OULU 2014
A 638

ACTA UNIVERSITATIS OULUENSIS

A
SCIENTIAE RERUM NATURALIUM

Tuomas Kortelainen

ON ITERATION-BASED SECURITY FLAWS IN MODERN HASH FUNCTIONS

*UNIVERSITY OF OULU GRADUATE SCHOOL;
UNIVERSITY OF OULU,
FACULTY OF SCIENCE, DEPARTMENT OF MATHEMATICAL SCIENCES;
FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING,
DEPARTMENT OF INFORMATION PROCESSING SCIENCE*

ACTA UNIVERSITATIS OULUENSIS
A Scientiae Rerum Naturalium 638

TUOMAS KORTELAINEN

ON ITERATION-BASED SECURITY FLAWS IN MODERN HASH FUNCTIONS

Academic dissertation to be presented with the assent of the Doctoral Training Committee of Technology and Natural Sciences of the University of Oulu for public defence in the OP auditorium (L10), Linnanmaa, on 9 December 2014, at 12 noon

UNIVERSITY OF OULU, OULU 2014

Supervised by
Professor Markku Niemenmaa
Doctor Ari Vesanen

Reviewed by
Professor Valtteri Niemi
Professor Orr Dunkelman

ISBN 978-952-62-0642-4 (Paperback)
ISBN 978-952-62-0643-1 (PDF)

ISSN 0355-3191 (Printed)
ISSN 1796-220X (Online)

Cover Design
Raimo Ahonen

JUVENES PRINT
TAMPERE 2014

Kortelainen, Tuomas, On iteration-based security flaws in modern hash functions.
University of Oulu Graduate School; University of Oulu, Faculty of Science, Department of Mathematical Sciences; Faculty of Information Technology and Electrical Engineering, Department of Information Processing Science
Acta Univ. Oul. A 638, 2014
University of Oulu, P.O. Box 8000, FI-90014 University of Oulu, Finland

Abstract

The design principles proposed independently by both Ralph Merkle and Ivan Damgård in 1989 are applied widely in hash functions that are used in practice. The construction reads the message in one message block at a time and applies iteratively a compression function that, given a single message block and a hash value, outputs a new hash value.

This iterative structure has some security weaknesses. It is vulnerable, for instance, to Joux's multicollision attack, herding attack that uses diamond structures and Trojan message attack.

Our principal research topic comprises the deficiencies in hash function security induced by the Merkle-Damgård construction. In this work, we present a variant of Joux's multicollision attack. We also develop a new, time-saving algorithm for creating diamond structures. Moreover, two new efficient versions of Trojan message attack are introduced.

The main contribution of the thesis is the analysis of generalized iterated hash functions. We study the combinatorial properties of words from a new perspective and develop results that are applied to give a new upper bound for the complexity of multicollision attacks against the so called q-bounded generalized iterated hash functions.

Keywords: cryptography, diamond structure, hash functions, multicollision, Trojan Message Attack, word combinatorics

Kortelainen, Tuomas, Iteraatioon perustuvia tietoturvaheikkouksia moderneissa hash-funktioissa.
Oulun yliopiston tutkijakoulu; Oulun yliopisto, Luonnontieteellinen tiedekunta, Matemaattisten tieteiden laitos; Tieto- ja sähkötekniikan tiedekunta, Tietojenkäsittelytieteiden laitos
Acta Univ. Oul. A 638, 2014
Oulun yliopisto, PL 8000, 90014 Oulun yliopisto

Tiivistelmä

Vuonna 1989 Ralph Merkle ja Ivan Damgård ehdottivat toisistaan riippumatta hash-funktioille suunnitteluperiaatteita, joita käytetään tänä päivänä laajasti. Niin kutsuttu Merkle-Damgård -rakenne lukee viestin sisään viestiblokki kerrallaan ja käyttää tiivistefunktiota, joka liittää hash-arvoon ja viestiblokkiin uuden hash-arvon.

Tällä iteratiivisella rakenteella on joitakin turvallisuusheikkouksia. Se on haavoittuva esimerkiksi Joux'n monitörmäyshyökkäykselle, timanttirakenteita hyödyntävälle paimennushyökkäykselle ja Troijan viesti -hyökkäykselle.

Väitöskirjan pääasiallinen tutkimusaihe on Merkle-Damgård -rakenteen aiheuttamat puutteet tietoturvassa. Tässä työssä esitetään uusi versio Joux'n monitörmäyshyökkäyksestä, luodaan uusi aikaa säästävä algoritmi timanttirakenteiden kehittämiseksi ja kaksi uutta tehokasta versiota Troijan viesti -hyökkäyksestä.

Väitöskirjan tärkein kontribuutio on yleistettyjen iteratiivisten hash-funktioiden turvallisuuden analysointi. Sanojen kombinatorisia ominaisuuksia tutkitaan uudesta näkökulmasta, jonka pohjalta kehitettyjä tuloksia soveltamalla luodaan uusi yläraja niin kutsuttujen q-rajoitettujen yleisten iteratiivisten hash-funktioiden monitörmäyshyökkäysten kompleksisuudelle.

Asiasanat: hash-funktiot, kryptografia, monitörmäykset, sanojen kombinatoriikka, timanttirakenteet, Troijan viesti -hyökkäyset

Acknowledgements

I wish to express my gratitude to my supervisors, Professor Markku Niemenmaa and PhD Ari Vesala for their support and rewarding collaboration. Also I wish to express my gratitude for the pre-examiners of my thesis, Professor Valtteri Niemi and Professor Orr Dunkelman for their critique and advice that have allowed me to significantly improve the quality of my thesis.

I have been lucky enough to be working with great people. I wish to thank Kimmo Halunen for our collaboration on hash functions. Especially I want to thank my father Juha for the constant, unyielding and selfless support that he has given me through the years. I also wish to thank Alli Huovinen, Keijo Väänänen and the staff of both the Department of Mathematical Sciences and the Division of Mathematics at the University of Oulu.

For financial support, I wish to thank the Tauno Tönnig foundation. I also wish to thank Professor Keijo Ruotsalainen who has created a pleasant atmosphere to teach and do research.

I want to thank my friends for relaxing and good times. Particularly, I wish to thank Soile Kontio for her friendship and the support she has given that has allowed me to concentrate on work more than would have been possible otherwise.

Finally, I wish to thank my family. Especially my precious children Venla and Arttu and my wife Tiina for all her love and encouragement.

Oulu, November 2014 Tuomas Kortelainen

Contents

1 Introduction

Before the late 20th century, cryptography was an art practiced mainly by military and intelligence organizations. The goal of cryptography was to hide your own secrets and at the same time find out the secrets of your possible enemies. There was very little theory or robust scientific study on the field. Instead, cryptography relied on creativity and personal skill.

The role cryptography plays in the society has changed dramatically. Today cryptography is everywhere. Protocols for exchanging secret keys, message authentication, digital signatures and other forms of cryptography are an integral and everyday part of our computer systems. Cryptography is used not only to protect secret military plans, but also every time we access a secured website. In their book *Introduction to Modern Cryptography* (2007) [16], Katz and Lindell define modern cryptography to be *the scientific study of techniques for securing digital information, transactions, and distributed computations*.

Cryptographic theory has experienced a similar drastic change. A broad and fast evolving theory has emerged to answer the new challenges and to allow the rigorous scientific study of cryptographic systems, i.e. algorithms that are designed to provide particular functionality while guaranteeing certain security properties. Cryptographic protocols are based on *Cryptographic primitives*, i.e. low-level cryptographic algorithms. One widely used class of cryptographic primitives is one-way hash functions or *cryptographic hashes*.

A hash function is an algorithm that maps data of arbitrary length to a data of fixed length. Such functions have many uses, for example in quick locating of data record using so called hash tables. However, when we use hash functions as building blocks for cryptographic systems, we are not interested in hash functions in general, but especially in cryptographic hashes. The input of a cryptographic hash function is usually called a message and the output a hash value. Cryptographic hash functions should have several security properties.

Ideally, a cryptographic hash function should work as a random oracle, meaning that for each message the resulting hash value should be chosen uniformly at random. Any regularities or undesired properties that appear can possibly be used by malicious adversaries in order to attack cryptographic systems that use the hash function as a cryptographic primitive.

In practice, hash functions are often used in situations where the whole message might not be ready at the beginning of the hashing process. The most successful solution to this problem was proposed independently by both Merkle and Damgård [6, 29] and is known as an *iterated hash function*. The basic idea of an iterated hash function is to divide the message in message blocks and apply a finite compression function on one message block at a time. Our work focuses on some security flaws this pattern creates and analyzes some proposed solutions to these flaws.

This work is organized in the following way. In chapter two, we define and explain some basic concepts and results of words, probabilities and hash functions. In chapter three, we give a brief survey of some previously proven results. The focus is on the Joux's multicollision attack, diamond structures and Trojan message attacks. The fourth chapter contains a variant of Joux's multicollision attack. The results of the chapter are based on [23], where the author was the main contributor. Chapter five describes new results concerning the complexity of creating a diamond structure and new versions of Trojan message attacks. Results are based on [22], where the author was the main contributor.

Chapters six and seven are the main contribution of this thesis and describe attacks against the so called generalized iterated hash functions. Chapter six explains the results that have been reached before and is based on several articles. The author contributed some results to articles [13, 19–21]. Finally, chapter seven contains the newest results on attacks against generalized iterated hash functions. The results of the chapter are based on article [24] where the author was the main contributor. Chapter eight is the conclusion of the thesis.

2 Basics

In this chapter, we will give a short introduction to concepts on words. The combinatorics on words (the theory of finite sequences) is needed in the formulation of definitions and in analysis of attack algorithms. We will also give prerequisites to probability theory and hash functions.

2.1 Words

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of all natural numbers and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. For each $l \in \mathbb{N}_+$, we define $\mathbb{N}_l$ to be the set of l first positive integers: $\mathbb{N}_l = \{1, 2, \ldots, l\}$. For each finite set S, let $|S|$ be the cardinality of S, i.e., the number of elements in S.

An *alphabet* is any finite nonempty set of abstract symbols called *letters*. Let A be an alphabet. A *word* (over A) is any finite sequence of symbols in A. Thus, assuming that w is a word over A, we can write $w = x_1 x_2 \cdots x_n$, where $n \in \mathbb{N}$ and $x_i \in A$ for $i = 1, 2, \ldots, n$. Above n is the *length* $||w||$ of w. Notice that n may be equal to zero; then w is the *empty word*, denoted by ε, that contains no letters. By $|w|_a$ we mean the number of occurrences of the letter a in w. Define the *alphabet of* w by $\text{alph}(w) = \{a \in A \mid |w|_a > 0\}$. A word α is q*-bounded* if $|\alpha|_a \leq q$ for all $a \in \text{alph}(\alpha)$.

For each $n \in \mathbb{N}$, denote by A^n the set of all words of length n over A. Let $A^* = \bigcup_{n=0}^{\infty} A^n$ be the set of all words and $A^+ = A^* \setminus \{\varepsilon\} = \bigcup_{n=1}^{\infty} A^n$ the set of all nonempty words over A.

The word u is a subword of w if there exist $m \in \mathbb{N}$ and $x_0, u_1, x_1 \ldots, u_m, x_m \in A^*$ such that $w = x_0 u_1 x_1 \ldots u_m x_m$ where $u = u_1 u_2 \cdots u_m$. A subword u of w is a *factor* of w if $w = x_0 u x_1$ for some $x_0, x_1 \in A^*$. A factor u of v is a *prefix* (*suffix*, resp.) if $w = u x_1$ for some $x_1 \in A^*$ ($w = x_0 u$ for some $x_0 \in A^*$, resp.).

The *concatenation* of two words u and v in A^* is the word uv obtained by writing u and v after one another. Clearly concatenation defines a binary operation $\cdot$ in A^*: $u \cdot v = uv$ for all $u, v \in A^*$. In algebraic terms $(A^*, \cdot)$ is a *free monoid* and $(A^+, \cdot)$ is a *free semigroup*. For any two sets U and V of words, let $UV = \{uv \,\big|\, u \in U, v \in V\}$.

