2^2$ such that $H(h_1, [4, 7], m_3^1) = H(h_1, [4, 7], m_3^2) = h_2$ (say).
3. Find $m_3^1 \neq m_3^2$ such that $H(h_2, [8, 10], m_5^1) = H(h_2, [8, 10], m_5^2) = h_3$.
4. Then C (as defined in (5.1)) is a 2^3-way multicollision set with collision value h_3.

In general, we have the following multicollision attack on a generalized sequential hash function.

Proposition 5.1 *Let H be a hash function based on a sequence $\alpha = \alpha^l$, where $\mathcal{I}(\alpha) = r$. Then we have a 2^r-way multicollision attack on H with complexity $\mathrm{O}(s\,2^{n/2})$, where $s = |\alpha|$.*

Proof. As $\mathcal{I}(\alpha) = r$, we have r independent elements $x_1, \ldots, x_r$ and r disjoint and exhaustive subintervals, $I_1, \cdots, I_r$. Now fix the message blocks m_i to be equal to an arbitrary string IV, for all $i \notin \{x_1, \ldots, x_r\}$.

Because the x_i's are independent, it follows that $H(h^*, I_i, M)$ will only depend on m_{x_i} for all i. Thus, for simplicity, we write $H(h^*, I_i, m_{x_i})$ instead of $H(h^*, I_i, M)$.

Now find r successive collisions as follows:

$$H(h_{i-1}, I_i, m^1_{x_i}) = H(h_{i-1}, I_i, m^2_{x_i}) = h_i,$$

for $1 \leq i \leq r$. Then, it is easy to check that the following set

$$\{m_1 \parallel \ldots \parallel m_l : m_{x_1} = m^1_{x_1} \text{ or } m^2_{x_1}, \ldots m_i = IV \; \forall i \notin \{x_1, \ldots, x_r\}\}$$

is a multicollision set of size 2^r.

To get the ith intermediate collision, we need to make $\mathrm{O}(|\alpha(I_i)|\, 2^{n/2})$ queries of f. For the complete attack we need $\mathrm{O}(\sum_i |\alpha(I_i)|\, 2^{n/2}) = \mathrm{O}(|\alpha|\, 2^{n/2})$ queries of f. □

Remark: Note that the above attack reduces to Joux's attack in the case of the classical iterated hash function. In this case, all the elements $1, 2, \cdots, l$ are independent and thus we find a collision for each intermediate hash value by varying each message block. This is what Joux's attack does.

To get a 2^r-way collision on the hash function based on the sequence θ^l (see Example 5.3), we can take $l = 2r - 1$. So $\mathcal{I}(\theta^l) = r$. By Proposition 5.1, we have a 2^r-way collision attack with complexity $\mathrm{O}(r\,2^{n/2})$. However, we cannot apply the same idea to the hash function based on the sequence $\vartheta^{(2)}_l$ (since $\mathcal{I}(\vartheta^{(2)}_l) = 1$, for all $l \geq 1$; see Example 5.2). Here, we have to use a different multicollision attack. This attack is summarized as follows.

1. First, we take $l = rn/2$ and we use Joux's multicollision attack to find l pairs,

$$(m^1_1, m^2_1), (m^1_2, m^2_2), \cdots, (m^1_l, m^2_l),$$

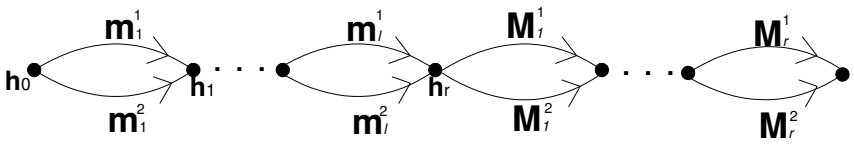

Figure 5.3: Graphical representation of multicollision attack on the hash function based on the sequence $\vartheta^{(2)}$

such that

$$f(h_{i-1}, m_i^1) = f(h_{i-1}, m_i^2) = h_i,$$

$1 \leq i \leq l$. Thus we have a 2^l-way collision set

$$\mathcal{C} = \{m : m_i = m_i^1 \text{ or } m_i = m_i^2\}$$

for the hash function based on the sequence $\vartheta^{(2)}[1, l]$.

2. Next, to get a 2^r-way multicollision for the desired hash function, we search for intermediate collisions within the set $\mathcal{C}$. Divide the index interval $[l+1, 2l]$ into r consecutive intervals, each consisting of $n/2$ elements, i.e., $I_1 = [l+1, l+n/2], \cdots, I_r = [l+1+(r-1)n/2, l+rn/2]$. Write $h'_0 = h_l$.

   Then, for each $1 \leq i \leq r$, find a pair $M_i^1 \neq M_i^2$ from the set

   $$\mathcal{C}_i = \left\{ m_{(i-1)n/2+1}^{j_1} \parallel \cdots \parallel m_{in/2}^{j_{n/2}} : j_1, \cdots j_{n/2} \in \{1, 2\} \right\}$$

   such that $H(h'_{i-1}, I_i, M_i^1) = H(h'_{i-1}, I_i, M_i^2) = h'_i$. Note that $|\mathcal{C}_i| = 2^{n/2}$ so the desired pair should exist.

3. Finally, it is easy to observe that

   $$\mathcal{C}^* = \{M_1^{j_1} \parallel \cdots \parallel M_r^{j_r} : j_1, \cdots j_r \in \{1, 2\}\}$$

   is a multicollision set (of size 2^r) for our hash function.

See Figure 5.3 for a diagrammatic representation of the attack.

Now, we prove a general result which says that for any sequence α with frequency at most two with $\mathcal{I}(\alpha(I)) \geq rn/2$ for some left end subinterval I, there is a 2^r-way collision attack with complexity $\mathrm{O}(r^2\, n\, 2^{n/2})$.

Proposition 5.2 *Let H be a hash function based on α^l with* $\mathrm{freq}(\alpha^l) \leq 2$. *If there is a left-end subinterval I such that $\mathcal{I}(\alpha^l(I)) \geq rn/2$, then there is a 2^r-way multicollision attack on H having complexity* $\mathrm{O}(r^2\, n\, 2^{n/2})$.

Proof. Let $x_1, \cdots, x_k$ be independent elements in $\alpha(I)$, where $k = rn/2$. As in Proposition 5.1, we have a set

$$\mathcal{C} = \{M = m_1 \parallel \ldots \parallel m_l; m_{x_1} = m^1_{x_1} \text{ or } m^2_{x_1}, \ldots, m_{x_k} = m^1_{x_k} \text{ or } m^2_{x_k}\}$$

of size 2^k so that $\mathcal{C}$ is a multicollision set for the hash function based on the sequence $\alpha(I)$. Let h'_0 be the collision value for the multicollision set $\mathcal{C}$. Without loss of generality, we assume that each x_i appears exactly once in the sequence $\alpha([a+1, s])$, in the same order as they appear in I, where $I = [1, a]$ and s is the length of the sequence. (If this is not the case, then the proof can be modified suitably.)

Define $\mathcal{C}_{i+1}$ for $0 \leq i \leq r-1$ as follows:

$$\mathcal{C}_{i+1} = \left\{ m^{j_1}_{x_{in/2+1}} \parallel \cdots \parallel m^{j_{n/2}}_{x_{(i+1)n/2}} : j_1, \cdots, j_{n/2} \in \{1, 2\} \right\}. \tag{5.2}$$

Now divide the interval $[a+1, s]$ into r disjoint and exhaustive subintervals $I'_1, I'_2, \cdots, I'_r$ so that $x_{in/2+1}, \cdots, x_{(i+1)n/2}$ appear in $I'_{i+1}, 0 \leq i \leq r-1$. To make the notation simpler, we ignore all other message blocks as these are fixed to be equal to a string IV. We write

$$H(h^*, I'_{i+1}, m_{x_{in/2+1}} \parallel \cdots \parallel m_{x_{(i+1)n/2}})$$

instead of

$$H(h^*, I'_{i+1}, M).$$

Note that $|\mathcal{C}_i| = 2^{n/2}$. Then we find r successive collisions:

$$H(h'_{i-1}, I'_i, M^1_i) = H(h'_{i-1}, I'_i, M^2_i) = h'_i,$$

for $1 \leq i \leq r$, where $M^1_i, M^2_i \in \mathcal{C}_i$. Now it is easy to observe that

$$\mathcal{C}^* = \{M^{j_1}_1 \parallel \cdots \parallel M^{j_r}_r : j_1, \cdots j_r \in \{1, 2\}\}$$

is a multicollision set of size 2^r. $\square$

So far, we have provided a multicollision attack if the underlying sequence satisfies certain conditions. More precisely, if $\mathcal{I}(\alpha) = r$ or if there exists an interval I such that $\mathcal{I}(\alpha(I)) = rn/2$, then there is a 2^r-way multicollision attack. Now we show that these conditions are satisfied by any sequence with a sufficient number of elements and having frequency at most two.

Definition 5.2 *Given any subsequence $\alpha(I)$ of α, we define*

$$S(\alpha(I)) = |\{x \in [1, l] : \text{freq}(x, \alpha(I)) \geq 1\}|.$$

Similarly, we can define

$$S^i(\alpha(I)) = |\{x \in [1, l] : \text{freq}(x, \alpha(I)) = i\}|.$$

So, when $\text{freq}(\alpha) \leq 2$ *we have* $S(\alpha(i)) = S^1(\alpha(i)) + S^2(\alpha(i))$.

Proposition 5.3 *Let α be a sequence of elements from $[1, l]$ with* $\text{freq}(\alpha) \leq 2$ *and $S(\alpha) = l$. Suppose that $l \geq MN$. Then one of the folllowing holds:*

1. *$\mathcal{I}(\alpha) \geq M$, or*
2. *there exists a left-end subinterval I such that $\mathcal{I}(\alpha(I)) \geq N$.*

Proof. The proof is by induction on l. Let $|\alpha| = s$. For the left-end subinterval, $I = [1, N]$, either $\mathcal{I}(\alpha(I)) \geq S^1(\alpha(I)) = N$ or there exists an element, say x_1, which appears twice in the sequence $\alpha(I)$. In the former case we are done, so assume the latter. Remove all elements from α which appear in $\alpha(I)$ and call this new sequence $\alpha_1 = \alpha(I_1)$ for some set I_1.

Note that $S(\alpha_1) \geq MN - N = (M-1)N$. By induction, either $\mathcal{I}(\alpha_1) \geq M - 1$ or there exists a left-end subinterval J of I_1, such that $\mathcal{I}(\alpha_1(J)) \geq N$. In the latter case, $\mathcal{I}(\alpha([1, r])) \geq N$, where r is the last element in the set J. In this case, we are done. In the former case there exist $M-1$ independent elements $x_2, \ldots, x_M$ in the subsequence α_1. Also x_1 does not appear in the subsequence $\alpha[N+1, s]$ and $x_2, \ldots, x_M$ do not appear in $\alpha([1, N])$. Thus, $x_1, x_2, \ldots, x_M$ are independent elements in α. $\square$

Now we have a multicollision attack for any generalized sequential hash function with frequency at most two. This is immediate from Propositions 5.1, 5.2 and 5.3.

Theorem 5.1 *Let H be a generalized sequential hash function based on the sequences $\langle \alpha^1, \alpha^2, \cdots \rangle$, where $\mathrm{freq}(\alpha^l) \leq 2$ for every $l \geq 1$. Then we have a 2^r-way multicollision attack on H with complexity $\mathrm{O}(r^2\, n\, 2^{n/2})$.*

5.5 Multicollision attacks on generalized tree-based hash functions

Similar attacks can be carried out on generalized tree based hash functions. First, we define the generalized tree based hash function and some terminology. We modify the notation somewhat to make the attacks easier to describe.

Here we consider a compression function, $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, on which an l-block generalized tree based hash function $H(\cdot)$ is defined. Suppose that $m = m_1 \parallel m_2 \parallel \cdots \parallel m_l$ is an l-block message where every block is a bit string of length n. Also, suppose that $h_1, h_2, \cdots \in \{0,1\}^n$ are constants (i.e., fixed initial values which only depend on l).

Define a list of s ordered pairs $\{(x_j^1, x_j^2)\}_{1 \leq j \leq s}$. For $1 \leq j \leq s$, we have that

$$x_j^1, x_j^2 \in \{h_1, h_2, \ldots\} \cup \{m_1, m_2, \cdots, m_l\} \cup \{z_1, \ldots, z_{j-1}\}$$

and $z_j = f(x_j^1, x_j^2)$. For $j \neq s$, the z_j's are the *intermediate hash values* and z_s is known as the final hash value. Finally, define $H(m) = z_s$.

We can assume that each intermediate hash value z_i and each message block m_j are in the list and hence they are inputs to some invocations of f. So there are no message blocks and intermediate hash values which are not hashed. The above hash function also can be defined using a directed binary tree and a MIV (*message-initial value*) assignment which will be defined soon.

Generalized Tree based Hash Functions

Let $G = (V, E)$ be a rooted directed binary tree with the set of leaves L. An *MIV assignment* is a mapping $\rho : L \to [1, l] \cup \{0,1\}^n$. If $\rho(v) \in [1, l]$ then it denotes an

index of a message block. When $\rho(v) \in \{0,1\}^n$, it denotes an initial value. Given a pair (G, ρ), we define a hash function H based on (G, ρ) as follows: For an l-block message $m = m_1 \parallel \cdots \parallel m_l$, we assign recursively an n-bit string to each vertex of G in the following manner:

1. For each leaf v, if $\rho(v) = i \in [1, l]$ then assign the n-bit string m_i to v. If $\rho(v) = h \in \{0,1\}^n$, assign h to v.
2. For any other node v, assign the n-bit string $f(z, z')$ to v, where z and z' are assigned to the vertices u and u' respectively, u is the left child of v and u' is the right child of v.
3. The output of the hash function, $H(m)$, is the value assigned to the root of the tree.

Now we define some more notation which will be used in the multicollision attack.

- For $x \in [1, l]$, we write $\text{freq}(x, G)$ (or simply $\text{freq}(x)$) for the number of times x appears in the multi-set $\rho(L)$ (this is called the *frequency* of x). That is, $\text{freq}(x)$ denotes the number of times the message block m_x is hashed to get the final output hash value. Also, define $\text{freq}(G) = \max\{\text{freq}(x) : x \in L\}$.
- We define the *hash output at vertex* v (i.e., the value assigned to v, given that the message is m) to be $H(v, m)$. Note that a message block m_i is used to compute $H(v, m)$ if and only if i is in $\rho(L[v])$ where $L[v]$ is the set of leaves of the subtree $G[v]$ rooted at v (see Section 2.2.1). We also use the notation $H(v, m_i)$ instead of $H(v, m)$ when $H(v, m)$ only depends on the ith message block, i.e., if the only index appearing in $\rho(L[v])$ is i.
- Given any vertex v, define $S(v, G)$ (or more simply, $S(v)$) to be the quantity

$$|\{x \in [1, l] : \text{freq}(x, G[v]) \geq 1\}|.$$

  Similarly, we define

$$S^i(v) = |\{x \in [1, l] : \text{freq}(x, G[v]) = i\}|.$$

  So $S(v)$ ($S^i(v)$, resp.) denotes the number of message blocks which are hashed at least once (exactly i times, resp.) to compute $H(v, m)$.

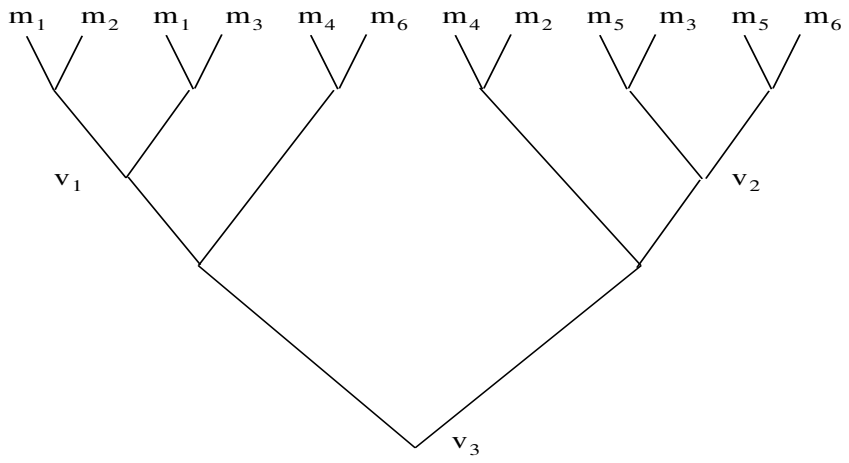

Figure 5.4: An example of 6-block binary tree based hash function.

Definition 5.3 (independent sequence of message indices) *Given a directed binary tree (G, ρ), we say that $(x_1, x_2, \ldots, x_k)$ is an* independent sequence *of message indices if there exist vertices $v_1, v_2, \ldots, v_k \in V$ such that*

1. *All occurrences of x_i are in $\rho(L[v_i])$ for all $1 \leq i \leq k$.*
2. *$x_i \notin \rho(L[v_j])$ for all $i > j$.*
3. *$v_k = q$, where q denotes the root of the directed binary tree G.*

We use the notation $\mathcal{I}(v)$ to denote the maximum value of k such that there exists an independent sequence of message indices in $G[v]$ of length k. In particular, $\mathcal{I}(q)$ denotes the maximum length of an independent sequence of message indices in the tree G. We say that v_i is the *corresponding vertex* of x_i.

Because of condition 2 in Definition 5.3, the order of independent elements is important. So $(x_2, x_1, x_3, \cdots, x_k)$ might not be an independent sequence, even if $(x_1, x_2, \cdots, x_k)$ is an independent sequence. The definition of the independent sequence coincides with independent elements in a subsequence (see Definition 5.1) if our tree has a suitable "sequential" structure.

We illustrate Definition 5.3 with a small example. In Figure 5.4, $(1, 5, 4)$ is an independent sequence. Here the corresponding vertices of $1, 5$ and 4 are v_1, v_2 and v_3,

respectively. However, note that $(4, 1, 5)$ is not an independent sequence since the only vertex v such that all occurrences of 4 are in $\rho(L[v])$ is v_3. One can also check that $(5, 4)$ is still an independent sequence in $G - G[v_1]$ and 1 does not appear in $G - G[v_1]$.

In general, we have the following lemma.

Lemma 5.2 *If $(x_1, x_2, \ldots, x_k)$ is an independent sequence in G, then $(x_2, \cdots, x_k)$ is also an independent sequence in $G - G[v_1]$, where v_1 is the corresponding vertex of x_1. Also, we have that $x_1 \notin \rho(L[G - G[v_1]])$.*

Proof. $x_1 \notin \rho(L[G - G[v_1]])$ since all occurrences of x_1 are in $\rho(L[v_1])$ (by condition 1 of Definition 5.3). Also, it is easy to check that $(x_2, \cdots, x_k)$ is an independent sequence in $G - G[v_1]$. □

Now we can state one of our main theorems of this section. It says that, given a pair (G, ρ) with r independent elements in G, there is a 2^r-way collision attack on the hash function H based on (G, ρ). The complexity of this attack is $\mathrm{O}((s+1)2^{n/2})$, where s is the number of intermediate nodes in G. The idea of the attack is very similar to that of Joux's attack. First we try to find r intermediate collision pairs $(m^1_{x_1}, m^2_{x_1}), \cdots (m^1_{x_r}, m^2_{x_r})$. Then we can combine all these pairs independently to obtain a 2^r-way collision attack.

We demonstrate the attack with the example shown in Figure 5.4.

1. First, fix the message blocks m_2, m_3 and m_6 to be an n-bit string, say IV.
2. Find $m^1_1 \neq m^2_1$ such that $H(v_1, m^1_1) = H(v_1, m^2_1) = h^*_1$ by using $3 \times 2^{n/2}$ computations of f (note that three computations of f are required to obtain a value assigned to v_1).
3. Consider the graph $G_2 = G - G[v_1]$. Find $m^1_5 \neq m^2_5$ such that $H(v_2, m^1_5) = H(v_2, m^2_5) = h^*_2$ by using $3 \times 2^{n/2}$ computations of f (note that three computations of f are required to obtain a value assigned to v_2).
4. Consider the graph $G_3 = G_2 - G[v_2]$ and the mapping $\rho_3(v_1) = h^*_1$, $\rho_3(v_2) = h^*_2$. For this pair (G_3, ρ_3), we can find $m^1_4 \neq m^2_4$ such that $H(v_3, m^1_4) = H(v_3, m^2_4) = h^*_3$ by using $5 \times 2^{n/2}$ computations of f (note that five computations of f are required to obtain a value assigned to v_3, given specified values for h^*_1 and h^*_2).

5. Now it is easy to check that the set

$$\{m : m_i = IV \text{ for } i = 2, 3, 6, m_j = m_j^1, m_j^2 \text{ for } j = 1, 4, 5\}$$

is a multicollision set with the collision value h_3^*. In this example we need $O(11 \times 2^{n/2})$ computations of f. Note that 11 is the number of non-leaf nodes (that is either an intermediate node or the root).

Note that in the above attack, if we choose $m_2 = m_6$ then the number of computation is ten. So in some cases, the number of queries is less than the number of non-leaf nodes. Now we state and prove a general theorem in detail.

Theorem 5.2 *There is a 2^r-way multicollision attack on H having complexity $O((s+1)2^{n/2})$, where s is the number of the intermediate nodes in the directed binary tree G and $\mathcal{I}(q) = r$ for the root of the binary tree, q.*

Proof. Suppose that $(x_1, \cdots, x_r)$ is an independent sequence in G. We will find a 2^r multicollision set where each m_{x_i} $(1 \le i \le r)$ takes on one of two possible values, and the other m_j's are fixed to be some value IV.

We prove the result by induction on r. Let v_i be the corresponding vertex of x_i. For $r = 1$ this is just a standard birthday attack on H, varying the message block m_{x_1} and fixing all other message blocks by a string IV.

For $r > 1$, we first define the message blocks m_i to be IV for all $i \notin \{x_1, \cdots, x_r\}$. Then we find a pair $(m_{x_1}^1, m_{x_1}^2)$ with $m_{x_1}^1 \neq m_{x_1}^2$ such that $H(v_1, m_{x_1}^1) = H(v_1, m_{x_1}^2) = h_1^*$ (say). This computation has complexity $t\, 2^{n/2}$, where $t = |V[v_1] - L[v_1]|$.

Now consider the graph $G' = G - G[v_1]$ and the MIV $\rho' : L[G'] \to [1, l] \cup \{0,1\}^n$ defined as $\rho'(v_1) = h_1^*$ and $\rho'(v) = \rho(v)$ for any other leaf v in $L[G']$. By Lemma 5.2, we know that $(x_2, \cdots, x_r)$ is an independent sequence for the hash function based on (G', ρ'). So, by induction, we can find a 2^{r-1}-way collision set

$$\{m : m_j = m_{x_i}^1 \text{ or } m_{x_i}^2 \text{ if } j = x_i, 2 \le i \le r, \text{ otherwise } m_j = IV, \text{ where } j \neq x_1\},$$

with the collision value h^* (say). This computation has complexity $O(|V' - L[G']|)$.

Note that there is no occurrence of index x_1 in the multi-set $\rho'(L[G'])$, and if the intermediate hash value at the vertex v_1 is h_1^*, then the final hash value for (G', ρ') is the same as the final hash value for (G, ρ). Hence,

$$\{m : m_j = m_{x_i}^1 \text{ or } m_{x_i}^2 \text{ if } j = x_i, 1 \le i \le r, \text{ otherwise } m_j = IV\}$$

is a 2^r-way collision set with collision value h^*. The complexity of the attack is

$$\mathrm{O}((|V' - L[G']| + |V[v_1] - L[v_1]|)2^{n/2}) = \mathrm{O}(|V - L[V]|2^{n/2}) = \mathrm{O}((s+1)2^{n/2}).$$

□

Now we prove a simple fact on directed binary trees relevant to our multicollision attack on generalized tree based hash functions. Recall that $S(v)$ denotes the number of indices which appear in $\rho(L[v])$. In the proof of the following lemma we use the notation $x \to y$ to mean that x is a child of y.

Lemma 5.3 *For any pair (G, ρ) with $S(q) \ge 2N$, there exists a vertex $v \in V$ with $N \le S(v) \le 2N$, where q is the root of the tree $G = (V, E)$.*

Proof. Let $u_1 \to v$ and $u_2 \to v$. Then it is easy to check that $S(v) \le S(u_1) + S(u_2)$. So, if $u_1 \to q$, $u_2 \to q$, then $S(u_1) + S(u_2) \ge 2N$. There will be one vertex, say u_1, with $S(u_1) \ge N$. If $S(u_1) \le 2N$, then the result follows for $v = u_1$. If not, we can continue until we reach a vertex v with $N \le S(v) \le 2N$. □

Proposition 5.4 *Let $l = |S(q)|$ where q is the root of the tree. If* $\mathrm{freq}(G) \le 2$*, then there is a vertex v such that $\mathcal{I}(v) \ge N$ or $\mathcal{I}(q) \ge M$ whenever $l \ge 2MN$.*

Proof. We prove the stated result by induction on l. For $M = 1$, the proof is trivial since $\mathcal{I}(q) \ge 1$. So assume $M > 1$. Since $S(q) \ge 2MN \ge 2N$, it follows from Lemma 5.3 that there is a vertex v such that $N \le S(v) \le 2N$. Now, if $S^1(v) = S(v) \ge N$, then $\mathcal{I}(v) \ge S^1(v) \ge N$. If $S^1(v) < S(v)$, then there is an element, say x_1, which appears exactly twice in $\rho(L[v])$ (note that $\mathrm{freq}(G) \le 2$). Let $G' = G - G[v]$. After we choose an index x_1 in $\rho(L[v])$, we want to make sure that no x_i $(i > 1)$ that is chosen later on also occurs in $\rho(L[v])$. To prevent this from happening, we take all indices of

message blocks in $\rho(L[v])$ and "remove" them from any other leaves in the graph, by fixing their values, before applying the inductive hypothesis. Formally, we define $\rho'(u)$ to be an n-bit string, where $u \in \rho(L[v]) \cap \rho(L[G'])$; otherwise, $\rho'(u) = \rho(u)$. Note that $S(G') \geq 2MN - 2N = 2(M-1)N$.