2.2 Some Mathematical Tools

In this section, we will present some well-known basic results of calculus and probability theory. We will later use these results repeatedly.

Fact 1. For all real numbers x

$$\lim_{n\to\infty}(1+\frac{x}{n})^n = e^x.$$

Remark 1. If we choose for example $x=-1$ and $n=100$ we get $|(1-\frac{1}{n})^n - e^{-1}| < 0.0019$. In this work we will use this Lemma in the context where generally $n \geq 2^{160}$. This means that effectively we can assume that $(1-\frac{1}{n})^n$ equals to e^{-1}. In addition for any $n \in \mathbb{N}_+$, $(1-\frac{1}{n})^n < e^{-1}$ so e^{-1} offers an upper bound for $(1-\frac{1}{n})^n$ when $n \in \mathbb{N}_+$.

Fact 2. Most of our attack constructions are realized as (a sequence) of distinct statistical experiments, where each experiment is repeated until the required event happens. Consider a statistical experiment where the probability of an event A is p. Suppose that the experiment is repeated as a sequence of independent Bernouli trials until A happens for the first time. Let x be the number of repetitions needed. Then x is a random variable that follows the geometric distribution with parameter p and the expected value $E(x)$ of x satisfies $E(x) = p$.

Lemma 1. From a set of n distinct elements k elements are randomly drawn with replacement. The probability $p(n,k)$ that at least two of the drawn elements are equal is

$$1-\binom{n}{k}\frac{k!}{n^k}.$$

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

$$\frac{n(n-1)\cdots(n-k+1)}{n^k} = \frac{n!}{k!(n-k)!}\cdot\frac{k!}{n^k} = \binom{n}{k}\frac{k!}{n^k}.$$

The result follows. □

Remark 2. Choosing $n = 365$ above we get the famous birthday problem.

Remark 3. Now when n is large enough

$$\binom{n}{k}\frac{k!}{n^k} = \frac{n(n-1)\cdots(n-k+1)}{n^k} = 1\cdot(1-\frac{1}{n})\cdots(1-\frac{k-1}{n})$$

$$= (1-\frac{1}{n})^{n\cdot\frac{1}{n}}\cdots(1-\frac{k-1}{n})^{n\cdot\frac{1}{n}} \approx (e^{-1})^{\frac{1}{n}}\cdot(e^{-2})^{\frac{1}{n}}\cdots(e^{-k+1})^{\frac{1}{n}}$$

$$= e^{-\frac{k(k-1)}{2n}} = e^{-\binom{k}{2}/n}.$$

Thus $p(n,k) \approx 1 - e^{-\binom{k}{2}/n}$ when n is large.

Lemma 2. Let A and B be two randomly chosen subsets of set C such that $|A| + |B| < |C|$. Then the probability that $A \bigcap B \neq \emptyset$ is greater than

$$1 - (1 - \frac{1}{|C|})^{|A||B|}.$$

Proof. The probability that $A \cap B = \emptyset$ is

$$\frac{\binom{|C|}{|A|}\binom{|C|-|A|}{|B|}}{\binom{|C|}{|A|}\binom{|C|}{|B|}} = \frac{\binom{|C|-|A|}{|B|}}{\binom{|C|}{|B|}} = \frac{(|C|-|A|)!}{(|C|-|A|-|B|)!} \cdot \frac{(|C|-|B|)!}{(|C|)!}$$

$$= (1 - \frac{|A|}{|C|})(1 - \frac{|A|}{|C|-1}) \cdots (1 - \frac{|A|}{|C|-|B|+1}) < (1 - \frac{|A|}{|C|})^{|B|} < (1 - \frac{1}{|C|})^{|A||B|}.$$

The claim follows. □

Remark 4. If $|A||B| = |C|$, then according to Fact 1, the probability that $A \cap B \neq \emptyset$ is greater than $1 - e^{-1} = \frac{e-1}{e}$.

Consider now two sets of random variable values $A = \{x_1, x_2, \ldots x_m\}$ and $B = \{y_1, y_2, \ldots, y_n\}$. Each variable can assume any of t discrete values with equally likely probability. Lemma 2 is a special case of such a situation and proves that if $|A| = m$ and $|B| = n$ the probability that $A \bigcap B \neq \emptyset$ is larger than $1 - (1 - \frac{1}{t})^{mn}$.

The exact probability that $A \bigcap B \neq \emptyset$ can be found for example in Theorem 1 in [37]. Article [37] considers also the validity of widely used binomial approximation that estimates the probability that $A \bigcap B \neq \emptyset$ with $1 - (1 - \frac{1}{t})^{mn}$. The binomial approximation overstates the true probability that $A \bigcap B \neq \emptyset$. However, the error decreases rabidly as a function of t (see Section 3 in [37]). In our work, the created sets of random variable values are extremely small when compared with the number of possible values those variables can assume. Thus within the context of this work the binomial approximation holds well.

2.3 General Description of Hash Functions

Messages over an arbitrary finite alphabet can be encoded into messages over a binary alphabet. It follows that we may certainly, without loss of generality, assume that all our messages are over the binary alphabet $\{0, 1\}$.

Definition 1. A hash function (of length n, $n \in \mathbb{N}_+$) is a mapping $\mathtt{H}: \{0,1\}^* \to \{0,1\}^n$.

From now on we will assume that, unless otherwise stated, all hash functions will have the codomain of $\{0,1\}^n$, where $n \in \mathbb{N}_+$ is arbitrary but fixed. In other words, all the hash functions will have n-bit long *hash values*.

There are three main security properties that are usually required from a cryptographic hash function $\mathtt{H}$. These are preimage resistance, second preimage resistance and collision resistance [28].

- *Preimage resistance:* For essentially all pre-specified hash values, it is computationally infeasible to find any input which hashes to that output, i.e ., to find any message x' such that $\mathtt{H}(x') = h$ when given any h for which a corresponding message is not known.
- *Second preimage resistance:* It is computatinally infeasible to find any second message which has the same hash value as any specified message, i.e., given x, to find a second preimage $x' \neq x$ such that $\mathtt{H}(x) = \mathtt{H}(x')$.
- *Collision resistance:* It is computationally infeasible to find any two distinct messages x and x' which have the same hash value, i.e., $\mathtt{H}(x) = \mathtt{H}(x')$.

It is clear that all hash functions are vulnerable to so called *birthday attack.* In a birthday attack the attacker simply chooses random messages and calculates the respective hash values. The attacker needs to hash only approximately $2^{n/2}$ random messages to create two messages with the same hash value (see for example [28]). This is known as *birthday paradox.* Usually, it is assumed that the attacker needs to hash approximately 2^n random messages to create a preimage for any hash value or a second preimage for a given message. We will look at the details later.

However creating collisions, second preimages and preimages are by no means the only ways to attack a hash function in practice. This means that in addition to preimage resistance, second preimage resistance and collision resistance there are several other properties a hash function should have. Three of these are generalized collision resistance, herding resistance and Trojan message resistance.

- *Generalized collision resistance:* It is computationally infeasible to find any set K of k distinct messages $k \geq 2$ such that $\mathtt{H}(x) = \mathtt{H}(x')$ for all $x, x' \in K$, $x \neq x'$.
- *Herding resistance:* In the herding attack the adversary commits to a single hash value h_d and is then challenged with a prefix p. The attacker cannot control p, altough

she/he knows its length. It is computationally infeasible to provide a suffix x_s, such that $\mathtt{H}(px_s) = h_d$.

– *Trojan message resistance:* Given any finite message set P, of at least two messages, it is computationally infeasible to find a message t and a message set M such that $|M| = |P|$ and for each $p \in P$ there exists $x \in M$ such that $pt \neq x$ and $\mathtt{H}(pt) = \mathtt{H}(x)$.

Generalized collision resistance has been studied in [32]. A k-collision in $\mathtt{H}$ can be found (with probability approx. $\frac{1}{2}$) by hashing $(k!)^{\frac{1}{k}} 2^{\frac{n(k-1)}{k}}$ messages. This is known as the *generalized birthday paradox*.

The herding attack was first presented in [17]. The attacker's work is divided into an offline and online phase. The offline phase happens before the attacker is challenged with a prefix and the online phase happens after the attacker is challenged. The attacker should not be able to use the offline phase to her/his advantage. In the online phase, the attack should require hashing approximately 2^n random messages.

The Trojan message attack was first presented in [1]. However, the complexity of the Trojan message attack against hash function that is a random oracle was calculated for the first time in [22]. We will take a look at this complexity in Chapter 5.

2.3.1 Why Are These Security Properties Important?

Any regularities or undesired properties that the attacker can find in the hash function are potentially dangerous because they give the attacker tools that can possibly be used in some unpredictable way to mount an attack in the future. Thus cryptographers are expecting hash functions to be indistinguishable from random oracles (see [3]).

We will now, briefly give examples on what the breaking of the security resistances presented at the beginning of this section could mean in practice. Our examples are by no means the only ways to use these breaches of security. The Nostradamus attack is an example of another way, see [17].

Assume that we have a situation where the parties $\mathscr{A}$ and $\mathscr{B}$ wish to form a contract in such way that anybody is able to verify later that they have made this contract. Assume furthermore that at the same time $\mathscr{A}$ and $\mathscr{B}$ wish to keep the details of their contract in secret. If $\mathscr{A}$ and $\mathscr{B}$ have access to a hash function $\mathtt{H}$ they can form their contract and use the given hash function to it, receiving a hash value h. After this $\mathscr{A}$ and $\mathscr{B}$ can publish the hash value they received and announce that they have made a contract that hashes to h when using hash function $\mathtt{H}$. If either party later breaks the contract, the other party can publish the contract and show that using `H` it really hashes to h.

It should be clear that if either party, say $\mathscr{A}$, is able to break the preimage resistance or second preimage resistance of `H` the situation becomes unbearable. The party with this kind of advantage can create her/his own version of the made contract that hashes to the same hash value h and claim that she/he never agreed to the original contract.

Breaking collision resistance alone is not directly as harmful. If $\mathscr{A}$ can create a single collision, but has no control over the created messages x and x', her/his ability to work directly in practice is still very limited. However, once the attacker gains the ability to create collisions, there are ways she/he can use to create meaningful messages with desired content. If $\mathscr{A}$ has this kind of ability, she/he can create two contracts x and x' that have the same hash value h and offer x to $\mathscr{B}$, while keeping x' for herself/himself in order to be able to deny ever making contract x.

It is clear that if the generalized collision resistance of H is broken, $\mathscr{A}$ can use the same kind of attack as with collision resistance but even more effectively. Assume now that $\mathscr{A}$ is able to create k different versions of the contract and offer only one of them to $\mathscr{B}$ while keeping the rest of them in secret. Later $\mathscr{A}$ can choose to use one of these forged contracts in order to deny ever making the original one.

The ability to break herding resistance can also be used in the previous contract setting when we assume that $\mathscr{B}$ is satisfied after creating the beginning part of the contract (i.e creating a prefix p) and allows $\mathscr{A}$ to formalize the rest of the contract. In this case $\mathscr{A}$ creates a herding attack and commits to hash value h_d. After this, she/he provides a suffix x_s such that the message px_s hashes to h_d. Later the attacker is able to use her/his ability to perform a herding attack again and create a forged contract that will have the same hash value h_d and include any p' as prefix.

Finally, breaking the Trojan message resistance can be used in an occasion where $\mathscr{B}$ is satisfied in choosing the beginning part of the contract from some set of known messages and $\mathscr{A}$ is free to create the rest of the contract to be a Trojan message t. An example could be a case where $\mathscr{A}$ does not know the day when the contract will be signed, but is allowed to otherwise formalize its details.