By the induction hypothesis on the graph G', either $\mathcal{I}(q) \geq M-1$ or there exists a vertex u such that $\mathcal{I}(u) \geq N$. In the latter case, $\mathcal{I}(u) \geq N$ (for the graph G). Otherwise there exist $M-1$ independent elements, $x_2, \ldots, x_M$, in the graph G'. Also, x_1 does not appear in $\rho(L[G'])$ and $x_2, \cdots, x_M$ do not appear in $\rho(L[v])$. So, $x_1, x_2, \ldots, x_M$ are independent elements in G. □

Whenever $l \geq 2r^2 n$ either $\mathcal{I}(q) \geq r$ or there is a vertex v such that $\mathcal{I}(v) \geq rn = k$ (say). In the former case, we already have a 2^r-way collision attack. In the latter case, we can do the same thing that we did in the sequential case: Let $(x_1, \cdots, x_k)$ be an independent sequence. Find r vertices $v_1, v_2, \cdots, v_r = q$ in G' $(=G - G[v])$ such that the following occurs:

1. $x_{in+1}, x_{in+2}, \cdots, x_{in+n/2} \in \rho(L(G'[v_i]))$ for all i.
2. $x_{in+1}, x_{in+2}, \cdots, x_{in+n/2} \notin \rho(L(G'[v_j]))$ for all $j < i$.

First, we find a 2^k-way collision on v. Then, we find r successive collisions from the multicollision set. The idea of the attack is very similar with that of the sequential case, so we ignore the details. Our main theorem is as follows.

Theorem 5.3 *If* $\mathrm{freq}(G(H)) \leq 2$ *then we have a* 2^r*-way multicollision attack having complexity* $\mathrm{O}(r^2\, n\, 2^{n/2})$.

5.5.1 A Note on Multi-Preimage Attacks

For the sake of completeness, we briefly study the corresponding multi-preimage attacks on generalized sequential or generalized tree-based hash functions. For a hash function $H : \{0,1\}^* \to \{0,1\}^n$, we define the *$r$-way preimage (multi-preimage) attack* as follows: Given a random $y \in \{0,1\}^n$, find a subset $\mathcal{C} = \{x_1, \cdots, x_r\}$ of size r (≥ 1) such that $H(x_1) = \cdots = H(x_r) = y$. The complexity of an r-way preimage attack for a random

oracle is $\Omega(r\,2^n)$. On the other hand, for a generalized tree based or sequential hash function there is an r-way preimage attack with complexity $\mathrm{O}(2^n)$. The attack is almost same as the multicollision attack. It starts out exactly the same as the multicollision attack. At the final step, instead of finding a collision, we instead look for outputs having a given value y. The complexity for last step is $\mathrm{O}(2^n)$ which will dominate the $O(r^2\,n\,2^{n/2})$ complexity of the remaining steps in the attack.

5.6 Conclusion

Recently there have been many proposed approaches to design hash functions with large output hash values. The most simple and natural one is the concatenation of two classical iterated hash functions. Joux [41] showed some collision (and preimage) attacks on this hash function. It is an interesting question to find design techniques for hash functions for which multicollision attacks are infeasible. In this paper, we have defined a large natural class of hash functions and we studied their security with respect to multicollision attacks. Unfortunately, we have found efficient multicollision attacks on the hash functions when the message blocks are processed at most twice. So the natural question that arises is if multicollision security can be obtained if the message blocks are used more than twice. One can also search for some other designs outside this class of hash functions, and study their multicollision security.

Chapter 6

Designs of Efficient Secure Large Hash Values

6.1 Introduction

In this chapter, several double length compression functions and double length hash functions based on single length compression functions have been designed. More precisely the following problem is being considered:

Problem : Given a compression function, $f : \{0,1\}^{n+m} \rightarrow \{0,1\}^n$ (or s compression functions $f_1, \cdots, f_s : \{0,1\}^{n+m} \rightarrow \{0,1\}^n$), $m > 0$, how to design a compression function $F : \{0,1\}^N \rightarrow \{0,1\}^{2n}$ and a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^{2n}$, where $N > 2n$.

An n-bit compression function or hash function is said to be maximally secure if the best collision attack requires $\Omega(2^{n/2})$ many queries (same as that of the birthday attack). To increase the security level, one can design $2n$-bit compression functions and hash functions (also known as *double length compressions or hash functions respectively*). Theoretically as well as practically, it would be interesting to design double length compression (or hash) functions and analyze it's security level.

A natural approach to design a double length hash function is the method of concatenation, that is, $H \parallel G$, where both H and G are n-bits hash functions. Unfortunately, this hash function is not secure as shown in [41] (also see Chapter 5). There were several attempts to construct a secure block cipher based double length compression functions. See Sect. 2.5.4 for more detailed discussion.

Organization of the Chapter

In this chapter we design several new double length hash functions and compute their security level and the rate. Our first design is a generalization of Lucks's [57] and Hirose's [37] constructions. Given a permutations $p(\cdot)$ on the set of all N-bit strings and a compression function, $f : \{0,1\}^N \to \{0,1\}^n$, define $f^p(X) = f(X)||f(p(X))$. We show that the double length function f^p is maximally secure provided the permutation p does not have any fixed point (see Sect. 6.3).

Next, we study the security level for the double length hash function defined by the classical iteration of a compression function f^p defined as above. We show that, along with secure compression functions there are many more compression functions which extend to secure double length hash functions. Thus, we have a wide class of maximally secure double length hash functions. Lucks and Hirose's construction belongs to this class.

Then we design two efficient double length compression functions. The first one is based on three independent compression functions and it has collision security level as $\Omega(2^{2n/3})$. The next construction is similar to the concatenated hash function. We show the collision security (or pre-image security) level of the double length hash function $\Omega(2^n/in^{i-1})$ (or $\Omega(2^{2n}/in^{i-1})$ respectively), where i is some parameter which also determines the efficiency of the hash function.

Finally, we propose a double length hash function which is the best known efficient double length hash function and we leave the security level of the hash function as an open problem.

6.2 Rate or Efficiency of a Double Length Hash Function

Let $f_1, f_2, \cdots f_k : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ be underlying compression functions. We design a double length compression function, $F : \{0,1\}^N \to \{0,1\}^{2n}$, based on $f_1, f_2, \cdots f_k$. A measurement of the efficiency of the compression function, $F(\cdot)$, known as the rate function of F, is defined as follows:

Definition 6.1 (Rate Function)

Let a double length compression function, F, be based on $f_1, \cdots, f_k$. Define the rate function of F by $\frac{N-2n}{m \times s}$, where s is the number of invocations of all f_i's required to compute $F(X)$, $X \in \{0,1\}^N$. When the rate function is constant, the term "rate" is used instead of rate function.

Roughly, it says the number of message blocks hashed per underlying compression function. By a message block, the size of hashed message of underlying compression functions is meant. Thus, a message block has size m, since $f_1, f_2, \cdots, f_k : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$.

A classical hash function, denoted by H^F, based on a compression function, $F : \{0,1\}^N \rightarrow \{0,1\}^{2n}$, is defined as follows:

- Let $M = M_1 || \cdots || M_l$ be a padded message, $|M_i| = N - 2n$.

- $H^F(M) = H_l$, where $H_i = f(H_{i-1}||M_i)$, $i \geq 1$ and H_0 is a fixed $2n$ bit initial value.

Rate functions of some compression functions are being computed in the following examples.

Example 6.1 *Let $f_1, f_2 : \{0,1\}^{n+m} \rightarrow \{0,1\}^n$ be compression functions. The underlying double length compression function, $F : \{0,1\}^{2n+m} \rightarrow \{0,1\}^{2n}$, of the concatenated hash function $H^{f_1}||H^{f_2}$ is $F(H_1, H_2, M) = f_1(H_1, M)||f_2(H_2, M)$, where $|H_1| = |H_2| = n$ and $|M| = m$. Here, $N = 2n + m$, $s = 2$ and size of message block is m. Thus rate function of this double length compression function is $(N - 2n)/(m \times 2) = 1/2$.*

Example 6.2 *Let $F(X) = f_1(X)||f_2(X)$ be a compression function with domain $\{0,1\}^{n+m}$. Here, the rate function is $\frac{n+m-2n}{2m} = \frac{1}{2} - \frac{n}{2m}$. Similarly, rate function of $f^p(X) = f(X) \parallel f(p(X))$ is $\frac{1}{2} - \frac{n}{2m}$, where p is a permutation on the set of $(n+m)$-bit strings and $m > n$. When $m = 2n$, the rate of the compression function is $\frac{1}{4}$.*

Example 6.3 *Consider a compression function (see Sect. 6.4) [72]. Let $f_i : \{0,1\}^{2n} \rightarrow \{0,1\}^n$ be three underlying compression functions, $i = 1, 2, 3$. Define, $F : \{0,1\}^{3n} \rightarrow \{0,1\}^{2n}$, where $F(x, y, z) = (f_1(x, y) \oplus f_2(y, z)) \parallel (f_2(y, z) \oplus f_3(z, x))$ with $|x| = |y| = |z| = n$. The rate of this compression function is $1/3$.*

6.3 Double Length Compression Functions

6.3.1 Double length hash function from a single compression function

Given a random function $f : \{0,1\}^{n+m} \rightarrow \{0,1\}^n$, $m > n+1$, two independent random function $f_0, f_1 : \{0,1\}^{n+m-1} \rightarrow \{0,1\}^n$ such that $f_i(X) = f(i||X)$ where $i = 0$ or 1 (see Sect. 2.3.3) can be defined. Now, the double length compression function $F(X) = f_0(X)||f_1(X)$ is maximally secure since F itself is a random function.

There can be other way to define a large hash value using a compression function. Define a double length compression functions $F : \{0,1\}^{n+m} \rightarrow \{0,1\}^{2n}$, $n > m$, as follows; $F(X) = f(X) \,||\, f(\bar{X})$, where $\bar{X}$ denotes the string obtained by making bitwise complement. Note that, $\bar{X}$ is a permutation. One can also consider any permutation instead of complement. In Sect. 6.3.2 security analysis of double length compression function, $f^p(X) = f(X)||f(p(X))$, is given.

Let $\mathcal{Q} = \{(x_1, y_1), \cdots, (x_q, y_q)\}$ be a view of an adversary. Thus, $f(x_i) = y_i$, $1 \leq i \leq q$. Now there are at most q messages, namely $X_i = x_i, 1 \leq i \leq q$, such that the values of $F(X_i)$'s are known from the queries. X_i is said to be a *computable message* of F with respect to the set of queries. Note that, the final collision pair should be a pair of computable messages. Now for any two computable messages $X \neq Y$,

- if $Y \neq \bar{X}$ then $\Pr[F(X) = F(Y)] = \Pr[f(X) = f(Y)\ , f(\bar{X}) = f(\bar{Y})] = 1/2^{2n}$ and

-if $Y = \bar{X}$ then $\Pr[F(X) = F(Y)] = 1/2^n$.

Hence the probability of the event that there is a collision pair obtained from the view is at most $q(q-1)/2^{2n+1} + q/2^n$. The first term is due to the collision pair of the form $\{X, Y\}$, where $Y \neq \bar{X}$ and the second term is due to a collision pair of the form $\{X, \bar{X}\}$. Thus, to have a significant probability we need to have $\Theta(2^n)$ many queries.

6.3.2 A Class of Permutation based Double Length Compression Functions

Fix a compression function, $f : \{0,1\}^{n+m} \rightarrow \{0,1\}^n$, $m > n$. For each permutation p on the set of $n+m$ bits, define $f^p : \{0,1\}^{n+m} \rightarrow \{0,1\}^{2n}$, where $f^p(X) = f(X)||f(p(X))$. Thus, a class of double length compression functions is defined as

$$\mathcal{C} = \{f^p = f(\cdot) \parallel f(p(\cdot)) : p \text{ is a simple permutation on } \{0,1\}^{n+m}\}.$$

A permutation is a simple permutation if both p and p^{-1} are easy to compute. The compression function is depicted in Fig. 6.1.

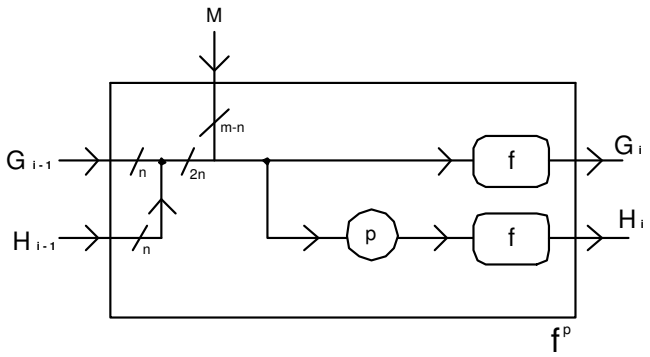

Figure 6.1: A double length compression function

One can also consider a compression function

$$f^{p_1,p_2}(X) = f(p_1(X))||f(p_2(X)), \quad \text{where } |X| = n+m$$

for two simple permutations p_1 and p_2. Since p_1 and p_2 are simple, it is enough to study the security properties of f^p where $p = p_2 \circ p_1^{-1}$. All these compression functions have rate $\frac{1}{2} - \frac{n}{2m}$ (as in Example 6.2). In this section, we study the security properties of the compression functions from the class $\mathcal{C}$ in the random oracle model of f.

Example 6.4 *[57] $p(H_1, H_2, M) = H_2||H_1||M$, where $|H_1| = |H_2| = n$ and $|M| = m - n$. Thus $f^p(H_1, H_2, M) = f(H_1, H_2, M)||f(H_2, H_1, M)$. By using the birthday attack, find H, G and M_1, M_2, such that $(H, M_1) \neq (G, M_2)$ and$f(H, H, M_1) =$*

$f(G, G, M_2)$. Now, it is easy to check that $f^p(H, H, M_1) = f^p(G, G, M_2)$. The complexity of the attack is $O(2^{n/2})$.

The reason for having the above attack is that the permutation p has many "fixed points". X is called a *fixed point* of a function $p(\cdot)$, if $p(X) = X$. The set of all fixed points of p is denoted by $\mathcal{F}_p$. In the above example, $\mathcal{F}_p = \{H||H||M : |H| = n, |M| = m - n\}$ is the set of fixed points of the permutation p and $|\mathcal{F}_p| > 2^n$. Thus, one can apply birthday attack to find a collision (or a preimage) on the compression function f from the fixed point set. Similar attack can be done for any compression function based on a permutation with more than 2^n many fixed points. In the light of the above discussion, one should use a permutation p, which does not have many fixed points. In fact, there are many permutations where the set of fixed points are the empty set. For example,

Example 6.5 *Let A be a non-zero N-bit string. Then define a permutation $p : \{0,1\}^N \rightarrow \{0,1\}^N$ such that $p(X) = X \oplus A$. In particular, if $A = 11\cdots1$ then $p(M) = \overline{M}$, where $\overline{M}$ is the bit-wise complement of M. It is easy to check that $\mathcal{F}_p$ is the empty set.*

Example 6.6 *Any N-bit string can be thought of as an integer modulo 2^N. Let $p(X) = X + A (mod\ 2^N)$ where $A \neq 0$. For simplicity, $X + A$ is used to denote the modulo addition $X + A (mod\ 2^N)$. Note that, $p(X) \neq X$ for all X. Moreover, if $A \neq 2^{N-1}$ then $p(p(X)) = X + 2A \neq X$. Thus, the set of fixed point for $p \circ p$ (in notation, p^2) is also empty.*

Suppose, f^p is a double length compression function based on a permutation p, where $\mathcal{F}_p$ is the empty set. Then a collision, $f^p(X) = f^p(Y)$ with $X \neq Y$ implies $f(X) = f(Y)$ and $f(p(X)) = f(p(Y))$, where $X \neq Y$. Thus, $\{X, Y\}$ and $\{p(X), p(Y)\}$ are collision sets of f. Now, we have the following two cases.

- **Case-1 :** $\{X, Y\} = \{p(X), p(Y)\}$. Since p does not have any fixed point, $Y = p(X)$ and $X = p(Y)$. Thus, $\{X, p(X)\}$ is a collision set, where $p(p(X)) = X$. Let $\Omega(K_1(n))$ (or in short K_1) be the complexity of the best attack to find a collision set of the form $\{X, p(X)\}$ with $p^2(X) = X$. If p^2 does not have any fixed point as in Example 6.6 then this case does not arise.

- **Case-2 :** $\{X, Y\} \neq \{p(X), p(Y)\}$. In this case, let $\Omega(K_2(n))$ (or in short K_2) be the complexity of the best attack to find two distinct collision sets of the form $\{X, Y\}$ and $\{p(X), p(Y)\}$.

Thus a collision on f^p reduces to one of the above two events and hence the complexity of best collision attack is $\min\{K_1, K_2\}$. If p^2 does not have any fixed point then we can exclude the first case also. We summarize the above discussion as

Proposition 6.1 *The complexity of the best collision attack on f^p is* $\min\{\Omega(K_1(n)), \Omega(K_2(n))\}$ *where p is a permutation with no fixed point and K_1 and K_2 are defined as above. Moreover, if the permutation p^2 does not have any fixed point (like in the Example 6.6) then the best collision attack on f^p is $\Omega(K_2(n))$.*

Now we give some evidences why K_1 and K_2 would be large for a good compression function f. Suppose an adversary tries to find two collision sets $\{X, Y\} \neq \{p(X), p(Y)\}$. After finding a collision set $\{X, Y\}$, he does not have any freedom to choose for the second collision set and he is forced to check whether $\{p(X), p(Y)\}$ is a collision set or not. Thus K_2 would be large and may be close to 2^n for a good underlying compression function. Next, an adversary tries to find a collision set $\{X, p(X)\}$. After fixing one message X, $p(X)$ is completely determined (and also vice-versa) and hence the adversary has to check equality of two values, $f(X)$ and $f(p(X))$, instead of comparing several values like in the birthday attack. Thus, K_1 is expected to be large. In the random oracle model of f, it can be proved that, $K_1(n) = K_2(n) = 2^n$

Theorem 6.1 *Under the assumption of the random oracle model of f, $K_1(n) = K_2(n) = 2^n$. Thus, for any permutation p where $\mathcal{F}_p$ is the empty set, any attack algorithm for finding collision requires $\Omega(2^n)$ many queries of f in the random oracle model of f.*

A set of pairs, $\{(x_1, y_1), \cdots, (x_q, y_q)\}$ is called *view* of a function f, if $f(x_i) = y_i$, $1 \leq i \leq q$. Similarly, $\mathcal{Q} = (\mathcal{Q}_1, \cdots, \mathcal{Q}_k)$ is called view of $f_1, \cdots, f_k$, where $\mathcal{Q}_i$ is the view of f_i. Intuitively, a computable message, X of F is a message so that $F(X)$ can be computed from view.

Definition 6.2 (Computable Message)

Let the double length compression function F be based on the compression functions, $f_1, \cdots, f_k$. Let $\mathcal{Q}_j = \{(x_1^j, y_1^j), \cdots, (x_{q_j}^j, y_{q_j}^j)\}$ be the view *of f_j, $1 \leq j \leq k$. Let $\mathcal{Q} = (\mathcal{Q}_1, \cdots, \mathcal{Q}_k)$ be the* view *of the underlying compression functions $f_1, \cdots, f_k$. An input X is said to be a* computable message *of F with respect to the view $\mathcal{Q}$, if the value of $F(X)$ can be computed from $\mathcal{Q}$.*

For example, when $F = f^p$, an input X is computable message of F with respect to $\{(x_1, y_1), \cdots, (x_q, y_q)\}$, view of f, if $X = x_i$ and $p(X) = x_j$ for some $i, j \in [1, q]$. Thus, $f^p(X) = f(x_i)||f(x_j) = y_i||y_j$, which can be computed from $\mathcal{Q}$.

Proof of Theorem 6.1: Since f is a random function, for $\{X, Y\} \neq \{p(X), p(Y)\}$,

$$\begin{aligned}
&\Pr\ [\ f(X) = f(Y) \text{ and } f(p(X)) = f(p(Y))\ ] \\
&= \Pr\ [\ f(p(X)) = f(p(Y))\ \mid\ f(X) = f(Y)\ ] \times \Pr\ [\ f(X) = f(Y)\ ] \\
&= \Pr\ [\ f(p(X)) = f(p(Y))\ ] \times \Pr\ [\ f(X) = f(Y)\ ] \\
&= 1/2^{2n}.
\end{aligned}$$

Since $\{X, Y\} \neq \{p(X), p(Y)\}$, one of $p(X)$ and $p(Y)$ does not belong to $\{X, Y\}$. Without loss of generality, assume that $p(X) \notin \{X, Y\}$. Then $f(p(X))$ is uniformly distributed given $(f(p(y)), f(X), f(Y))$. Thus, the conditional probability $\Pr[f(p(x)) = f(p(Y)) \mid f(p(Y)), f(X), f(Y)]$ is $1/2^n$ and hence so is the unconditional probability. This explains the second equality.

If an adversary can ask at most q queries, then he can have at most q computable messages and hence at most $\binom{q}{2}$ 2-sets $\{X, Y\}$. Hence the probability that the adversary finds $X \neq Y$ with $\{X, Y\} \neq \{p(X), p(Y)\}$ such that $f(X) = f(Y)$ and $f(p(X)) = f(p(Y))$ is at most $\binom{q}{2}/2^{2n}$. Thus, to have a significant success probability, q should be $\Omega(2^n)$.

Similarly, from a set of q queries one can get $\mathrm{O}(q)$ many pairs of the form $(X, p(X))$, where X and $p(X)$ both are computable. For fixed X, $\Pr[f(X) = f(p(X))] = 1/2^n$ provided $p(X) \neq X$. Thus success probability is at most $q/2^n$ and hence $q = \Omega(2^n)$ for significant success probability. □

6.3.3 A Class of Secure Double Length Hash Functions

Now we study the double length hash functions defined by the classical iteration of the compression functions f^p stated in Sect. 6.3.

Definition 6.3 *Let p be a permutation on the set of $(n+m)$-bits strings, $m \geq n$. Define $\mathcal{F}_p[2n] = \{Z \in \{0,1\}^{2n} : \exists\ M \in \{0,1\}^{m-n}$ such that $Z||M \in \mathcal{F}_p\ \}$. That is, it is a first $2n$ bit projection on $\mathcal{F}_p$. The permutation $p(\cdot)$ is called* good *if $2^{2n}/|\mathcal{F}_p[2n]| = \Omega(2^n)$. In other words, $|\mathcal{F}_p[2n]| < 2^n$.*

Now we define the following attack. Find M and $H \notin \mathcal{F}_p[2n]$, such that $f^p(H, M) \in \mathcal{F}_p[2n]$, where $|M| = m - n$ and $|H| = 2n$. Let the complexity of the best attack be $\Omega(K_3(n))$ (or in short K_3).

Theorem 6.2 *The classical hash function H^{f^p}, based on a good permutation and an initial value $H_0 \notin \mathcal{F}_p[2n]$ has collision security* $\min\{K_1, K_2, K_3\}$. *For a good permutation p and random compression function f, $K_3(n) = 2^n$. Thus, in the random oracle model of f, H^{f^p} is maximally secure against collision attack for a good permutation $p(\cdot)$.*

Proof. Let (M, M') be a collision on H^{f^p} and $H_0 \notin \mathcal{F}_p[2n]$. We denote H_i and G_i for internal hash values while computing the final hash value for messages $M = M_1||M_2\cdots$ and $M' = M'_1||M'_2\cdots$ respectively. Now we have one of the following :

1. There is an i such that $H_i \notin \mathcal{F}_p[2n]$ but $f^p(H_i||M_{i+1}) \in \mathcal{F}_p[2n]$ or there is a j such that $G_j \notin \mathcal{F}_p[2n]$ but $f^p(G_j||M'_{j+1}) \in \mathcal{F}_p[2n]$. To achieve this we need $\Omega(K_3)$ many queries.

2. There are $H_i, G_j \notin \mathcal{F}_p[2n]$ such that $X = (H_i, M_{i+1}) \neq (G_j, M'_{j+1}) = Y$ and $f^p(X) = f^p(Y)$. Since $H_i, G_j \notin \mathcal{F}_p[2n]$, $p(X) \neq X$ and $p(Y) \neq Y$. Thus either $\{X, Y\}$ and $\{p(X), p(Y)\}$ are two different collision sets or $\{X, p(X)\}$ is a collision set for the compression function f. To achieve this, adversary requires $\min\{K_1, K_2\}$ many queries of f.

Thus the adversary needs $\min\{K_1, K_2, K_3\}$ complexity. We have already seen that after q many queries the adversary can have at most q many computable messages for f^p. Given a computable message $H||M$ with $H \notin \mathcal{F}_p[2n]$, we have $p(H||M) \neq H||M$ and hence $f^p(H||M)$ is uniformly distributed over the set $\{0,1\}^{2n}$. But $|\mathcal{F}_p[2n]| < 2^n$ since the permutation $p(\cdot)$ is good. Thus we have, $\Pr[f^p(H||M) \in \mathcal{F}_p[2n]] \leq 1/2^n$. Since we have at most q computable messages, the success probability of the adversary is less than $q/2^n$. This proves the fact that $K_3(n) = 2^n$ under the random oracle model of $f(\cdot)$. By Theorem 6.1, H^{f^p} is maximally secure under the random oracle model of f. □

6.4 A Rate 1/3 Secure Double Length Compression Function

In this section, we design a double length compression function by using 3 invocations of the underlying compression functions. The rate of this compression function is 1/3 as we compress a message block for each invocation of the double length compression function and we use three invocations of the underlying compression functions. Thus, it is more efficient than the constructions given in Section 6.3. Unfortunately, it is not maximally secure as we have an attack better than the birthday attack. But we prove that, under random oracle model it has good security which is much more than security level of a most secure single length compression function. More precisely, any adversary in the random oracle model needs $\Omega(2^{2n/3})$ queries of the underlying compression functions to find collision. Thus, we increase the security level of the underlying compression function significantly.