2.4 Iterated Hash Functions

The idea of an iterated hash function was presented in [6, 29]. The underlying method is quite simple and easy to implement. The basic method is to read the message in from

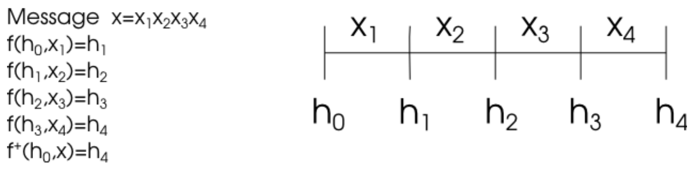


Fig 1. Example of using iterated hash function on a message that is four message blocks long.

left to right, block by block and in each step use the hash value created in previous step, to create a new hash value.

In practice hash functions are often used online in situations, where one does not necessarily have the whole message ready beforehand. An iterated hash function construction allows one to start the hashing process immediately after the first message block of the message is known.

A compression function forms the core of an iterated hash function.

Definition 2. A compression function (of block size m and length n) is a mapping $f: \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ where $m,n \in \mathbb{N}_+$ and $m > n$.

Let $m,n \in \mathbb{N}_+$, $m > n$, and the compression function $f\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ be given. The *iterated hash function* $f^+ : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$ (based on f and the initial hash value $h_0 \in \{0,1\}^n$) is defined as follows: Given the message $x = x_1x_2\cdots x_l$, where $l \in \mathbb{N}_+$, and $x_1,x_2,\ldots,x_l \in \{0,1\}^m$, let $h_i = f(h_{i-1},x_i)$ for $i = 1,2,\ldots,l$. Then $f^+(h_0,x) = h_l$.

Since all messages do not have length that is divisible by m it is necessary to *preprocess* the message by adding extra bits (*padding*) at the end of the message to attain a suitable bit length. This padding usually includes also the length of the original message in bits. This is known as *Merkle-Damgård strengthening*.

For each message $x \in \{0,1\}^*$, let the $x_{MD} \in (\{0,1\}^m)^+$ be the bit string achieved from x after Merkle-Dåmgård strengthening. The standard Merkle-Damgård construction based on compression function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ and initial value $h_0 \in \{0,1\}^n$ is a function $H_{f,h_0} : \{0,1\}^* \to \{0,1\}^n$ defined as follows: for each $x \in \{0,1\}^*$ let $H(x) = f^+(h_0, x_{MD})$.

In this work, for the sake of simplicity, we will assume that all messages can be directly divided into message blocks and we will not worry about the padding. In our attacks we will ensure that all second preimages we create have the same length as the original messages and all messages forming a collision have the same number of message blocks. This means that Merkle-Damgård strengthening is not a problem for our second preimage and multicollision attacks.

We would also like to note that in Definition 2 property $m > n$ is chosen for the sake of simplicity. It is not a necessary property for the attacks presented in this work to be successful. All the attack algorithms presented in chapters 3, 4 and 5 can be build to work also in the case where $m \leq n$. The attacker simply needs to replace single message blocks in the constructions with several blocks (enough to ensure that they contain more than n bits).

When constructing attacks against generalized iterated hash functions that are considered in chapters 6 and 7, situation is slightly more complex. Thus the situation where $m \leq n$ is considered there separately.

2.5 Different Types of Hash Functions

Although the Merkle-Damgård structure has been a successful way of constructing fast and secure hash functions, many of them have been found flawed [10, 33, 35, 36, 38]. Often these flaws come from the weaknesses in the underlying compression functions. However, rigorous mathematical study has also found some weaknesses in the Merkle-Damgård structure itself as we shall see later.

There have also been many hash constructions that are not exactly iterated hash functions, but similar in many respects. We will now present a couple of them.

2.5.1 Concatenated Hash Functions

In practice a natural way to build hash functions of large length is to take hash functions of smaller length and concatenate their results (see for example [28]). This means that if we have iterated hash functions $f_1^+ : \{0,1\}^{n_1} \times (\{0,1\}^m)^+ \to \{0,1\}^{n_1}$ and $f_2^+ : \{0,1\}^{n_2} \times (\{0,1\}^m)^+ \to \{0,1\}^{n_2}$ we can simply set $C(h_{0,1}, h_{0,2}, x) = f_1^+(h_{0,1}, x) || f_2^+(h_{0,2}, x)$, where $h_{0,1} \in \{0,1\}^{n_1}$, $h_{0,2} \in \{0,1\}^{n_2}$ and $f_1^+(h_{0,1}, x) || f_2^+(h_{0,2}, x)$ mean the concatenation of n_1-bits long hash value string $f_1^+(h_{0,1}, x)$ and n_2-bits long hash value string $f_2^+(h_{0,2}, x)$.

Ideally $C : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \times (\{0,1\}^m)^+ \to \{0,1\}^{n_1+n_2}$ should be indistinguishable from a random oracle. This however is not the case and such structure has severe weaknesses that we will later study further.

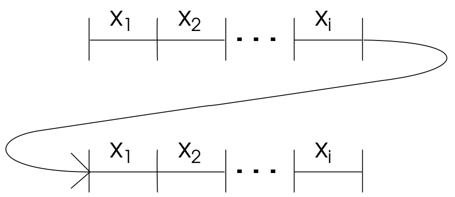


Fig 2. Hash Twice function.

Remark 5. The concatenation is defined in Section 2.1, without $||$ notation. We use notation $||$ when we concatenate two hash values. The purpose of this is to make it clear that we are not multiplying the hash values.

2.5.2 Hash Twice

An easy way to add security to a hash function is to hash the message twice. If we have iterated hash function f^+ with initial value h_0 we can create Hash Twice function T_{f,h_0}, based on compression function f and initial value h_0 simply by choosing $T_{f,h_0}(x) = f^+(f^+(h_0,x),x)$. At first this seems to protect an iterated hash function from most of the weaknesses of the Merkle-Damgård structure, such as from Joux's attack described in Section 3.1. However, security benefits gained by hashing each message twice are not as large as one would hope as we will discover.

2.5.3 Zipper Hash

Zipper Hash was introduced by Moses Liskov [25]. The idea was to construct an ideal hash function from two weak compression functions. However, further study showed that assumed weaknesses in compression functions resulted in fatal flaws, see [31].

Let two compression functions f_1 and f_2 from $\{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ be given. Zipper Hash based on compression functions f_1, f_2 and initial value h_0 is a function $Z_{f_1,f_2,h_0} : (\{0,1\}^m)^+ \to \{0,1\}^n$ defined as follows. For each message $x = x_1 x_2 \cdots x_s$ where x_j is a single message block for all $j \in \{1,2,\cdots,s\}$, $Z_{f_1,f_2,h_0}(x) = f_2^+(f_1^+(h_0,x), x_s x_{s-1} \cdots x_1)$. In this thesis, we assume f_1 and f_2 to be two distinct random oracles. Even this assumption is not enough to protect the Zipper Hash as we will later see.

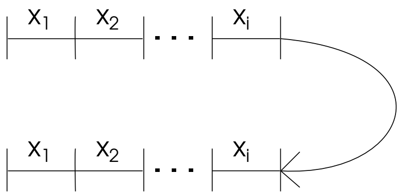


Fig 3. Zipper Hash function.

2.5.4 Generalized Iterated Hash Functions

The basic idea of Hash Twice and Zipper Hash is that a single message block can be used multiple times in the hashing process to gain security. This idea was developed even further to create the so called generalized iterated hash functions that have been studied for example in [14, 30].

Assume now that $l \in \mathbb{N}_+$ and $\alpha \in \mathbb{N}_l^+$ are such that $\alpha = a_1 a_2 \cdots a_s$, where $s \in \mathbb{N}_+$ and $a_i \in \mathbb{N}_l$ for $i = 1, 2, \ldots, s$. Assume that f is as a compression function. Define the *iterated compression function* $f_\alpha : \{0,1\}^n \times (\{0,1\}^m)^l \to \{0,1\}^n$ (based on α and f) as follows: Given the initial value $h_0 \in \{0,1\}^m$ and message $x = x_1 x_2 \cdots x_l$, where $x_1, x_2, \ldots x_l \in \{0,1\}^m$, let $h_i = f(h_{i-1}, x_{a_i})$ for $i = 1, 2, \ldots, s$. Then $f_\alpha(h_0, x) = h_s$.

Now for each $l \in \mathbb{N}_+$, let $\alpha_l \in \mathbb{N}_l^+$ be such that $\mathrm{alph}(\alpha_l) = \mathbb{N}_l$, $\hat{\alpha} = (\alpha_1, \alpha_2, \ldots)$ and $h_0 \in \{0,1\}^n$.

Definition 3. A generalized iterated hash function (based on $\hat{\alpha}$, f and h_0) is a mapping $H_{\hat{\alpha},f} : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$ such that when $x = x_1 x_2 \cdots x_i$, $i \in \mathbb{N}_+$ is a message, where x_j is a message block for all $j \in \{1, 2, \ldots, i\}$: $H_{\hat{\alpha},f}(h_0, x) = f_{\alpha_i}(h_0, x)$.

It is easy to see that if $\hat{\alpha} = (1, 1\cdot 2, 1\cdot 2\cdot 3, \cdots)$ the generalized iterated hash function $H_{\hat{\alpha},f}$ is actually an iterated hash function. If $\hat{\alpha} = (1\cdot 1, 1\cdot 2\cdot 1\cdot 2, 1\cdot 2\cdot 3\cdot 1\cdot 2\cdot 3, \cdots)$ then $H_{\hat{\alpha},f}$ is Hash Twice function.

We can also create a generalized hash function that differs only very slightly from Zipper Hash. If we choose $\hat{\alpha} = (1\cdot 1, 1\cdot 2\cdot 2\cdot 1, 1\cdot 2\cdot 3\cdot 3\cdot 2\cdot 1, \cdots)$ then $H_{\hat{\alpha},f}$ works very much like Zipper Hash. The only difference is that there is only one compression function instead of two.

Call the sequence $\hat{\alpha} = (\alpha_1, \alpha_2, \ldots)$ $q-bounded$, $q \in \mathbb{N}_+$, if $|\alpha_j|_i \le q$ for each $j \in \mathbb{N}_+$ and $i \in \mathbb{N}_+$. The generalized iterated hash function $H_{\hat{\alpha},f}$ is $q-bounded$ if $\hat{\alpha}$ is $q-bounded$.

2.6 Attack Algorithms and Their Complexity

From now on when constructing an attack algorithm against a hash function that use compression function we will assume that the attacker knows, how the hash function depend on the respective compression function f, but models f only as a black box. She/he does not know anything about the internal structure of f and can only make queries on f to get the respective responses (see [3]). We assume f to be a random oracle, which means that the only way to find collisions or preimages on f is through random search. Furthermore, as we have stated, we will assume that the hash function is protected with effective Merkle-Damgård strengthening.

Since f is a random oracle, it is clear that the attacker cannot perform successful preimage attack in any other way than through random search. The same naturally goes for second preimage attacks against messages with just a single message block. It is also quite easy to show that, if f is collision resistant, then the respective hash function is collision resistant [6, 29].

However, despite of these assumptions, there are still ways to create attacks. These attacks make use of the Merkle-Damgård structure of the hash function to create regularities that are then used to mount attacks. It has been shown that given assumptions do not mean that the hash function achieves the generalized collision resistance, herding resistance, Trojan message resistance or second preimage resistance (when the original message is long) of a random oracle hash function.

From now on we will assume that every message x can be written as $x = x_1x_2\cdots x_r$, where $r \in \mathbb{N}_+$ and x_i is a message block of m bits for every $i \in \{1,2,\cdots,r\}$. The *length* of the message x is marked as $|x|$ and it is the number of message blocks in x i.e., $|x| = r$.

2.6.1 The Complexity of the Attack Algorithm

The expected number of required compression function calls is a natural way to measure the efficiency of some attack algorithm. Many papers (see for example [15, 17, 18]) approach the number of compression function calls required through intuitive approximations. For example in [15] (Subsection 4.1) it is simply stated that if a compression function f gives out a hash value with n_f bits then it has a security level of $2^{n_f/2}$ with respect to collision resistance and thus the attacker needs approximately

$t \cdot 2^{n_f/2}$ compression function calls to repeat the collision attack successfully for t times. The same kind of reasoning can be found in [1] and [18].

This sort of an estimation is of course somewhat rough, but can still very well be useful. It gives roughly the expected number of compression function calls the attacker should need in order to complete the attack. We shall later see that the probability of finding a collision among $2^{n_f/2}$ messages is approximately 0.4. Thus it seems reasonable to assume that the amount of required compression function calls to repeat this successfully, would be somewhere near $t \cdot 2^{n_f/2}$.

2.6.2 Asymptotic Complexity

Many articles (see for example [5, 19, 30]) consider the complexity of the attack through the so called big O notation. One writes that $f(x) = O(g(x))$ if and only if there exists a constant r and a real number x_0 such that $|f(x)| \leq r|g(x)|$ for all $x > x_0$. The big O notation is used to give insight to the upper bound of the complexity of the attack algorithm. Big Omega notation, where one writes $f(x) = \Omega(g(x))$ if and only if there exists a constant r and a real number x_0 such that $|f(x)| \geq r|g(x)|$ for all $x > x_0$, is used to give insight to the lower bound of the complexity of the attack. If these two coincide, we can use the big Theta notation, i.e. $f(x) = \Theta(g(x))$ if and only if there exists constants r_1, r_2 and a real number x_0 such that $r_1|g(x)| \leq |f(x)| \leq r_2|g(x)|$ for all $x > x_0$.

The big O notation is a useful tool, when we wish to examine the complexity of some attack algorithm from the theoretical point of view. It allows us to simplify the calculations, since we do not have to worry about the constant factors of the attack. At the same time we still get a good insight to the complexity that the attack algorithm has.

For example we can simply state that the complexity of finding a collision through random search is $O(2^{n/2})$, since it is clear that the expected number of compression function calls required to create a collision is $O(2^{n/2})$. Thus we can easily deduce that the complexity of finding k different collisions is $O(k \cdot 2^{n/2})$, since we can create collisions one at a time and add the expected number of compression function queries needed together.

However, from a practical point of view the attack algorithms are applied in situations that are not asymptotic. For example the complexity $O(k \cdot 2^{n/2})$ is equal to $O(2^{n/2})$ as long as we assume that k is constant. This means that the complexity of creating 2^{80} collisions is $O(2^{n/2})$. On the other hand for hash functions with 160 bits hash values

$2^{80} \cdot 2^{160/2} = 2^{160}$ that is the number of compression function calls required to create a preimage attack with probability $1-(1-\frac{1}{2^{160}})^{2^{160}} \approx \frac{e-1}{e}$. From this point of view it is of course good to have an upper bound of the complexity evaluated also without the big O notation, if this is possible.

2.6.3 Complexity Analysis in This Work

In this work we have chosen to define the *complexity of an attack algorithm* as the expected number of queries on compression function f required to complete the attack. This kind of complexity is often called the time complexity of the attack. We will go through algorithms in detail to be able to give certain upper bounds for required complexities.

Our attacks are build as stepwise statistical experiments in a manner where the probability that the attack fails altogether is so small that it can be ignored. Thus each step of the attack is a statistical experiment which, from the viewpoint of the attacker, either succeeds or fails. The attacker keeps repeating the statistical experiment until it succeeds. Clearly this is not an optimal approach, if our aim is to minimize the number of required compression function calls. Thus the given complexities are somewhat rough upper bounds.

There are three reasons, why we have chosen this model. Firstly, the upper bounds for complexities are easy to calculate this way and they certainly hold. Secondly, only the constant multipliers of the complexities are affected by this approach and the aim of this work is not to optimize them. Thirdly, the model allows us to give strict memory requirements for the attacks, when they are performed in the manner described in this work. This is possible since, if some step is unsuccessful, we could simply erase the created messages and hash values from the memory.

2.6.4 On Memory Requirements

In addition to the ability to call the compression function for sufficiently many times, the attacker also needs the ability to store the received hash values. This can of course become problematic, if the attacker is required to store for example $2^{\frac{n}{2}}$ hash values (as is the case for collision attacks) or even more than $2^{\frac{2n}{3}}$ hash values (as is the case for attacks that use a diamond structure that we will later present).

However, there are algorithms the attacker can use to decrease the memory requirements of the attack, but at the cost of increased number of compression function calls. This is called the space-time tradeoff. For example [34] offers an algorithm that allows the attacker to do quite efficient space-time tradeoff, when creating collisions.

We shall evaluate the memory requirements of the attack in two parts. Firstly, we give the *algorithm memory requirement* (AMR) that tells the amount of hash values the attacker should be able to store to ensure that the attack algorithm can be completed without doing any space-time tradeoff at all. This analysis concerns the attacks in the form given in this work.

Secondly, some of the attacks run in offline and online phase and require the attacker to store some construction created in the offline phase (for example a diamond structure). Often it is assumed that the attacker can spend more time at the offline phase and thus it could be wise to add required amount of compression function calls in the offline phase, if this allows the attacker to reduce the number of compression function calls required to complete the online phase.

Thus we measure the number of hash values and message blocks of such stored structure with the *online memory requirement* (OMR). For the sake of simplicity, we assume that it takes the same amount of memory to store a hash value and a message block. The online memory requirement also gives us the amount of memory the attacker needs to complete the online phase with given complexity.

So in short the algorithm memory requirement gives us the amount of memory the attacker needs to complete the whole attack in the given form without doing space-time tradeoff and the online memory requirement gives us the amount of memory the attacker needs to complete the online phase of the attack in the given form without doing space-time tradeoff.

2.6.5 About Wide-Pipe Hash

In [27] a new type of hash function construction, labeled Wide-Pipe Hash, was introduced. The basic structure resembles the structure of iterated hash function that we have presented, but the compression function is $f : \{0,1\}^s \times \{0,1\}^m \to \{0,1\}^s$, where $s, m \in \mathbb{N}_+$ and $s > n$. In the final step of the hashing process a second compression function $f_1 : \{0,1\}^s \to \{0,1\}^n$ is used to compress the last internal hash value (with s bits) to the final hash value (with n bits). A Wide-Pipe Hash function $W_{h_0,f,f_1} : \{0,1\}^s \times (\{0,1\}^m)^+ \to \{0,1\}^n$ based on initial value $h_0 \in \{0,1\}^s$ and compression functions f, f_1 is

$$W_{h_0,f,f_1}(x) = f_1(f^+(h_0,x)).$$

If we choose $s \geq 2n$, then all the attacks we present in this work will have complexity of at least 2^n since we assume the compression functions to be random oracles. The downside of the Wide-Pipe Hash is that in practise one needs more resources to use a compression function with a larger domain and range than a compression function with smaller ones. Thus some new hash function schemes allow the user to decide the size of the final hash value up to the size of the compression function output (see for example [4]).

For the sake of simplicity, we assume in this work that the final hash value of any iterated hash function is of the same size as the output of the compression function i.e. n bits. This means that the efficiencies of presented attacks are compared with the size of the output of the compression function.

2.7 About Collisions

A *k-collision* on hash function $\mathtt{H} : \{0,1\}^* \to \{0,1\}^n$ is a set $X \subseteq \{0,1\}^*$ such that $|X| = k$ and $\mathtt{H}(x) = \mathtt{H}(y)$ for all $x, y \in X$. In this work, we will also demand $|x| = |y|$ for all $x, y \in X$ to ensure that Merkle-Damgård strengthening is not a problem. A concept of k-collision is generalized in a natural way to any function $g : \{0,1\}^n \times A \to \{0,1\}^n$ where $A \subseteq \{0,1\}^*$. Usually $2-$collisions are simply referred to as collisions while larger collisions are referred to as *multicollisions*.

Assume, that the hash function $\mathtt{H}$ is a random oracle. Assume furthermore that the attacker wants to minimize the number of hash function calls needed to create a collision. Since the only way to attack is to make random queries to $\mathtt{H}$, the best possible attack algorithm is to create new messages and calculate respective hash values one at a time and stop the process immediately when the collision occurs. This means that the attacker creates first message x_1 and calculates hash value $\mathtt{H}(x_1)$. Then the attacker creates x_2, calculates $\mathtt{H}(x_2)$ and checks if $\mathtt{H}(x_1) = \mathtt{H}(x_2)$. If this is not the case, the attacker creates x_3, calculates $\mathtt{H}(x_3)$ and checks if $\mathtt{H}(x_3) = \mathtt{H}(x_1)$ or $\mathtt{H}(x_3) = \mathtt{H}(x_2)$ and so on. The process continues until a collision is found.

We will now take a look at probability and complexity of finding a collision with more detail. From the attacker's point of view we are especially interested in the expected number of compression function calls that are needed to create a collision, when using an iterated hash function.

2.7.1 The Probability of Finding a Collision

Assume that the attacker creates k message blocks and calculates the respective hash values. According to Lemma 1 the probability that there is a collision is approximately $1 - e^{-\frac{k(k-1)}{2^{n+1}}}$, when n is large. This means that the probability of finding a collision is approximately $1 - e^{-\frac{k^2}{2^{n+1}}}$ when we assume that also k is large. Subsection 7.1 in [8] offers the same result using different technique for the proof.

2.7.2 Collision Complexity

We will now determine an upper bound for the complexity of finding a collision on an iterated hash function. Assume that the compression function f is a random oracle and h_0 is a hash value. This means that the only possible way to find two message blocks x_1 and x_2 such that $x_1 \neq x_2$ and $f(h_0, x_1) = f(h_0, x_2)$ is through random search. What is the complexity of this kind of attack?

One mathematically robust approach can be found for example in [19, 30]. In this approach we create a set of $2^{n/2}$ message blocks, calculate the respective hash values and search for a collision. If no collision is found, we repeat the process. We keep creating a set of $2^{\frac{n}{2}}$ messages until a collision is found. In [30] it is evaluated that since the probability of finding a collision in $2^{\frac{n}{2}}$ randomly created message blocks is aproximately $1 - e^{-\frac{1}{2}} \approx 0.4$, the expected number of compression function calls is $2.5 \cdot 2^{n/2}$ (see Fact 2). The algorithm memory requirement of this attack is $2^{\frac{n}{2}}$.

Clearly, this is not the best possible way to create a collision if our aim is to minimize the number of compression function calls. The best way to attack is to create new message blocks one at a time and after each new message block calculate the respective hash value and see, if we have created a collision. Assuming that the attacker can ignore the memory requirements, we can use Theorem 2 of [12] to deduce that the expected value of compression function calls required is approximately $\sqrt{\frac{\pi}{2}} \cdot 2^{\frac{n}{2}}$, when n is large. In this work we shall, however, use the model, where a step is repeated, if it is unsuccessful.