The design of the double length compression function

We first illustrate some statistical notations and properties. We write $X \models U_N$ if the probability distribution of X is uniformly distributed on the set $\{0,1\}^N$. We denote U_N by the uniform random variable on $\{0,1\}^N$.

Proposition 6.2 *If $X = (X_1, \cdots, X_k) \models U_{kn}$ then for any vector $l \in Z_2^k$ with $l \neq 0$, the random variable $X \cdot l^T \models U_n$, X_i is an n-bit random variable. For any matrix $L_{k\times r}$ with rank $r(\leq k)$, the random vector $X \cdot L \models U_{nr}$.*

Proof. For any $y = (y_1, \cdots, y_r) \in \{0,1\}^r$, the size of the set $\{x \in \{0,1\}^k : xL = y\}$ has size 2^{k-r} since the matrix L has full column rank r. Thus $\Pr[Y_1 = y_1, \cdots Y_r = y_r]$ is equal to $1/2^r$. □

Example 6.7 *Take $r = 2$ and $k = 3$. Let $L = [l_1^T, l_2^T]$ $l_1 = (0,1,1)$ and $l_2 = (1,1,0)$ then $X \cdot L = (X_2 \oplus X_3, X_1 \oplus X_2)$, where $X = (X_1, X_2, X_3) \models U_{3n}$. By Proposition 6.2, both $X_2 \oplus X_3$ and $X_1 \oplus X_2$ are independently and uniformly distributed on $\{0,1\}^n$ since L has rank 2.*

Let $f_i : \{0,1\}^{2n} \to \{0,1\}^n$ be independent random functions, $i = 1,2,3$. Define, $F : \{0,1\}^{3n} \to \{0,1\}^{2n}$, where $F(x,y,z) = (f_1(x,y) \oplus f_2(y,z)) \,||\, (f_2(y,z) \oplus f_3(z,x))$ with $|x| = |y| = |z| = n$. We also write $F = F_1 \,||\, F_2$, where $F_1(x,y,z) = f_1(x,y) \oplus f_2(y,z)$ and $F_2(x,y,z) = f_2(y,z) \oplus f_3(z,x)$ (see Figure 6.2). The rate of the compression function is $n/3n = 1/3$.

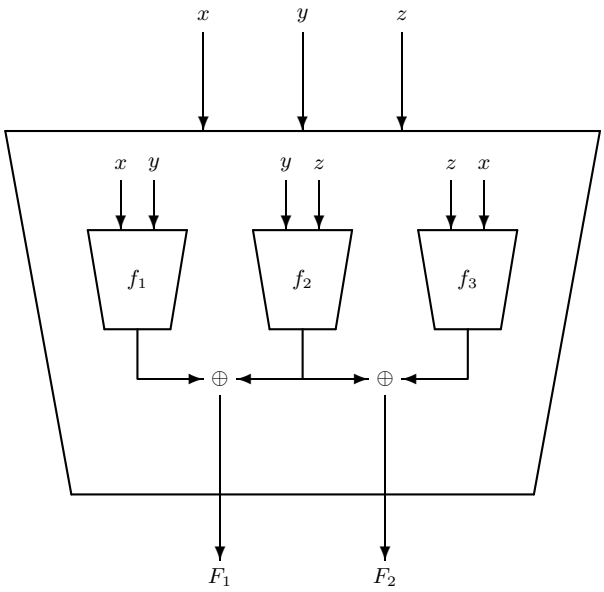

Figure 6.2: A double length compression function

Theorem 6.3 *$(F(x_1,y_1,z_1), F(x_2,y_2,z_2)) \models U_{4n}$, $(x_1,y_1,z_1) \neq (x_2,y_2,z_2)$. In particular, $\forall M \neq N$ and Z, $Pr[F(M) = F(N)] = \frac{1}{2^{2n}}$ and $Pr[F(M) = Z] = \frac{1}{2^{2n}}$.*

Proof. Let $M = (x_1, y_1, z_1) \neq (x_2, y_2, z_2) = N$. Assume that $x_1 \neq x_2$, $y_1 = y_2 = y$ (say), and $z_1 = z_2 = z$ (say). For the other cases, we can prove the result similarly. Since f_1, f_2 and f_3 are independent random functions and $x_1 \neq x_2$, $f_1(x_1, y), f_1(x_2, y), f_2(y, z), f_3(z, x_1)$ and $f_3(z, x_2)$ are independently distributed. Thus, by Proposition 6.2, we know that $f_1(x_1, y) \oplus f_2(y, z)$, $f_3(z, x_1) \oplus f_2(y, z)$, $f_1(x_2, y) \oplus f_2(y, z)$ and $f_3(z, x_2) \oplus f_2(y, z)$ are independently distributed as it is a linearly independent combination. □

Definition 6.4 (Computable message)

Let $\mathcal{Q}_1 = \{(x_i^1, y_i^1)\}_{1 \leq i \leq q}$, $\mathcal{Q}_2 = \{(y_i^2, z_i^2)\}_{1 \leq i \leq q}$ *and* $\mathcal{Q}_3 = \{(z_i^3, x_i^3)\}_{1 \leq i \leq q}$ *be the three sets of queries for the random oracles* f_1,f_2 *and* f_3*, respectively. We say a message* $M = (x, y, z)$ *is* computable *if* $(x, y) \in \mathcal{Q}_1, (y, z) \in \mathcal{Q}_2$ *and* $(z, x) \in \mathcal{Q}_3$.

Thus it is easy to observe that a message M is *computable* if and only if $F(M)$ can be computed from the set of queries. Because of Theorem 6.3 of this section if we can bound the number of computable message by some number say Q then it is easy to check that the adversary will get a collision with probability at most $Q(Q-1)/2^{2n+1}$. Thus the question reduces how to get an upper bound of the number of computable messages from any set of queries $\mathcal{Q}_1, \mathcal{Q}_2$ and $\mathcal{Q}_3$ where $|\mathcal{Q}_i| \leq q$, $1 \leq i \leq 3$. To have an upper bound we can convert our problem into a combinatorial graph theoretical problem.

A Combinatorial Graph Theoretical Problem

Recall that a graph $G = (V, E)$ is known as a tripartite graph if $V = A \sqcup B \sqcup C$ (disjoint union) and for any edge $\{u, v\} \in E$ either $u \in A, v \in B$ or $u \in A, v \in C$ or $u \in B, v \in C$ (see Figure 2.1). We use the notation $e(A, B)$ for the set of edges between A and B. Similarly we can define $e(B, C)$ and $e(A, C)$. Note that for every triangle $\triangle$ in G, the vertices of $\triangle$ are from A, B and C with one vertex from each one. Now we can state the following problem.

Problem : Given an integer q, what is the maximum number of triangles of a tripartite graph G on $A \sqcup B \sqcup C$ such that $|e(A, B)|, |e(B, C)|, |e(A, C)| \leq q$.

Theorem 6.4 *Given a positive integer n, the number of triangles of any tripartite graph G on $A \sqcup B \sqcup C$ is at most $(XYZ)^{1/2}$ such that, $|e(A,B)| \leq X, |e(A,C)| \leq Y$ and $|e(B,C)| \leq Z$. In particular, when $X = Y = Z = n^2$ the number of triangle is at most n^3.*

Proof. Let x_a be the number of edges from the vertex $a \in A$ between A and B. Similarly, y_a is the number of edges between A and C from the vertex a. Obviously,

$$\sum_{a \in A} x_a = X \text{ and } \sum_{a \in A} y_a = Y.$$

Now the number or triangles containing the vertex a is bounded by $\min\{Z, x_a y_a\}$. Since a triangle containing the vertex a is determined by two edges containing a or determined by the opposite edge of a. But we have, $\min\{Z, x_a y_a\} \leq \sqrt{Z x_a y_a}$. Thus the number of triangles is bounded by

$\sum_a \sqrt{Z x_a y_a} = \sqrt{Z} \sum_a \sqrt{x_a y_a} \leq \sqrt{Z}.\sqrt{\sum_a x_a)(\sum_a y_a)}$ (by Cauchy-Schwartz inequality). So we are done. □

(**Cauchy-Schwartz Inequality :** Let $a_1, \cdots a_k$ and $b_1, \cdots, b_k$ be positive real numbers. Then we have, $(x_1^2 + \cdots x_k^2)(y_1^2 + \cdots y_k^2) \leq (x_1 y_1 + \cdots x_k y_k)^2)$.

If we take $X = Y = Z = n^2$ then the number of triangles is bounded by n^3. We have an example where the number of triangles is exactly n^3 namely when we take a complete tripartite graph. That is we have three disjoint set of vertices A, B and C each of size n. Consider all possible edges between A and B, between A and C and between B and C. Obviously the number of edges between A and B or B and C or A and C are exactly n^2 and the number of triangles is n^3.

Security Study of The Double Length Compression Function.

Let $A = \{0,1\}^n \times \{1\}, B = \{0,1\}^n \times \{2\}$ and $C = \{0,1\}^n \times \{3\}$. We can correspond each query by an edge of a tripartite graph G on $A \sqcup B \sqcup C$ as follow: given a query (x, y) on f_1 we add an edge $\{(x,1),(y,2)\}$. The number 1,2 and 3 are used to make A, B and C disjoint. Similarly we can add edges for queries on f_2 and f_3. Now it is easy to note that a computable message corresponds to a triangle in the graph G. Thus the number of computable message is equal to the number of triangles. Also

the adversary can ask at most q queries for each f_i and hence the number of edges between A and B or B and C or A and C are at most q. Thus by Theorem 6.4 we have at most $q^{3/2}$ computable inputs for F. Thus the winning probability is bounded by $q^{3/2}(q^{3/2}-1)/2^{2n+1}$. Thus, the number of queries needed to get a collision is $\Omega(2^{2n/3})$. We show an attack which makes $\mathrm{O}(2^{2n/3})$ queries to get a collision on F.

A Collision Attack on F. The attack procedure is very much similar to the security proof. We first choose $2^{n/3}$ values of x_i, y_i and z_i independently, $1 \leq i \leq 2^{n/3}$. Now we query $f_1(x_i, y_j)$ for all $1 \leq i, j \leq 2^{n/3}$. Thus we have to make $2^{2n/3}$ queries of f_1. Similarly, we query for f_2 and f_3. Now we have 2^n computable inputs and check whether there is any computable collision pair.

Remark 6.1 *It is easy to note that, in the security proof of F we do not use the fact that $|x| = |y| = |z| = n$. In fact, if we have $f_i : \{0,1\}^{3n} \rightarrow \{0,1\}^n$, $1 \leq i \leq 3$ and define $F(x,y,z) = (f_1(x,y||0^n) \oplus f_2(y,z))||f_2(y,z) \oplus f_3(x,z)$, where $|x| = |y| = n$ and $|z| = 2n$ then we have same security level and similar attack as in the previous definition. Since the message block size is $2n$, the rate of this compression function is $2n/(3 \times 2n) = 1/3$.*

Remark 6.2 *Note that, in these above collision attacks the time and space complexities both are $\Omega(2^n)$. Very recently, Knudsen and Muller [47] describes an attack which need $O(2^{2n/3})$ time and memory. Their algorithm also needs $O(2^{2n/3})$ queries.*

Remark 6.3 *One can define a function $F : \{0,1\}^{4n} \rightarrow \{0,1\}^{2n}$ by $F(x,y,z_1,z_2) = (f_1(x,y,z_1) \oplus f_2(y,z_1,z_2))||(f_2(y,z_1,z_2) \oplus f_3(x,z_1,z_2))$ hoping for more security. But an attack can be shown with complexity $O(2^{2n/3})$. First, fix some z_1 and then choose $2^{n/3}$ values of x, y and z_2 independently. By the same argument like previous attack, it still has 2^n computable messages and hence we expect to have a collision on F.*

6.5 An Efficient Double Length Hash Function

Let the compression function $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$. For $i > 1$, define $f^{(i)} : \{0,1\}^{(i+1)n} \rightarrow \{0,1\}^n$ by using the classical iteration. Thus, for $x_0||\cdots||x_i$ with $|x_j| = n$, $0 \leq j \leq i$ and $h_0 = x_0$.

$$f^{(i)}(x_0||\cdots||x_i) = h_i, \text{ where } h_j = f(h_{j-1}, x_j),\ 1 \leq j \leq i.$$

We say $f^{(i)}$, the *i-iterated compression function.* Now we can observe that the multicollision on this compression function is not as easy as the classical hash function, since we restrict the number of message blocks (see Chapter 5 for more detail discussion about multicollision). Any r^i-way collision on $f^{(i)}$ reduces to at least r-way collision on the underlying compression function f (by using pigeon-hole principle). Thus, if we assume that $(r+1)$-way collision on f is infeasible then we can have at most r^i-way collision on $f^{(i)}$. Recall that, in the random oracle model of f, r-way collision requires $\Omega(2^{n(r-1)/r})$ queries. Now we summarize this by the following lemma.