We will now optimize the complexity of this approach, where we choose a large set of message blocks, calculate the respective hash values and then repeat the process, if no collisions are found. Assume that we create a message set with k message blocks, calculate the respective hash values and search for a collision. The number of compression function calls needed is k, while the probability of finding a collision can be evaluated to be $1-e^{-\frac{k^2}{2^{n+1}}}$ according to previous subsection. This means that the expected number of compression function calls needed, when we assume n to be large, according to Fact 1 and Fact 2 is

$$\frac{k}{1-e^{-\frac{k^2}{2^{n+1}}}}.$$

We will now try to find the minimal value of this function with respect to k. Taking the derivative with respect to k and setting it to zero gives us an equation

$$1-(e)^{-\frac{k^2}{2^{n+1}}}+(e)^{-\frac{k^2}{2^{n+1}}}\cdot\left(-\frac{2k}{2^{n+1}}\right)\cdot k=0$$

that holds if and only if

$$e^{\frac{k^2}{2^{n+1}}}=1+\frac{k^2}{2^n}.$$

For the sake of simplicity we now set $k=s\cdot 2^{n/2}$ and get a new equation

$$e^{\frac{s^2}{2}}=1+s^2.$$

It is now easy to evaluate the value of s by using for example the Maple program and see that the only positive real root is $s\approx 1.585201$ and so $k\approx 1.585201\cdot 2^{n/2}$. From now on we will mark this positive real root with $\check{s}$. Using $s=\check{s}$ we get the total complexity of our attack to be approximately

$$\frac{\check{s}}{1-e^{\frac{\check{s}^2}{2}}}\cdot 2^{\frac{n}{2}}\approx(2.21606)\cdot 2^{\frac{n}{2}}.$$

This is a slight improvement of $2.5\cdot 2^{\frac{n}{2}}$ found in [30]. From now on we will mark

$$\frac{\check{s}}{1-e^{\frac{\check{s}^2}{2}}}=\check{a}$$

and thus the complexity of finding a collision is $\check{a}\cdot 2^{\frac{n}{2}}$. The algorithm memory requirement of this attack is $\check{s}\cdot 2^{\frac{n}{2}}$.

2.7.3 Multicollision Complexity

As we have stated before, according to the (generalized) *birthday paradox*, given any hash function H of length n (random oracle hash functions included), a k-collision for H can be found (with probability approx. $\frac{1}{2}$) by hashing $(k!)^{\frac{1}{k}}2^{\frac{n(k-1)}{k}}$ messages [32].

It is easy to see that given an integer k, where $2 < k \leq n$ when n is sufficiently large, finding a $(k+1)$-collision consumes much more resources than finding a k-collision in the case of the random oracle hash function. This is especially true, when k is relatively small. The complexity of finding a collision is $O(2^{\frac{n}{2}})$ while the complexity of finding a $3-$collision is $O(2^{\frac{2n}{3}})$.

Later in this work the method invented by Joux [15] is repeatedly used to create attacks on different kinds of hash functions in order to create k-collisions with complexity well below $(k!)^{\frac{1}{k}} 2^{\frac{n(k-1)}{k}}$. In other words, it is shown that it is difficult to create iterative hash functions that achieve the same generalized collision resistance as a random oracle hash function, even when we assume that the compression function is a random oracle.

2.7.4 Preimage and Second Preimage Complexities

Assume that the compression function f is a random oracle and f^+ is an iterated hash function with initial value h_0. It is relatively easy to calculate the number of compression function queries required to create a preimage for any given hash value h. The only possible way the attacker can attack is through random search. The best possible way to do this is to create a random message block x, calculate respective hash value and see if $f(h_0,x) = h$. If this is not the case the attacker repeats the process. The probability that $f(h_0,x) = h$ is 2^{-n}, so the expected number of message blocks and thus the compression function queries required is 2^n according to Fact 2.

If f^+ would work as a random oracle, then the same reasoning should apply to a second preimage attack. However in the next chapter we shall see that this is not the case, if the original message is long.

3 What is Known Before

In this chapter we give a brief survey of some previously proven weakness results for iterated hash functions. We shall introduce the following types of approaches and their applications: Joux's attack, expandable messages, diamond structures and Trojan messages attacks. We shall study some attacks that apply these using our attack model where each step is repeated if it is unsuccessful. This allows us to give rough but certain upper bounds for the expected value of compression function calls required to complete these attacks when the attacker has the required memory available.

Some of the following attacks contain a theoretical (and extremely small) possibility that they do not succeed. We have made some small changes to attacks presented in previous literature to ensure that we can disregard the possibility of unsuccessful attack from our complexity considerations. If the attack has not succeeded after 2^n compression function calls (or $2^{n_1+n_2}$ for concatenated hash functions) the attacker should stop and start it all over again.

As an example let us consider a situation where the attacker has a single hash value h', a set of hash values $\{h_1, h_2, \ldots, h_{2^{n/3}}\}$ and she/he has to find a message block x such that $f(h', x) = h_i$ for some $i \in \{1, 2, \ldots, 2^{n/3}\}$ or the attack fails. Now there are 2^m different message blocks and $m > n$, but it is still theoretically possible that there is no message block that satisfies the required property.

The possibility that this is the case, is of course extremely small (less than $e^{-2^{\frac{n}{3}}}$). However, in such a rare situation the attacker should give up after 2^n compression function calls and start the whole attack again. We wish to point out that in practice this condition is effectively meaningless since, if the attacker can call the compression function for 2^n times, she/he is able to create a preimage for any hash value with high probability. We have included it for a theoretical purpose only, i.e. to avoid a situation, where a failed attack would result in an infinite complexity.

3.1 Joux's Attack

In [15], Antoine Joux introduced an easy way to construct 2^k-collision for an iterated hash function f^+ with the same amount of work it takes to create k times an ordinary collision. Joux's attack works as follows.

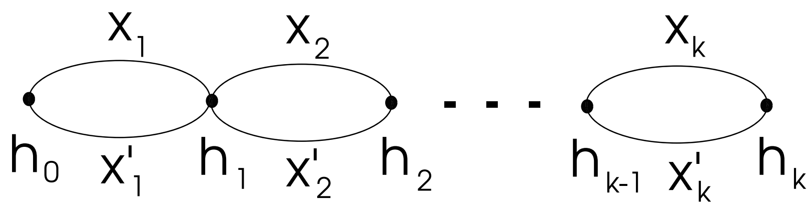


Fig 4. Joux's method of multicollision creation.

Assume that iterated hash function has the initial value h_0. First we will simply search for two message blocks x_1 and x_1' such that $x_1 \neq x_1'$, $f(h_0,x_1) = f(h_0,x_1')$. We will find two such message blocks, even if f is a random oracle. Let us mark $h_1 := f(h_0,x_1) = f(h_0,x_1')$.

Next we start searching for message blocks x_2,x_2' such that $f(h_1,x_2) = f(h_1,x_2')$ and mark $h_2 := f(h_1,x_2) = f(h_1,x_2')$. As before we can use the birthday paradox, if we do not have a better way to attack against f. We continue this process; when we have calculated the value h_i we will search for message blocks x_{i+1},x_{i+1}' such that $f(h_i,x_{i+1}) = f(h_i,x_{i+1}')$ and denote $h_{i+1} = f(h_i,x_{i+1}) = f(h_i,x_{i+1}')$.

When we have completed k such steps, we have created our multicollision $M = M_1M_2\cdots M_k$, where $M_i = \{x_i,x_i'\}$ for $i \in \{1,2,\cdots,k\}$. There are 2^k different messages in M, each of them k blocks long. Next we will evaluate how complex such an attack could be, if we have to rely on the birthday paradox to produce multicollisions.

Let us suppose that Joux's attack is carried out as a statistical experiment in the following way. In step i of the attack we create a set X of $2^{\frac{n}{2}}$ message blocks. Then we calculate hash values $f(h_{i-1},x)$ for all $x \in X$. Next we look for a collision, in other words message blocks $x,x' \in X$ such that $x \neq x'$ and $f(h_{i-1},x) = f(h_{i-1},x')$ and denote $h_i = f(h_{i-1},x)$. In k steps this method creates a collision with size 2^k.

As we have seen in subsection 2.7.2, the probability for finding a collision in $2^{\frac{n}{2}}$ messages is approximately 0.4. We repeat the procedure until a collision is found. The expected number of repetitions is 2.5 according to Fact 2. This implies that the expected complexity of creating a 2^k-collision with Joux's attack as described above is approximately $2.5 \cdot k \cdot 2^{\frac{n}{2}}$ with the algorithm memory requirement $2^{\frac{n}{2}}$.

Remark 6. If the attacker is able to store $\check{s} \cdot 2^{\frac{n}{2}}$ hash values, the results of the subsection 2.7.2 allow us to decrease this complexity to $\check{a} \cdot k \cdot 2^{\frac{n}{2}} \approx 2.2 \cdot k \cdot 2^{\frac{n}{2}}$ by choosing $\check{s} \cdot 2^{\frac{n}{2}}$ as the size of the created message set. If the attacker ignores the memory requirements altogether and performs the attack in the optimal way, i.e. creates the new message blocks one by one and ends the process, when collision is found, the complexity drops to $\sqrt{\frac{\pi}{2}} \cdot k \cdot 2^{\frac{n}{2}}$ (based on [12]).

3.1.1 Joux's Attack on Concatenated Constructions

In [15] also a clever way to attack against concatenated hash functions is presented. Assume that we have two iterated hash functions $f_1^+ : \{0,1\}^{n_1} \times (\{0,1\}^m)^+ \to \{0,1\}^{n_1}$ and $f_2^+ : \{0,1\}^{n_2} \times (\{0,1\}^m)^+ \to \{0,1\}^{n_2}$, where f_1^+ is based on the compression function f_1 and the initial value $h_{0,1}$ and f_2^+ is based on the compression function f_2 and the initial value $h_{0,2}$. Assume further that $n_1 \leq n_2 < m$ and the concatenated hash function C is defined by setting $C(h_{0,1}, h_{0,2}, x) = f_1^+(h_{0,1}, x) || f_2^+(h_{0,2}, x)$.

We can now simply use Joux's attack for f_1^+ to a create message set $M = M_1 M_2 \cdots M_{n_2}$ where $M_i = \{x_i, x_i'\}$, $|M| = 2^{n_2}$ and mark $f_1^+(h_{0,1}, x) = h_{n_2}$ for all $x \in M$. The complexity of this is approximately $2.5 \cdot n_2 \cdot 2^{\frac{n_1}{2}}$ as we have seen.

Now it is possible to simply execute a birthday attack to f_2^+ with messages from M. The complexity of finding messages x and x' such that $x, x' \in M$, $x \neq x'$ and $f_2^+(h_{0,2}, x) = f_2^+(h_{0,2}, x')$ is approximately $2.5 \cdot 2 \cdot 2^{\frac{n_2}{2}}$.

It is worth noticing that the first impression would be that the complexity is approximately $2.5 \cdot n_2 \cdot 2^{\frac{n_2}{2}}$ since all $x \in M$ have the length of n_2 message blocks. However, the tree like structure of the multicollision means that the attacker can lower this complexity. To create $2^{\frac{n_2}{2}}$ messages and calculate the respective hash values, the attacker can simply fix $\frac{n_2}{2}$ first message blocks and use only the set $M_{\frac{n_2}{2}+1} M_{\frac{n_2}{2}+2} \cdots M_{n_2}$. Thus the number of required compression function calls is only $\frac{n_2}{2} + 2 + 4 + \cdots + 2^{\frac{n_2}{2}} \approx 2 \cdot 2^{\frac{n_2}{2}}$.

Now since $x, x' \in M$ and $f_2^+(h_{0,2}, x) = f_2^+(h_{0,2}, x')$ clearly

$$C(h_{0,1}, h_{0,2}, x) = C(h_{0,1}, h_{0,2}, x').$$

Complexity of creating a collision for a hash function with $n_1 + n_2$ bits hash value through random search is more than $2^{\frac{n_1+n_2}{2}}$. This is certainly a lot larger than $2.5 \cdot (n_2 \cdot 2^{\frac{n_1}{2}} + 2 \cdot 2^{\frac{n_2}{2}})$ if n_1 is of any significant size.

The result above means that creating a collision for a concatenated hash function with two iterated hash functions is well below the complexity of a brute force attack. As has been shown in [15], the attacker's ability to use Joux's attack, however, does not end here. It is possible to use Joux's multicollision attack to create preimage and second preimage attacks on the concatenated hash function.