Lemma 6.1 *(r^i+1)-way collision on $f^{(i)}$ reduces to at least $(r+1)$-way collision on f. In particular, when f is a random function, the complexity of (r^i+1)-way collision attack on $f^{(i)}$ is $\Omega(2^{nr/(r+1)})$ and the complexity of (n^i+1)-way collision attack on $f^{(i)}$ is $\Omega(2^n)$.*

Like the concatenation of two independent hash functions, we can define the concatenation of two independent i-iterated compression functions. Thus, given two independent compression functions, f_1 and f_2, we can define a double length compression function, $F^{(i)}(X) = f_1^{(i)}(X)||f_2^{(i)}(X)$, $|X| = n(i+1)$. Obviously, in this construction, we need to assume $i \geq 2$. Otherwise, for $i = 1$, it does not compress the input. Following Fig. 6.3 depicts the compression function for $i = 2$.

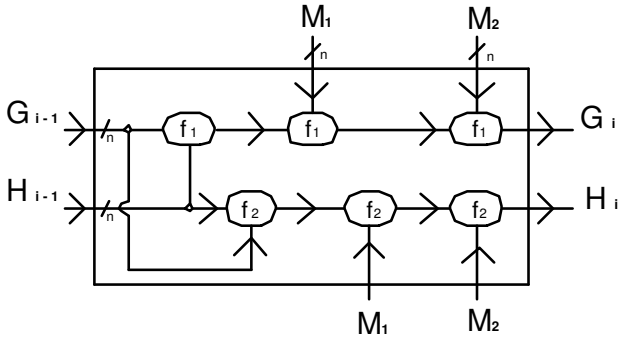

Figure 6.3: An efficient double length compression function, $i = 2$

Now we can study the security property of this concatenated compression function in the random oracle model.

Lemma 6.2 *If f is a random function then for any two distinct $(i+1)$-block inputs X and Y, $Pr[f^{(i)}(X) = f^{(i)}(Y)] \leq i/2^n$. If f_1 and f_2 are two independent random functions then $Pr[F^{(i)}(X) = F^{(i)}(Y)] = i^2/2^{2n}$.*

Proof. Let j be the round number where collision of f occurs. Call this event by C_j. Thus, $f^{(i)}(X) = f^{(i)}(Y)$ implies $\cup_{j=1}^{i} C_j$. Now, $\Pr[C_j] = \Pr[f(X_j) = f(Y_j)$ and $X_j \neq Y_j] = 1/2^n$, X_j and Y_j denote the input of f at j^{th} invocation for messages X and Y respectively. Thus, $\Pr[\cup_{j=1}^{i} C_j] \leq i/2^n$.

$$\begin{aligned}\Pr[F^{(i)}(X) = F^{(i)}(Y)] &= \Pr[f_1^{(i)}(X) = f_1^{(i)}(Y), f_2^{(i)}(X) = f_2^{(i)}(Y)] \\ &= \Pr[f_1^{(i)}(X) = f_1^{(i)}(Y)] \times \Pr[f_2^{(i)}(X) = f_2^{(i)}(Y)] \\ &= i^2/2^{2n}.\end{aligned}$$

The second equality follows from the fact that f_1 and f_2 are independent random functions and the last equality is immediate from the first half of the Lemma. □

Thus to find the collision probability for any adversary we need to compute the number of pairs (X, Y) the adversary can get from any possible set of queries. Note that the adversary should compute the F-values of both X and Y. Now we state the computable message which means the message whose hash value can be computed from the set of queries the adversary made. We fix $i \geq 2$.

Definition 6.5 (Computable message) *Let $\mathcal{Q}_j$ be the set of query response tuples for the random function f_j, $j = 1, 2$. X is said to be a computable message for $f_j^{(i)}$ (also for $F^{(i)}$) with respect to $\mathcal{Q}_j$ if the value of $f_j^{(i)}(X)$ (or $F^{(i)}$) can be computed from $\mathcal{Q}_j$ (or $\mathcal{Q}_1 \cup \mathcal{Q}_2$ respectively).*

More precisely, if $X = x_0||\cdots||x_i$ then X is computable for $f_1^{(i)}$ with respect to $\mathcal{Q}_1$ if $(x_0||x_1, h_1)$, $(h_1||x_2, h_2)$,$\cdots$,$(h_{i-1}||x_i, h_i) \in \mathcal{Q}_1$. Thus the $f_1^{(i)}$-value of X is h_i. Similarly one can define computable messages for $f_2^{(i)}$. A message X is computable with respect to $\mathcal{Q}_1 \cup \mathcal{Q}_2$ for the compression function $F^{(i)}$, if X is computable for both $f^{(i)}$ with respect to $\mathcal{Q}_j$, $j = 1, 2$.

Let q be the number of queries. We assume that $q = o(2^n)$. Thus there is no n-way collision on both f_1 and f_2. Note that, the complexity of n-way collision on a random function is $\Omega(2^{n(n-1)/n}) = \Omega(2^n)$. Thus we can have at most n^{i-1}-way collision on $f_1^{(i-1)}$

or $f_2^{(i-1)}$. The number of computable messages for $F^{(i)}$ is at most qn^{i-1}. Thus, total number of pairs of the form (X, Y), where X and Y are two different $(i+1)$-block computable messages, is at most $q^2n^{2(i-1)}/2$. Thus, probability that we have a collision among these pairs is bounded by $i^2q^2n^{2(i-1)}/2^{2n+1}$. To have non-negligible probability we need $q = \Omega(2^n/in^{i-1})$. Thus we have the following theorem :

Theorem 6.5 *If f_1 and f_2 are two independent random functions then the complexity for finding a collision on $F^{(i)}$ requires $\Omega(2^n/(in^{i-1}))$ queries.*

Efficiency of the compression function. The rate function of the compression function, $F^{(i)}$, is $((i+1)n - 2n)/2ni = \frac{1}{2} - \frac{1}{2i}$. Thus, the rate of the compression function is close to $1/2$ provided i is large. So we have a trade-off between the security level and the efficiency.

Now we define a double length hash function $H^s : (\{0,1\}^n)^* \rightarrow \{0,1\}^{2n}$, $s \geq 2$. We can define the hash function on arbitrary domain by applying some standard padding rule. Let $M = m_1||\cdots||m_l$ be l-block message, $|m_i| = n$ for each i. Let $l = (s-1)b+r$, where $0 \leq r < s-1$. Thus, we divide the message $M = M_1||\cdots||M_b||M_{b+1}$, where $|M_i| = (s-1)n$, $1 \leq i \leq b$ and $|M_{b+1}| = rn$. In case of $r = 0$ we do not have any message block M_{b+1}. Let H_0 be an initial two block message that is $|H_0| = 2n$. Now define the hash function $H^s(H_0, M)$ as follows;

Algorithm $H^s(H_0, m_1||\ldots||m_l)$

$H_i = F^s(H_{i-1}, M_i)$, $i = 1$ to b

If $r > 0$ **then** $H = F^{r+1}(H_b, M_{b+1})$

If $r = 0$ **then** $H = H_b$

Return H

Thus, the hash function is the classical iterated hash function by using two underlying compression functions $F^{(s)}$ and $F^{(r+1)}$. Thus any collision on $H^{(s)}$ reduces to the collision on one of the compression function. Thus we have the following theorem:

Theorem 6.6 *If the underlying compression functions f_1 and f_2 are two independent random functions, then for any $s \geq 2$, collision on $H^{(s)}$ requires $\Omega(2^n/s^2n^{s-1})$ complexity.*

(2nd) Preimage security of the new hash function.

Similar to the previous section we can study the (2nd) preimage security. Recall that we say a message X is computable from the set of queries $\mathcal{Q}$ if $f^{(i)}(X)$ can be computed from the set $\mathcal{Q}$. We have already observed that if q is the maximum number of queries and at most r-way collision is possible then we can have qr^{i-1} computable messages. Now given M, $F^{(i)}(M)$ is a $2n$-bit random string. We also have observed that $\Pr[F(M) = F(N)] = i^2/2^{2n}$, where $M \neq N$. So, if $q = \mathrm{o}(2^n)$ then the number of computable message for N is at most $n^{i-1}q$. Thus, the probability of the event that there will be a computable message $N \neq M$ such that $F^{(i)}(M) = F^{(i)}(N)$ is bounded by $qn^{i-1}/i^2 2^{2n}$. Hence complexity for any attack algorithm of 2nd preimage attack is $\Omega(2^{2n}/i^2 n^{i-1})$.

6.5.1 A Proposal of Most Efficient Hash Function

Let $f_1, f_2 : \{0,1\}^{2n} \to \{0,1\}^n$ be two independent random compression functions. We define a compression function $g^{(i)} : \{0,1\}^{(i+1)n} \to \{0,1\}^{2n}$ as follows;

$$g^{(i)}(x_0||x_1||\cdots||x_i) = f_1^{(i)}(x_0||x_1||\cdots||x_i)||f_2(h,x),$$

where $h = h_1 \oplus \cdots \oplus h_i \oplus x_0$ and $x = x_0 \oplus \cdots \oplus x_i$. The rate function of $g^{(i)}$ is almost equal to one provided i is large. More precisely, rate is $(i-1)n/(i+1)n = 1-2/(i+1)$. The following Fig. 6.4 is depicted the compression function for $i = 2$.

Now we give some intuitions why it could be a secure compression function. Let us consider $i = 2$. Suppose we have two different inputs $X = x_0||x_1||x_2 \neq Y = y_0||y_1||y_2$ such that $g^{(2)}(X) = g^{(2)}(Y)$. Now we consider several cases. First of all, if we have collisions on both f_1 and f_2 then the probability of the event $\{g^{(2)}(X) = g^{(2)}(Y)\}$ is $1/2^{2n}$ since both f_1 and f_2 are independent random function. So assume that $x = x_0 \oplus x_1 \oplus x_2 = y = y_0 \oplus y_1 \oplus y_2$ and $h = h_1 \oplus h_2 \oplus x_0 = h'_1 \oplus h'_2 \oplus y_0 = h'$.

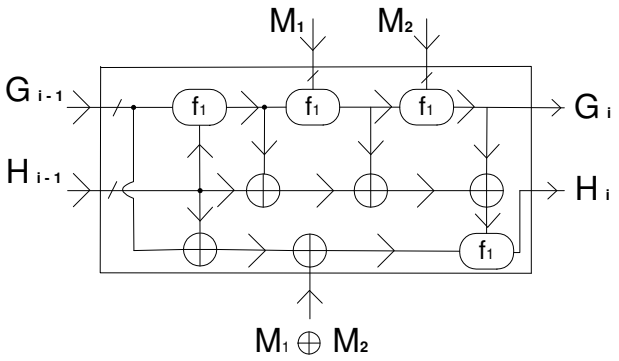

Figure 6.4: The most efficient double length compression function, $i = 2$

1. If $x_0 = y_0$ then $x_1 \neq y_1$ and $x_2 \neq y_2$. Thus we need $f_1(h_1||x_2) = f_1(h'_1||y_2)$ and $f_1(x_0||x_1) = f_1(y_0||y_1)$. In fact, after finding $x_1 \neq y_1$ such that $f_1(h_1||x_1) = f_1(h'_1||y_1)$, we need to find a pair (x_2, y_2) such that $x_2 \oplus y_2 = x_1 \oplus y_1$ and $f_1(h_1||x_2) = f_1(h'_1||y_2)$. To achieve this it might require 2^n many queries.

2. If $x_0 \neq y_0$ then we need $h_1 \oplus h'_1 = x_0 \oplus y_0$. Now, again we need a pair (x_2, y_2) such that $x_2 \oplus y_2 = x_0 \oplus y_0 \oplus x_1 \oplus y_1$ and $f_1(h_1||x_2) = f_1(h'_1||y_2)$. Intuitively we need 2^n many queries to find it.

Now only thing we need to verify is that the number of computable messages is not high compare to the number of queries. If the number of computable message would be qr^{i-1}, then the number of queries is likely to be at least $q.2^{n(r-1)/r}$. So we conjecture that the collision complexity of the above compression function is $\Omega(2^n/i)$. It would be interesting to prove this conjecture in the random oracle model.

6.6 Conclusion

We have studied several new double length compression functions. We first introduced a class of double length compression functions which contains recently known constructions [37, 57]. We studied their security levels in the random oracle model. A rate 1/3 double length compression function has been designed. The best collision attack of this compression function requires $\Omega(2^{2n/3})$ queries (also time and space complexity). We also designed a double length compression function F_i of rate close to 1/2 (the rate

of concatenated hash function). The design is very much similar to the concatenated hash functions. It had almost maximal security level. In fact, we believe that the complexity of the best collision attack on the above double length compression function F_i is $\Omega(2^n/i)$. It would be interesting to prove our belief. Finally, we proposed a rate almost equal to one double length hash function. We pose finding the security level of this hash function as an open problem. A possible future research would be to design efficient and as well as secure double length hash functions.