Assume that we have a concatenated hash function C as above and the hash value $h' = h'_1||h'_2$ where h'_1 has the length of n_1 bits and h'_2 has the length of n_2 bits. We can create a multicollision M, with size 2^{2n_2}, for function f_1^+ as above with complexity $2.5 \cdot 2 \cdot n_2 \cdot 2^{\frac{n_1}{2}}$. Now we have $|M| = 2^{2n_2}$ and $f_1^+(h_{0,1}, x) = h_{2n_2}$ for all $x \in M$. After this we simply search for a single message block x_{2n_2+1} such that $f_1(h_{2n_2}, x_{2n_2+1}) = h'_1$. The complexity of finding such a message block is usually stated to be approximately 2^{n_1}. We will now evaluate the complexity using our definition, where the attack complexity is the expected number of compression function calls.

The attacker creates a random message block x and calculates the hash value $f_1(h_{2n_2}, x)$. If $f_1(h_{2n_2}, x) = h'_1$ the attacker has found the required message block. If not, the attacker repeats the procedure until the required message block is found. If we assume that the f_1 is a random oracle, then probability that $f_1(h_{2n_2}, x) = h'_1$ is 2^{-n_1}. According to Fact 2 the expected number of compression function calls required is indeed 2^{n_1}.

Next we can search for a message x' such that $x' \in M$ and $f_2^+(h_{0,2}, x' x_{2n_2+1}) = h'_2$. As before we can prove by using the tree like structure of the set M that finding such a message x' has the complexity approximately $2 \cdot 2^{n_2}$. This means that the total complexity of the preimage attack is at most $2.5 \cdot 2 \cdot n_2 \cdot 2^{\frac{n_1}{2}} + 2^{n_1} + 2 \cdot 2^{n_2}$ while the complexity of the preimage attack using a random search is $2^{n_1+n_2}$. The algorithm memory requirement of both attacks is $2^{\frac{n_2}{2}}$.

It is clear that it is possible to create a second preimage attack against any message in the same manner.

Remark 7. In [15] the multicollisions that are used as a base of collision and (second) preimage attacks, are smaller than in this work ($2^{n_2/2}$ for a collision attack and 2^{n_2} for a preimage and a second preimage attack). We have decided to use larger multicollisions to ensure that we can disregard the possibility of failure from our complexity analysis.

3.1.2 Multicollisions on Generalized Hash Functions

As we have seen, Joux's attack allows us to create 2^k-collision on iterated hash function with the complexity $2.5 \cdot k \cdot 2^{n/2}$. This means that by increasing the complexity linearly, we can increase the size of the created multicollision exponentially. The question arises whether or not the ideas of Joux's can be applied in a broader setting, i.e., can Joux's approach be used to create multicollisions on generalized iterated hash functions? This question has been studied in [14, 30] and is studied in detail in Chapters 6 and 7.

3.2 Creating Expandable Messages

With an *expandable message* we mean a set of messages X, such that for any two distinct messages $x,x' \in X$ we have $|x| \neq |x'|$ and $f^+(h_0,x) = f^+(h_0,x')$. This means that without Merkle-Damgård strengthening, X would be a multicollision with messages of varying length. Expandable messages are used to create second preimage attacks against long messages, as was discovered in [7, 18].

The first use of expandable messages against an iterated hash function f^+ has been carried out with fixed points [7]. A *fixed point* of compression function f is a pair (h,x) such that $f(h,x) = h$. It is possible to find fixed points easily only, if the compression function f is weak. If we assume that f is a random oracle, as we do in this work, the complexity of creating a fixed point is approximately 2^n which means that there is no efficient way to produce them.

However, later Kelsey and Schneier showed [18] that fixed points are not necessary for the creation of expandable messages. Expandable messages can be created even, when the compression function is a random oracle. We will now show how.

We start from the initial value h_0 and first search for a pair of colliding messages, where the first one has the length of one block and the other one has the respective length $2^{k-1}+1$, $k \in \mathbb{N}_+$ message blocks. We name the respective hash value to h_1 and launch a new collision attack using it as an initial value. This time we are searching for a colliding pair of messages, where the first has the length of one and the second is of length $2^{k-2}+1$. Once such a collision is found we name the respective hash value to h_2 and continue the attack as above until we find hash value h_k. We call this final hash value *expanding value*.

Now it is possible to choose any integer s between k and $k+2^k-1$ and find a message x that satisfies conditions $|x| = s$ and $f^+(h_0,x) = h_k$. In [18] it is assumed that the complexity of each step of creating an expandable message is approximately $2^{\frac{n}{2}+1}$, so the total complexity of the attack is approximately $k \cdot 2^{\frac{n}{2}+1}$.

In this work we define the complexity through the expected number of compression function calls needed to complete the attack. From this point of view we can apply Lemma 2 and deduce that the probability to complete each step successfully, directly by creating two sets of messages with $2^{\frac{n}{2}}$ messages in each of them, is around

$$1-(1-\frac{1}{2^n})^{2^n} \approx \frac{e-1}{e}$$

for a sufficiently large n. If we now assume that the attacker repeats the step, if it fails until it succeeds, we can apply Fact 2 and conclude that the expected number of compression function calls needed is $2 \cdot k \cdot \frac{e}{e-1} \cdot 2^{\frac{n}{2}}$ with the algorithm memory requirement of $2^{\frac{n}{2}+1}$. There are approximately 2^k message blocks in an expandable message with maximum length of $2^k + k - 1$, so the online memory requirement of such message is about 2^k. It is possible to lower this for example by using the same message block more than once.

3.3 Second Preimage Attacks with Expandable Messages

Kelsey and Schneier presented a second preimage attack using expandable messages [18]. The attack is effective only against messages with large number of message blocks.

Assume, that we have a message $y = y_1 y_2 \cdots y_{2^l}$, where y_i is a message block for all $i \in \{1, 2, \cdots 2^l\}$. We begin our attack by creating an expandable message with initial hash value h_0, the minimum length of l and the maximum length of $2^l + l - 1$. The complexity of this is at most $2 \cdot \frac{e}{e-1} \cdot l \cdot 2^{\frac{n}{2}}$ with the algorithm memory requirement $2^{\frac{n}{2}+1}$. Let us assume that the expandable message gives us a hash value h_v.

Next we search for a message block x_v such that $f(h_v, x_v) = f^+(h_0, y_1 y_2 \cdots y_i)$ for some $i \in \{l+1, l+2, \cdots, 2^l\}$. In [18] it is stated that the attacker needs to call the compression function for approximately 2^{n-l} times to find such x_v. We will now verify this.

The attacker creates a random message block x and calculates the hash value $f(h_v, x)$. If $f(h_v, x) = f^+(h_0, y_1 y_2 \cdots y_i)$ for some $i \in \{l+1, l+2, \cdots, 2^l\}$, the attacker has found the required message block. If not, the attacker repeats the procedure until the required message block is found. Assume that f is a random oracle and $f^+(h_0, y_1 y_2 \cdots y_i) \neq f^+(h_0, y_1 y_2 \cdots y_j)$ for all $i, j \in \{1, 2, ..., 2^l\}$, $i \neq j$, then probability that $f(h_v, x) = f^+(h_0, y_1 y_2 \cdots y_i)$ for some $i \in \{l+1, l+2, \cdots, 2^l\}$ is $\frac{2^l - l}{2^n}$ that can be approximated to be 2^{l-n} for sufficiently large n and l. According to Fact 2 it follows that the expected number of compression function calls required is approximately 2^{n-l}.

Now it is a simple task to create a second preimage by choosing message $x_r x_v y_{i+1} y_{i+2} \cdots y_{2^l}$, where x_r is the expandable message chosen so that $|x_r| = i - 1$. Thus we have created a new message with the same hash value and length as the original one.

The total complexity of the attack is $\frac{e}{e-1} \cdot l \cdot 2^{\frac{n}{2}+1} + 2^{n-l}$. Since usually the length of an even long message is well below $2^{\frac{n}{2}}$ message blocks, we have $2^{n-l} \gg 2 \cdot \frac{e}{e-1} \cdot l \cdot 2^{\frac{n}{2}}$,

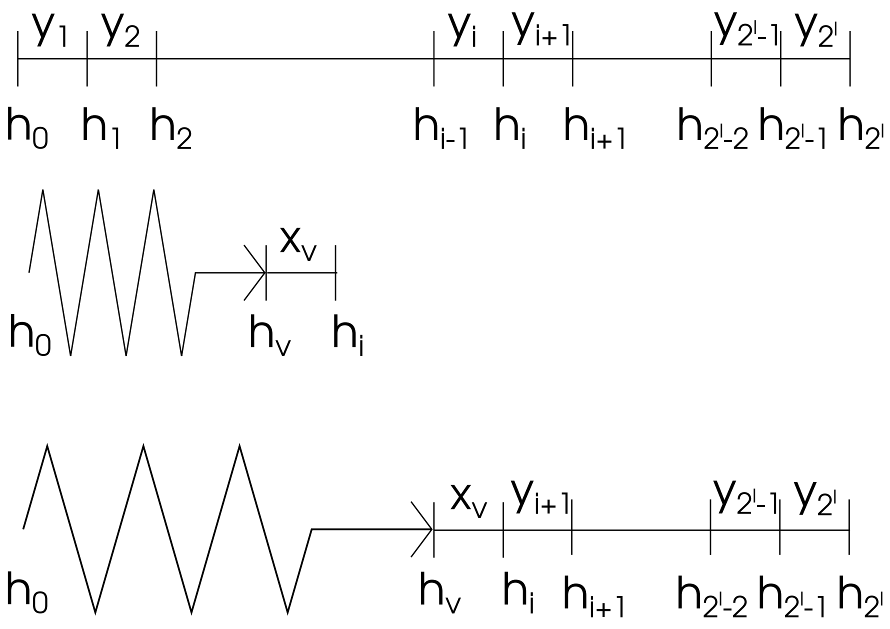


Fig 5. Second preimage attack with expandable message.

where the notation $a \gg b$ means that a is much bigger than b. In this case the complexity of this attack is about 2^{n-l} and the algorithm memory requirement is $2^{\frac{n}{2}+1}$.

3.4 Creating a Diamond Structure

Collision trees or diamond structures have been used in both herding and second preimage attacks against an iterated hash function [1, 2, 17]. The idea of the diamond structure is to take a fairly large set of different hash values and force these to converge towards a single hash value along paths of equal length. We will now offer a more formal definition of a diamond structure by defining it as a binary tree. The following definition can be found in [22].

A *diamond structure* with 2^d chaining hash values or of breadth 2^d, where $d \in \mathbb{N}_+$, is a node labeled and an edge labeled complete binary tree D satisfying the following conditions:

1. The tree D has 2^d leaves, i.e., the height of the tree is d.

2. The nodes of the tree D are labeled by hash values (strings in the set $\{0,1\}^n$) so that the labels of nodes that are at the same distance from the root of D, are pairwise disjoint.
3. The edges of the tree D are labeled by message blocks (strings in the set $\{0,1\}^m$).
4. Let v_1, v_2, and v with labels h_1, h_2, and h, respectively, be any nodes of the tree D such that v_1 and v_2 are the two children of v. Suppose furthermore that x_1 and x_2 are (message) labels of the edges connecting v_1 to v and v_2 to v, respectively. Then $f(h_1,x_1) = f(h_2,x_2) = h$.

It should be noted that the definition above forces quite strict requirements for the diamond structure. It does not, for example, allow the nodes to have more than two children, something that surely would not be a problem when constructing a diamond structure in practice. This is a choice made for the simplicity of notation.

Assume that we have an iterated hash function with the compression function f. The creation method presented in [17] works as follows: We start by storing 2^d chaining hash values (leaves) $a_1, a_2, \cdots a_{2^d}$, where $d \in \mathbb{N}_+$. Next we create $2^{\frac{n-d+1}{2}}$ single-block messages x_i and calculate $f(h_i,x_j)$ for all $i \in \{1,2,\cdots,2^d\}$ and $j \in \{1,2,\cdots,2^{\frac{n-d+1}{2}}\}$. The large number of created hash values gives us reason to expect that collisions will occur. In [17] it is assumed that for each $i \in \{1,2,\cdots 2^d\}$ it is possible to find $j_i \in \{1,2,\cdots,2^{\frac{n-d+1}{2}}\}$ such that set $\{f(a_i,x_{j_i}) | i \in \{1,2,\cdots 2^d\}\}$ contains only 2^{d-1} distinct values. This assumption is not actually proved in [17].

We can repeat this process for $d-1$ times finally reaching a single hash value h' which from now on will be known as the diamond value. Now for each a_i we have created a message x'_i, which is d message blocks long and satisfies $f^+(a_i,x'_i) = h'$. We say that the breadth of the created diamond is 2^d. The length of the diamond is the number of message blocks in each x'_i, where $i \in \{1,2,\cdots,2^d\}$ i.e d. With intuitive reasoning [17] deduces that the complexity of the algorithm is $2^{\frac{n+d}{2}+2}$.

However, a recent study of Blackburn et al. [5] has shown that creating diamond structure is not that simple and that the assertion concerning complexity in [17] is incorrect. It was proven that it is possible to create a diamond structure with the method implied in [17] with complexity $O(\sqrt{d} \cdot 2^{\frac{n+d}{2}})$. The reasoning of [5] is sound. However, in Chapter 5 we shall introduce a new method of creating a diamond structure with $O(2^{\frac{n+d}{2}})$ compression function calls.

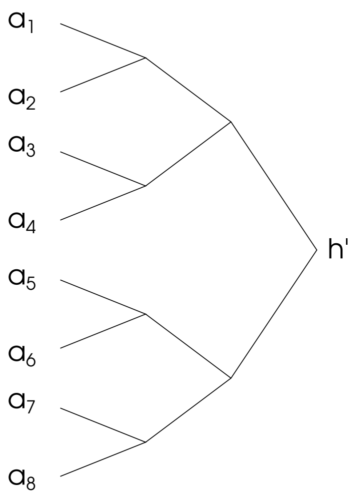


Fig 6. Diamond structure with eight chaining hash values.

In [5] we can also find an upper bound for the complexity of creating a diamond structure (in the proof of Theorem 1 in page 179). This bound is

$$0.83 \cdot \sqrt{d} \cdot 2^{n/2} \frac{2^{1/2}(2^{(d+1)/2} - 1)}{2^{1/2} - 1} \approx 4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}.$$

It should be noted that [5] does not use the same model in complexity analysis as this work. Namely in our model, if a step is unsuccessful it is repeated until it succeeds (and thus it is guaranteed to succeed). Taking such approach would increase the complexity above.

We can also use [5] to get an upper bound for the algorithm memory requirement of the diamond structure creation (equation (3) on page 178). This amount is $0.83 \cdot \sqrt{d} \cdot 2^{(n+d)/2}$. Unlike other algorithm memory requirements computed in this work, this value is not strict, but instead the amount that is needed in the average case. It is, however the best number we have, so we will use it. In Chapter 5 we will present a new method of creating a diamond structure and we are able to give a strict algorithm memory requirement for it, as well as a complexity evaluated in the model, where each step is repeated, if unsuccessful.

The attacker can store a diamond structure with breadth 2^d by storing 2^d chaining hash values and all the message blocks in the structure. This brings the online memory requirement to approximately $3 \cdot 2^d$.

3.4.1 Elongated Diamond Structure

Assume for a moment that the attacker can use long messages i.e. messages with 2^{l+1} message blocks where l is fairly large. In this situation it is possible to create a variant of a diamond structure with significantly lower complexity [17]. We begin by creating 2^d messages x_i with 2^l message blocks $x_{i,j}$ in each of them, that is to say $x_i = x_{i,1}x_{i,2}\cdots x_{i,2^l}$ for $i = 1,2,\cdots 2^d$.

Let the initial hash value be h_0. We can now calculate hash values $h_1 = f^+(h_0,x_1)$, $h_2 = f^+(h_0,x_2),...,h_{2^d} = f^+(h_0,x_{2^d})$. Denote $h_{i,j} = f^+(h_0,x_{i,1}x_{i,2}\cdots x_{i,j})$ for each $i \in \{1,2,\cdots,2^d\}$ and $j \in \{1,2,\cdots,2^l\}$. Altogether 2^{d+l} compression function calls are required to carry out the computations. Next we proceed to construct a diamond structure beginning with hash values $h_1,h_2,\cdots,h_{2^d}$. Let us assume that we have now reached the hash value h' as in a standard diamond structure creation. After this we create an expandable message, starting from the hash value h' with maximum length of 2^l+l-1 and minimum length of l. Assume that the expandable message ends to a hash value h''.

Since we can choose the length of the expandable message freely between l and 2^l+l-1, there exists for each chaining hash value $h_{i,s}$ $(i \in \{1,2,\cdots 2^d\}, s \in \{1,2,\cdots,2^l\})$ a message x' such that $|x'| = 2^l+d+l$ and $f^+(h_{i,s},x') = h''$. We have now created a collision tree of the length 2^l+d+l message blocks possessing 2^{d+l} chaining hash values.

The complexity of creating an elongated diamond structure is thus $2^{l+d} + 4\cdot\sqrt{d}\cdot 2^{\frac{n+d}{2}} + l\cdot\frac{e}{e-1}\cdot 2^{\frac{n}{2}+1}$, with the algorithm memory requirement $0.83\cdot\sqrt{d}\cdot 2^{\frac{n+d}{2}}$. We can usually approximate the complexity to be $4\cdot\sqrt{d}\cdot 2^{\frac{n+d}{2}}$ since typically $2^{l+d} \ll 2^{\frac{n+d}{2}}$ and $l\cdot\frac{e}{e-1}\cdot 2^{\frac{n}{2}+1} \ll 2^{\frac{n+d}{2}}$.

There are approximately $2^{l+d}+2^{d+1}$ message blocks in an elongated diamond structure. In addition we should be able to store 2^{l+d} hash values. Since usually $2^{l+d} \gg 2^{d+1}$ we can thus approximate that the online memory requirement of an elongated diamond structure is approximately 2^{l+d+1}.

3.4.2 Multicollision Diamond

Assume now that we have a Joux's type multicollision M of size $2^{(d+1)n}$. In [1, 11] it is shown that it is possible to create a variant of a diamond structure based on M. The variant structure has 2^d chaining hash values h_i and for each of these a $n\cdot d$ message blocks long message $x_i \in M$ that satisfies $f^+(h_i,x_i) = h'$, where h' is the final hash

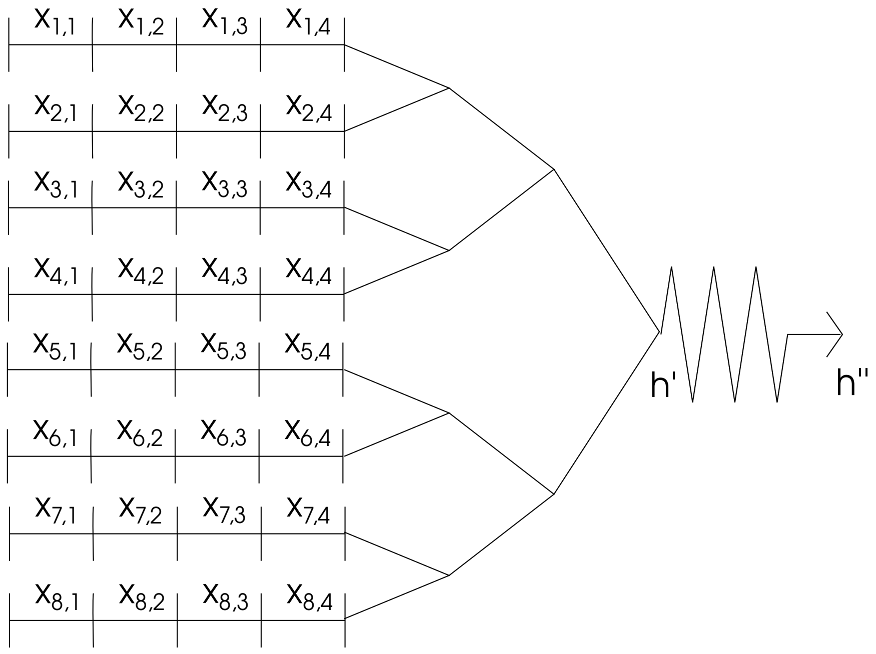


Fig 7. Elongated Diamond Structure with l=2,d=3 and 32 chaining hash values.

value of the structure. In this work we call this structure a *multicollision diamond*. We will now study the mechanism to construct it.

Assume that we are provided with an iterated hash function f^+, some hash values h_0 and message pairs $M_1, M_2, ..., M_{(d+1)n}$ such that for each $i \in \{1,2,...,(d+1)n\}$:

(i) $M_i = \{x_i, x_i'\}$ where x_i and x_i' are distinct message blocks; and
(ii) $f(h_{i-1}, x_i) = f(h_{i-1}, x_i') = h_i$.

We can now choose 2^d chaining hash values $b_1, b_2, \cdots, b_{2^d}$ and create a diamond structure with messages chosen from the message set $M' = M_{n+1}M_{n+2}\cdots M_{(d+1)n}$ as follows. In the set $M_{n+1}M_{n+2}\cdots M_{2n}$ there are 2^n messages, so for each hash value b_i we should be able to find a message x_i such that, $x_i \in M_{n+1}M_{n+2}\cdots M_{2n}$ and $\{f^+(b_i, x_i) | i \in \{1,2,\cdots,2^d\}\}$ contains only 2^{d-1} hash values $b_1', b_2', \cdots, b_{2^{d-1}}'$. We then proceed by searching message x_i' for each b_i' such that $x_i' \in M_{2n+1}M_{2n+2}\cdots M_{3n}$ and $\{f^+(b_i', x_i') | i \in \{1,2,\cdots,2^{d-1}\}\}$ contains only 2^{d-2} hash values. We will then proceed, step by step, in the same manner.

After d steps we should reach a single hash value h_k and for each b_i, $i \in \{1,2,\cdots,2^d\}$ we have created a message y_i such that $f^+(b_i, y_i) = h_k$ and $y_i \in M_{n+1}M_{n+2}\cdots M_{(d+1)n}$. The first impression is that the number of compression function calls required is the same that would be required to create a standard diamond structure multiplied by n (since in each step the length of the messages is n). However, as earlier we can use the tree like structure of the multicollision to lower this complexity, thus resulting to complexity that is approximately $8 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$.

Now we know that there are 2^n messages in the set $M_1 M_2 \cdots M_n$, which means that given any hash value h_b we should be able to find $z_b \in M_1 M_2 \cdots M_n$ such that $f^+(h_b, z_b) = b_i$ for some $i \in \{1, 2, \cdots, 2^d\}$. The complexity of this is approximately $2 \cdot 2^{n-d}$.

The complexity of creating a multicollision diamond is thus $8 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$ with the algorithm memory requirement of around $0.83 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$ and the online memory requirement of approximately $3 \cdot n \cdot 2^{\frac{d}{2}}$. However, all the messages are created from the same multicollision M. This means that it is possible to simply store the hash values $b_1, b_2, \cdots, b_{2^d}$, the message blocks that appear in M and for each b_i the index of the message sequence used to reach h_k. This drops the online memory requirement to approximately $2^d + 2(d+1)n$.