Chapter 7

Conclusion and Future Work

This thesis dealt with several design of hash function and multicollision attack on hash function. We proposed two parallel mask based domain extension algorithms. One of them was based on a complete binary tree which is very easy to implement. It was a reasonable improvement over the best known complete binary tree based extension [89]. It uses $m_t = O(t+\log_2^* t)$ many masks for a complete binary tree of height t. Previously, it was $(t+\log_2 t)$ [89]. We also proved that this is optimum among all complete binary tree based domain extension where the masking assignment is level-uniform. We can pose the following conjecture;

Conjecture : The minimum number of masks to be required to have a strongly even-free masking assignment on a complete binary tree is m_t.

We provided a sufficient condition which says that if the underlying masking assignment is strongly even-free then the extension is valid. In fact, we see that all known valid constructions were based on strongly even-free masking assignment. Thus, it would be interesting to prove the condition as a necessary condition.

Conjecture : A necessary condition for UOWHF-preserving domain extension algorithm is that the underlying masking assignment is strongly even-free.

Finally we proposed a tree based construction which is optimum (with respect to both the number of masks and time complexity or parallelism) among all tree based constructions.

In this thesis we generalized the definition of PGV-hash functions into a PGV-hash families. In the new definitions we had more secure hash family (forty-two hash families) with respect to collision resistant and one-wayness. Unlike the original definition

of PGV hash functions, the new definition is a keyed family. Thus we can study other security notions like target collision resistant. In fact all these forty-two hash families become target collision resistant. As AES is considered to be a good candidate for a block cipher, we can implement these hash families using AES. Because of our results, the only attacks for these hash families should be based on some internal weakness of AES. In other words, these hash families can be practically constructed using AES. The proof techniques used here are natural and apply directly to the security notions. So one can also study these proof techniques to get better ideas about using the black box model.

Recently there have been many proposed approaches to design hash functions with large output hash values. The most simple and natural one is the concatenation of two classical iterated hash functions. Joux [41] showed some collision (and preimage) attacks on this hash function. It is an interesting question to find design techniques for hash functions for which multicollision attacks are infeasible. In this thesis, we have defined a large natural class of hash functions and we studied their security with respect to multicollision attacks. Unfortunately, we have found efficient multicollision attacks on the hash functions when the message blocks are processed at most twice. So the natural question that arises is if multicollision security can be obtained if the message blocks are used more than twice. One can also search for some other designs outside this class of hash functions, and study their multicollision security.

We also studied several new double length compression functions. We first introduced a class of double length compression function which contains recently known constructions [37, 57]. We studied their security levels in the random oracle model. We designed a double length compression functions of rate 1/3 and we study its security level. We also designed a double length compression function $F^{(i)}$ of rate close to 1/2 (the rate of concatenated hash function). The design is very much similar to the concatenated hash functions. It had almost maximal security level. In fact, we believe that the complexity of the best collision attack on the above double length compression function $F^{(i)}$ is $\Omega(2^n/i)$. Finally, we proposed a rate almost equal to one double length hash function. We leave as an open problem to find the security level of this hash function. Our future research would be to design efficient and as well as secure double length hash functions.

Bibliography

[1] R. Anderson, E. Biham, *Tiger: A new fast hash function*. Fast Software Encryption'96, Lecture Notes in Computer Science, vol **1039**, Springer-Verlag, pp. 89-97, 1996.

[2] T. Baritaud, H. Gilbert, M. Girault. *FFT Hashing is not Collision free*. Advances in Cryptology - Eurocrypt'92, Lecture Notes in Computer Science, vol **658**, Springer-Verlag, pp. 35-44, 1992.

[3] P. S. L. M. Barreto and V. Rijmen. *Whirlpool hashing function*. Primitive submitted to NESSIE, September 2000. Available at http://www.cryptonessie.org/.

[4] D. Bayer, S. Haber and W. S. Stornetta. *Improving the efficieny and reliablity of digital time-stamping*. In sequences H, Methods in Communication, Security, and Computer Science, Springer-Verlag, pp. 329-334, 1993.

[5] M. Bellare. *A Note on Negligible Function*. Journal of Cryptology, Springer-Verlag, vol **15**, No. 4, pp. 271-284, September, 2002.

[6] M. Bellare and T. Kohno. *Hash function balance and its impact on birthday attacks*. Advances in Cryptology - Eurocrypt'04, Lecture Notes in Computer Science, vol **3027**, Springer-Verlag, pp. 401-418, 2004.

[7] M. Bellare and P. Rogaway. *Collision-Resistant Hashing: Towards Making UOWHFs Practical*. Advances in Cryptology - Crypto'97, Lecture Notes in Computer Science, vol **1294**, Springer-Verlag, pp. 470-484, 1997.

[8] E. Biham, R. Chen, A. Joux, P. Carribault, W. Jalby and C. Lemuet. *Collisions of SHA-0 and Reduced SHA-1*. To appear in Eurocrypt-05, 2005.

[9] J. Black, M. Cochran and T. Shrimpton. *On the Impossibility of Highly Efficient Blockcipher-Based Hash Functions*. To appear in Eurocrypt-05, 2005. ePrint Archive, 2004. Available at http://eprint.iacr.org/2004/062.

[10] J. Black, P. Rogaway, and T. Shrimpton. *Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV*. Advances in Cryptology - Crypto'02, Lecture Notes in Computer Science, vol **2442**, Springer-Verlag, pp. 320-335, 2002.

[11] B. O. Brachtl, D. Coppersmith, M. M. Hyden, S. M. Matyas, C. H. Meyer, J. Oseas, S. Pilpel, M. Schilling. *Data Authentication Using Modification Detection Codes Based on a Public One Way Encryption Function*. U.S. Patent Number **4,908,861**, March 13, 1990.

[12] E. Brickell, D.Pointcheval, S. Vaudenay and M. Yung. *Design validation for discrete logarithm based signature scheme*. PKC'00, Lecture Notes in Computer Science, vol **1751**, Springer-Verlag, pp. 276-292, 2000.

[13] L. Brown, J. Pieprzyk and J. Seberry. *LOKI - a cryptographic primitive for authentication and secrecy applications*. Advances in Cryptology- AusCrypt'90, Lecture Notes in Computer Science, vol **453**, Springer-Verlag, pp 229-236, 1990.

[14] R. Canetti, O. Goldreich and S. Halvei. *The random oracle methodology, revisited*. 30^{th} Annual ACM Symposium on Theory of Computing (STOC), pp. 209-218, 1998.

[15] D. Chaum, E. Van. Heijst and B.Pfitzmann. *Undeniable signatures*. Advances in Cryptology - Crypto'91, Lecture Notes in Computer Science, vol **576**, Springer-Verlag, pp. 470-484, 1992.

[16] D. Coppersmith, *Another birthday attack*. Advances in Cryptology - Crypto'85, Lecture Notes in Computer Science, vol **218**, Springer-Verlag, pp. 14-17, 1986.

[17] R. Cramer and V. Shoup. *Using Hash Functions as a Hedge against Chosen Ciphertext Attack*. Advances in Cryptology - Eurocrypt'00, Lecture Notes in Computer Science, vol **2442**, Springer-Verlag, pp. 275-288, 2000.

[18] J. Daemen, C. S. K. Clapp. *Fast Hashing and Stream Encryption with PANAMA*. Fast Software Encryption'98, Lecture Notes in Computer Science, vol **1372**, Springer-Verlag, pp. 60-74, 1998.

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

[20] I. B. Damgård. *Collision Free Hash Functions and Public Key Signature Schemes*. Advances in Cryptology - Eurocrypt'87, Lecture Notes in Computer Sciences, Vol. **304**, Springer-Verlag, pp. 203-216, 1987.

[21] I. B. Damgård. *A Design Principle for Hash Functions*. Advances in Cryptology - Crypto'89, Lecture Notes in Computer Sciences, vol **435**, Springer-Verlag, pp. 416-427, 1989.

[22] D. W. Davies and W. L. Price. *The Application of Digital Signatures based on Public Key Cryptosystem*. NPL Report DNACS 39/80, National Physical Laboratory, Teddington, Middlesex, England, Dec 1980.

[23] C. Debaert and H. Gilbert. RIPEMD^L *and* RIPEMD^R *improved variants of MD4 are not collision free*. Fast Software Encryption'02, vol **2355**, Lecture Notes in Computer Science, pp. 52-65, Springer- Verlag, 2002.

[24] R. Diestel. *Graph Theory*. Electronic Edition, 2000, Springer-Verlag. Available at http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/

[25] H. Dobbertin. *Cryptanalysis of MD4*. Fast Software Encryption, Cambridge Workshop. Lecture Notes in Computer Science, vol **1039**, Springer-Verlag, 1996.

[26] H. Dobbertin. *Cryptanalysis of MD5*, Rump Session of Eurocrypt'96, 1996. Available at http//www.iacr.org/conferences/ec96/rump/index.html.

[27] C. Dwork and M. Naor. *An Efficient Existentially Unforgeable Signature Scheme and Its Applications*. Advances in Cryptology, Crypto-94, Lecture Notes in Computer Science, vol **839**, Springer-Verlag, pp. 234-246, 1994.

[28] H. Finney. *More problems with hash functions.* The cryptographic mailing list, 24 Aug 2004. Available at http://lists.virus.org/cryptography-0408/msg00124.html.

[29] FIPS 46-3, *Data Encryption Standard*. National Institute of Standards and Technology, Oct. 1999.

[30] R. Gennaro, L. Trevisan. *Lower Bounds on the Efficiency of Generic Cryptographic Constructions*, FOCS'00, pp. 305-313, 2000.

[31] J. K. Gibson. *Discrete logarithm hash function that is collision free and one-way.* IEEE Proceedings on Information Theory **138**, pp. 407-410, 1991.

[32] M. Girault, R. Cohen and M. Campana. *A Generalized Birthday Attack.* Advances in Cryptology-Eurocrypt'88, Lecture Notes in Computer Science, vol **330**, Springer-Verlag, pp. 129-157, 1988.

[33] M. Girault and J. Stern. *On the length of cryptographic hash-values used in identification schemes.* Advances in Cryptology-Crypto'94, Lecture Notes in Computer Science, vol **839**, Springer-Verlag, pp. 428-446, 1994.

[34] S. Goldwasser, S. Micali and R. Rivest. *A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks*, SIAM Journal of Computing, vol **17**, No. 2, pp. 281-308, April 1998.

[35] S. Haber and W. S. Stornetta. *How to timestampting a digital document.* Journal of Cryptology **3(2)**, pp. 99-111, 1991.

[36] M. Hattori, S. Hirose and S. Yoshida. *Analysis of Double Block Lengh Hash Functions.* 9th IMA International Conference Cryptographi and Coding, 2003, Lecture Notes in Computer Science, vol **2898**, pp. 290-302, 2003.

[37] S. Hirose. *Provably Secure Double-Block-Length Hash Functions in a Black-Box Model*, 7th International Conference on Information Security and Cryptology, 2004.

[38] W. Hohl, X. Lai, T. Meier and C. Waldvogel. *Security of iterated hash functions based on block ciphers.* Advances in Cryptology- Crypto'93, Lecture Notes in Computer Science , vol **773**, Springer-Verlag, pp. 379-390, 1993.

[39] D. Hong, B. Preneel and Sangjin Lee. *Higher Order Universal One-Way Hash Functions.* Advances in Cryptology - Asiacrypt'04, Lecture Notes in Computer Science, vol **3329**, Springer-Verlag, pp. 201-213, 2004.

[40] R. Impagliazzo and M. Naor. *Efficient cryptographic schemes provably as secure as subset sum.* Journal of Cryptology, Vol **9**, No4, Autumn 1996.

[41] A. Joux. *Multicollision on Iterated Hash Function. Application to cascaded constructions.*. Advances in Cryptology - Crypto'04, Lecture Notes in Computer Science, vol-**3152**, Springer-Verlag, pp. 306-316, 2004.

[42] B. S. Kaliski, *MD2 Message-Digest algorithm*. Request for Comments (RFC) **1319**, Internet Activities Board, Internet Privacy Task Force, April 1992.

[43] B. S. Kaliski and M. Robshaw. *Message Authentication with MD5*. RSA Labs, CryptoBytes, vol **9**, No. 1, Spring 1995.

[44] J. Kelsey and B. Schneier. *Second Preimages on n-bit Hash Functions for Much Less than* 2^n *Work*. Cryptology ePrint Archive, 2004. Available at http://eprint.iacr.org/2004/304.

[45] J. Kilian, and P. Rogaway. *How to protect DES against exhaustive key search*. Journal of Cryptology, **14(1)**: pp. 17-35, 2001, Earlier version in Crypto'96, 1996.

[46] J. H. Kim, D. R. Simon and P. Tetali. *Limits on the Efficiency of One-Way Permutation-Based Hash Functions*. Proceedings of the **40**th Annual Symposium on Foundations of Computer Science, p. 535, October 17-18, 1999.

[47] L. R. Knudsen and F. Muller*Some Attacks Against a Double Length Hash Proposal*. To appear in Asiacrypt-05.

[48] L. Knudsen, X. Lai and B. Preneel. *Attacks on fast double block length hash functions*. Journal of Cryptology, vol **11**, no-1, winter, 1998.

[49] L. Knudsen and B. Preneel. *Construction of Secure and Fast Hash Functions Using Nonbinary Error-Correcting Codes*. IEEE transactions on information theory, vol **48**, No. 9, Sept-2002.

[50] H. Krawczyk. *LFSR-based hashing and authentication*. Advances in Cryptology, Proceedings Crypto'94, Lecture Notes in Computer Science, vol **839**, Springer-Verlag, pp. 129-139, 1994.

[51] H. Krawczyk, M. Bellare and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. Internet RFC **2104**, February 1997.

[52] H. Lai and J. L. Massey. *Hash functions based on block ciphers*. Advances in Cryptology, Eurocryp'92. Lecture Notes in Computer Science, vol **658**, Springer-Verlag, pp. 56-70, 1992.

[53] W. Lee, D. Chang, S. Lee, S. Sung and M. Nandi. *New Parallel Domain Extenders for UOWHF*. Advances in Cryptology, Asiacrypt-03. Lecture Notes in Computer Science, vol **2894**, Springer-Verlag, pp. 208-227, 2003.

[54] W. Lee, D. Chang, S. Lee, S. Sung and M. Nandi. *Construction of UOWHF : Two New Parallel Methods*. IEICE transaction on Fundamentals, vol **E88-A**, no.1, pp. 49-58, January 2005.

[55] W. Lee, M. Nandi, P. Sarkar, D. Chang, S. Lee and K. Sakurai *A Generalization of PGV-Hash Functions and Security Analysis in Black-Box Model*. Information Security and Privacy: 9th Australasian Conference, ACISP'04, Lecture Notes in Computer Science, vol **3108**, pp. 212-223, 2004.

[56] W. Lee, M. Nandi, P. Sarkar, D. Chang, S. Lee and K. Sakurai *PGV-style Block-Cipher-Based Hash Families and Black-Box Analysis*. IEICE transaction on Fundamentals, vol **E88-A**, no.1, pp. 39-48, January 2005.

[57] S. Lucks. *Design principles for Iterated Hash Functions*. ePrint Archive Report, 2004. Available at http://eprint.iacr.org/2004/253.

[58] T. Matsuo and K. Kurosawa. *On Parallel Hash Functions Based on Block-Cipher*. Information Security and Privacy: 8th Australasian Conference, ACISP'03, Lecture Notes in Computer Science, vol **2727**, pp. 510-521, 2003.

[59] S. M. Matyas, C. H. Meyer, J. Oseas. *Generating strong one-way functions with cryptographic algorithm*. IBM Techn. Disclosure Bull., vol **27**, No. 10A, 1985, pp. 5658-5659.

[60] A. J. Menezes, P. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press ISBN: 0-8493-8523-7, October 1996.

[61] R. C. Merkle. *One Way Hash Functions and DES*. Advances in Cryptology - Crypto'89, Lecture Notes in Computer Sciences, Vol. **435**, Springer-Verlag, pp. 428-446, 1989.

[62] R. C. Merkle. *A certified digital signature*. Advances in Cryptology - Crypto'89, Lecture Notes in Computer Sciences, Vol. **435**, Springer-Verlag, pp. 218-238, 1989.

[63] C. H. Meyer and M. Schilling. *Secure program load with manipulation detection code*. Proceedings Securicom, pp. 111-130, 1988.

[64] I. Mironov. *Hash functions: from Merkle-Damgard to Shoup*. Advances in Cryptology - Eurocrypt'01, Lecture Notes in Computer Science, vol **2045**, Springer-Verlag, pp. 166-181, 2001.

[65] S. Miyaguchi, M. Iwata, and K. Ohta. *New 128-bit hash function*. Proceedings 4th International Joint Workshop on Computer Communications, pp. 279-288, 1989.

[66] F. Muller. *The MD2 Hash Function Is Not One-Way*. Asiacrypt'04, Lecture Notes in Computer Science, vol **3329**, Springer-Verlag, pp. 214-229, 2004.

[67] M. Nandi. *A New Tree based Domain Extension of UOWHF*. Cryptology ePrint Archive, 2003. Available at http://eprint.iacr.org/2003/142.

[68] M. Nandi. *Study of Domain Extension of UOWHF and its Optimality*, Cryptology ePrint Archive, 2003. Available at http://eprint.iacr.org/2003/158.

[69] M. Nandi. *A Sufficient Condition for Optimal Domain Extension of UOWHFs*, Seleceted Areas in Cryptology, 2004, Lecture Notes in Computer Science, vol **3357**, Springer-Verlag, pp 341-354, 2004.

[70] M. Nandi. *Towards Optimal Double Length Hash Functions*. To appear in Indocrypt'05.

[71] M. Nandi. *Designs of Efficient Secure Large Hash Values*. Cryptology ePrint Archive, 2004. Available at http://eprint.iacr.org/2004/296.

[72] M. Nandi, W. Lee, K. Sakurai and S. Lee. *Security Analysis of a* 2/3*-rate Double Length Compression Function in The Black-Box Model*. Fast Software Encryption'05, Lecture Notes in Computer Science, vol **3557**, Springer-Verlag, pp 243-254, 2005.

[73] M. Nandi and D. R. Stinson. *Multicollision Attacks on Generalized Hash Functions*. Cryptology ePrint Archive, 2004. Available at http://eprint.iacr.org/2004/330.

[74] M. Naor and M. Yung. *Universal one-way hash functions and their cryptographic applications*, Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing, ACM Press, pp 33-43, 1989.

[75] National Institute of Standards, FIPS 180-1, Secure Hash Standard. April 1995.

[76] NIST/NSA. *FIPS 180-2 Secure Hash Standard*. August, 2002. Online available at http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf.

[77] P. C. Oorschot and M. J. Wiener. *Parallel Collision Search with Cryptanalytic Applications*. Advances in Cryptology - Crypto'96, Lecture Notes in Computer Sciences, vol **1109**, Springer-Verlag, pp. 229-236, 1996.

[78] B. Preneel. *Analysis and Design of cryptographic hash*. PhD Thesis, Katholieke Universiteit Leuven, 1995.

[79] B. Preneel, A. Bosselaers and H. Dobbertin. *Cryptographic hash function RIPEMD-160*. CryptoBytes, vol. **3**, no. 2, pp. 9-14, 1997.

[80] B. Preneel, R. Govaerts and J. Vandewalle. *Cryptographically secure hash functions: an overview*. ESAT Internal Report, K. U. Leuven, 1989.

[81] V. Rijmen, B. Van Rompay, B. Preneel and J. Vandewalle. *Producing collisions for PANAMA*. Fast Software Encryption'02, Lecture Notes in Computer Science, no. **2355**, pp. 37-51, Springer-Verlag, 2002.

[82] R. L. Rivest. *MD4 message digest algorithm*. Advances in Cryptology, Proceedings Crypto'90, Lecture Notes in Computer Science, vol **537**, Springer-Verlag, pp. 303-311, 1991.

[83] R. L. Rivest *The MD5 message digest algorithm*. Available online : http://www.ietf.org/rfc/rfc1321.txt.

[84] R. L. Rivest and A. Shamir. *PayWord and MicroMint - two simple micropayment schemes*. CryptoBytes, **2(1)**:7-11, Springer-Verlag, 1996.

[85] P. Rogaway, T. Shrimpton. *Cryptographic Hash Function Basics: Definitions, Implications and separations for Pre-image resistance, Second Pre-image Resistance and Collision Resistance*, Fast Software Encryption'04, Lecture Notes in Computer Scince, Springer-Verlag, vol **3017**, pp. 371-388, 2004.

[86] B. V. Rompay, A. Biryukov, B. Preneel, J. Vandewalle. *Cryptanalysis of 3-Pass HAVAL*. Advances in Cryptology - Asiacrypt'03, Lecture Notes in Computer Scince, Springer-Verlag vol- **2894**, pp. 228-245, 2003.

[87] J. Rompel. *One-way functions are necessary and sufficient for signatures*, Proceedings of STOC, pp. 387-394, 1990.

[88] P. Sarkar. *Domain Extender for Collision Resistant Hash Functions: Improving Upon Merkle-Damgard Iteration*. ePrint Archive Report, 2002. Available at http://eprint.iacr.org/2003/173.

[89] P. Sarkar. *Construction of UOWHF : Tree Hashing Revisited*. ePrint Archive Report, 2002. Available at http://eprint.iacr.org/2002/058.

[90] P. Sarkar. *Masking Based Domain Extenders for UOWHFs: Bounds and Constructions*. Advances in Cryptology- Asiacrypt'04, Lecture Notes in Computer Science, vol **3329**, Springer-Verlag, pp. 187-200, 2004.

[91] P. Sarkar. *Domain Extenders for UOWHF: A Finite Binary Tree Algorithm*. ePrint Archive Report, 2003. Available at http://eprint.iacr.org/2003/009.

[92] P. Sarkar and Paul J. Schellenberg. *A Parallelizable Design Principle for Cryptographic Hash Functions*. ePrint Archive Report http://eprint.iacr.org/2002/031.

[93] T. Satoh, M. Haga and K. Kurosawa. *Towards Secure and Fast Hash Functions*. IEICE Trans. vol **E82-A**, No. 1 January, 1999.

[94] B. Schneier. *Cryptanalysis of MD5 and SHA*. Crypto-Gram Newsletter, Sept-2004. http://www.schneier.com/crypto-gram-0409.htm#3.

[95] C. P. Schnorr. *FFT Hashing : An Efficient Cryptographic Hash Function*. Presented at the rump session of the Crypto-91, 1991.

[96] C. P. Schnorr. *FFT Hash II : Efficient Cryptographic Hashing*. Advances in Cryptology - Eurocrypt'92, Lecture Notes in Computer Science, vol **658**, pp. 45-54, 1992.

[97] C. P. Schnorr and S. Vaudenay. *Parallel FFT-Hashing*, Fast Software Encryption'93, Lecture Notes in Computer Science, vol **809**, pp 149-156, 1993.

[98] C. P. Schnorr and S. Vaudenay. *The Black-Box Model for Cryptographic Primitives*, Journal of Cryptology, vol **11**, pp. 125-140, 1998.

[99] C. Shannon. *Communication theory of secrecy systems*, Bell Systems Technical Journal, **28(4)**: pp. 656-715, 1949.

[100] V. Shoup. *A composition theorem for universal one-way hash functions.* Advances in Cryptology - Eurocrypt'00, Lecture Notes in Computer Science, vol. **1807**, Springer-Verlag, pp 445-452, 2000.

[101] V. Shoup. *Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack.* SIAM Journal of Computing **33**: 167-226, 2003.

[102] D. Simon. *Finding collisions on a one-way street: can secure hash functions be based on general assumptions?*, Advances in Cryptology - Eurocrypt'98, Lecture Notes in Computer Science, vol **1403**, Springer-Verlag, pp. 334-345, 1998.

[103] D. R. Stinson. *Cryptography : Theory and Practice*, Second Edition, CRC Press, Inc.

[104] D. R. Stinson. *Some observations on the theory of cryptographic hash functions.* ePrint Archive Report, 2001. Available at http://eprint.iacr.org/2001/020/.

[105] S. Vaudenay. *FFT-Hash II is Not Yet Collision-Free*, Advances in Cryptology, Crypto-92, Lecture Notes in Computer Science, vol **740**, Springer-Verlag, pp. 587, 1992.

[106] S. Vaudenay. *On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER*, Fast Software Encryption'95, Lecture Notes in Computer Science, vol **1008**, 1995.

[107] D. Wagner. *Cryptanalysis of the Yi-Lam hash.* Advances in Cryptology- Asiacrypt'00 Lecture Notes in Computer Science, vol **1976**, Springer-Verlag, pp 483-488, 2000.

[108] D. Wagner: *A Generalized Birthday Problem.* Advances in Cryptology - Crypto-02, Lecture Notes in Computer Science, vol **2442**, pp. 288-303, 2002.

[109] M. Wegman and L. Carter. *New Hash Functions and their use in Authentication and set equality.* Journal of Computer and System Science, **22**: 265-279, 1981.

[110] X. Wang, D. Feng, X. Lai, H. Yu. *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD.* ePrint Archive Report, http://eprint.iacr.org/2004/199.

[111] R. Winternitz. *A secure one-way hash function built from DES*, In Proceedings of the IEEE Symposium on Information Security and Privacy, pp. 88-90, IEEE Press, 1984.

[112] X. Yi and K. Y. Lam. *A new hash function based on block cipher.* ACISP'97, Information Security and Privacy: 2nd Australasian Conference, Lecture Notes in Computer Science, vol **1270**, 1997.

[113] X. Yi and K. Y. Lam. *A hash function based on block cipher.* Electronics Letters, 6 Nov 1997, vol **33**, (no.23): 1938-1940, IEEE, 1997.

[114] Y. Zheng, T. Matsumoto and H. Imai. *Structural properties of one-way hash functions.* Advances in Cryptology - Proceedings of Crypto90, Lecture Notes in Computer Science, vol **537**, pp. 303311. Springer-Verlag, 1991. Extended version is available at www.calyptix.com/files/haval-paper.pdf.