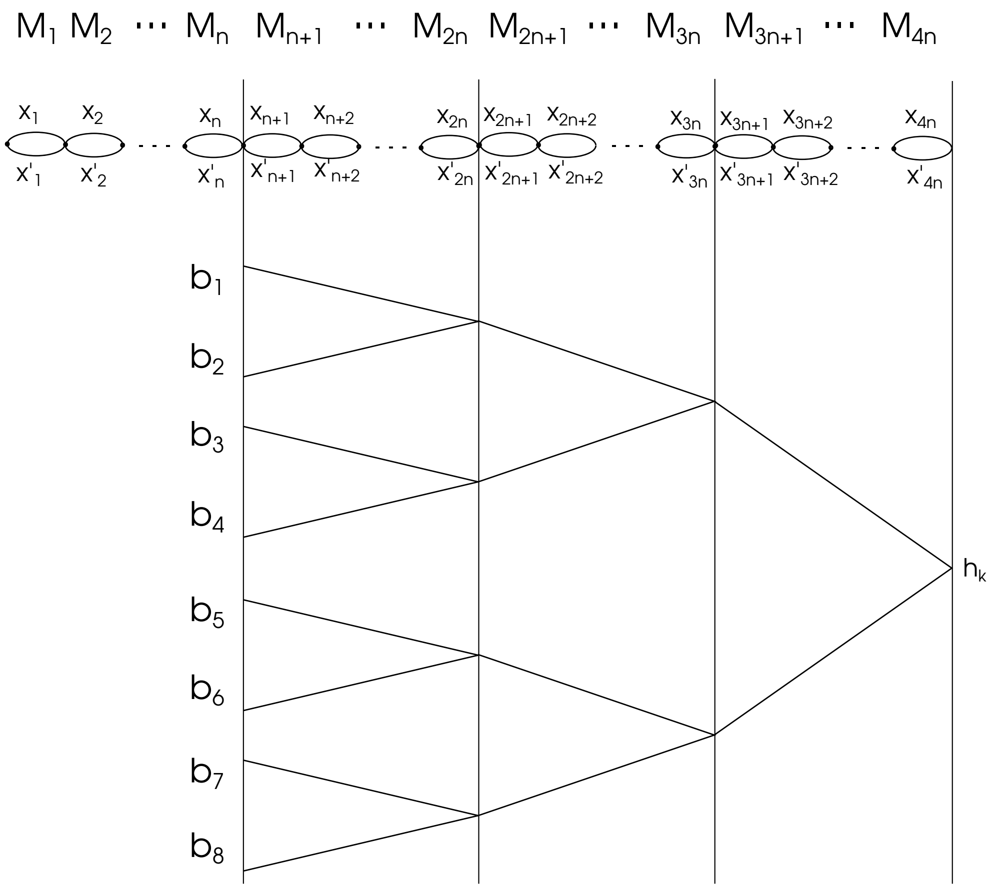
Remark 8. In [1, 11] the multicollisions that are used as a base of the multicollision diamond, are smaller. Again we have chosen the collisions of this size to ensure that we can disregard the possibility that the construction method fails. This choice has very little effect on the actual complexity of the attacks that apply the multicollision diamond.

3.5 Herding Attack with a Diamond Structure

It is possible to use the diamond structure to create a herding attack against an iterated hash function f^+ (see [17]). The basic idea is to create a diamond structure in the offline phase and connect the prefix to it in the online phase.

The attacker creates a diamond structure with 2^d chaining hash values and reaches the diamond value h'. The complexity of this operation is $4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$. The attacker needs to know the length of the message and the final hash value given by the compression function. He commits to the length $|p| + 1 + d$, where $|p|$ is the length of the prefix, and to the hash value h'.

After this the attacker is challenged with a prefix p. She/he then searches for a single message block z_a such that $f^+(h_0, pz_a) = a_i$, where a_i is a chaining hash value for the diamond structure. Several articles state that the complexity of this is 2^{n-d} ([17], [1]). We will now verify this.



M_1 M_2 ⋯ M_n M_{n+1} ⋯ M_{2n} M_{2n+1} ⋯ M_{3n} M_{3n+1} ⋯ M_{4n}
x_1 x_2 x_n x_{n+1} x_{n+2} x_{2n} x_{2n+1} x_{2n+2} x_{3n} x_{3n+1} x_{3n+2} x_{4n}
x'_1 x'_2 x'_n x'_{n+1} x'_{n+2} x'_{2n} x'_{2n+1} x'_{2n+2} x'_{3n} x'_{3n+1} x'_{3n+2} x'_{4n}
b_1
b_2
b_3
b_4
b_5
b_6
b_7
b_8
h_k

Fig 8. Multicollision diamond with d=3.

Let us denote $h_p := f^+(h_0, p)$. The attacker creates a random message block x and calculates the hash value $f(h_p,x)$. If $f(h_p,x) = a_i$ for some $i \in \{1,2,\cdots,2^d\}$, the attacker has found the required message block. If not, then the attacker repeats the procedure, until the required message block is found. If we assume that f is a random oracle, then probability that $f(h_p,x) = a_i$ for some $i \in \{1,2,\cdots,2^d\}$ is 2^{d-n}. According to Fact 2 the expected number of required compression function calls is 2^{n-d}.

Now there exists x_i' such that $f^+(h_0, pz_ax_i') = h'$. The complexity of creating a diamond structure is $4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$, while the complexity of finding z_a is 2^{n-d} giving us the offline complexity of $4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$ and the online complexity of 2^{n-d}. This means that by choosing $d = \frac{n}{3}$, we get the total complexity of $(4 \cdot \sqrt{\frac{n}{3}} + 1) \cdot 2^{\frac{2n}{3}}$.

The algorithm memory requirement of this attack is the same as in a diamond structure creation. In addition the attacker has to be able to store the diamond structure giving this kind of an attack the online memory requirement $3 \cdot 2^d$.

Remark 9. It is possible to use an elongated diamond structure to reduce this complexity. If the attacker is able to create expandable messages and can store the elongated diamond structure, she/he can reduce the total complexity of this attack to approximately $(4 \cdot \sqrt{\frac{n-2l}{3}} + 1) \cdot 2^{\frac{2n-l}{3}}$ by choosing $d = \frac{n-2l}{3}$.

3.5.1 Herding Attack against Concatenated Hash

Diamond structures and multicollision diamonds can be used to create herding attacks against the concatenated hash function as presented in [1, 11].

Assume that we have $f_1^+ : \{0,1\}^{n_1} \times (\{0,1\}^m)^+ \to \{0,1\}^{n_1}$ and $f_2^+ : \{0,1\}^{n_2} \times (\{0,1\}^m)^+ \to \{0,1\}^{n_2}$, where f_1^+ is based on the initial value $h_{0,1}$ and f_2^+ is based on the initial value $h_{0,2}$. Assume that $n_1 \le n_2 < m$ and the concatenated hash function C is defined by setting $C(h_{0,1}, h_{0,2}, x) = f_1^+(h_{0,1}, x) || f_2^+(h_{0,2}, x)$.

In the offline phase the attacker goes through steps:

(1) Create a diamond structure D, on f_1^+. Denote the diamond value of D by h_1.

(2) Create a large Joux type multicollision, on f_1^+ based on h_1. Assume that the final hash value is h_2.

(3) Create a multicollision diamond K based on the multicollision created in step (2) on f_2^+. Assume that the final hash value is h_3.

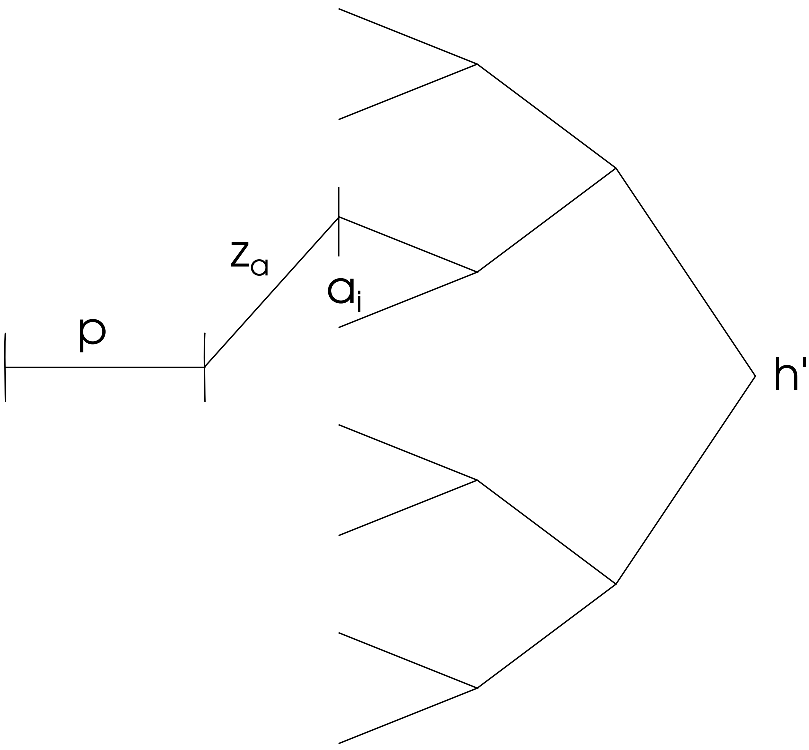


Fig 9. Herding Attack with diamond structure and d=3.

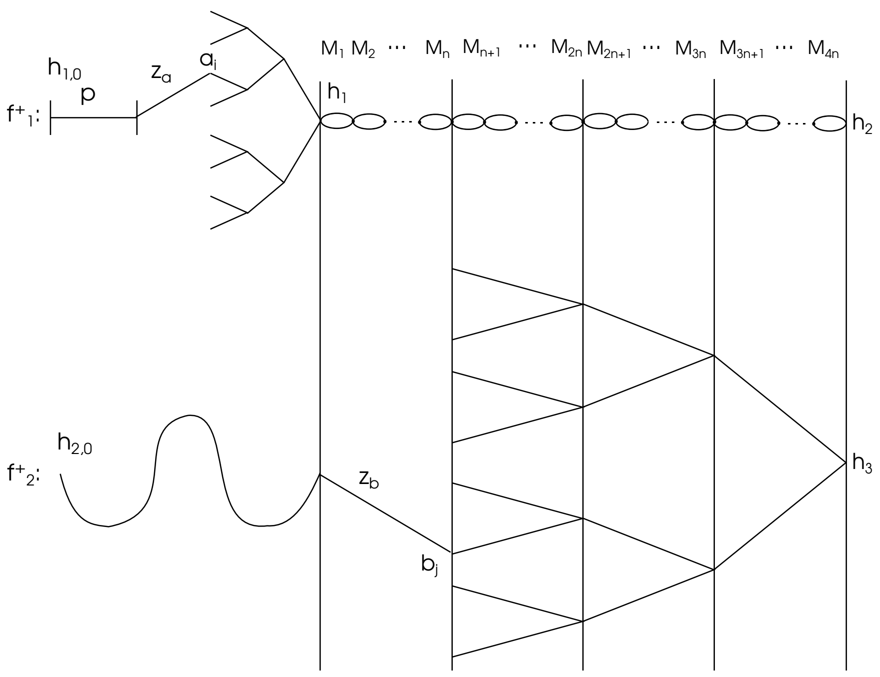
(4) Commit to the hash value $h_2||h_3$ and a message length $|p|+1+d+n_2(d+1)$, where $|p|$ is the length of the prefix in message blocks and the diamond structure has 2^d chaining hash values.

Then in the online phase, where the attacker is challenged with prefix p, the attack proceeds through steps:

(5) Connect the prefix p to the diamond structure D using f_1^+.
(6) Connect the prefix p to the multicollision diamond K using f_2^+.

A detailed description follows. On the first step of the attack the attacker creates a diamond structure D for f_1^+ with 2^d chaining hash values $a_1, a_2, \cdots, a_{2^d}$ and a diamond value h_1. The complexity of this is approximately $4 \cdot \sqrt{d} \cdot 2^{\frac{n_1+d}{2}}$.

In step two, the attacker creates a Joux type multicollision of size $2^{(d+1)n_2}$ for f_1^+ starting from the initial value h_1. In other words the attacker creates a message set $M_1, M_2, ..., M_{(d+1)n_2}$ such that:



$M_1\ M_2 \cdots M_n\ M_{n+1} \cdots M_{2n}\ M_{2n+1} \cdots M_{3n}\ M_{3n+1} \cdots M_{4n}$
$h_{1,0}$
p
z_a
a_i
$f^+{}_1$:
h_1
h_2
$h_{2,0}$
$f^+{}_2$:
z_b
b_j
h_3

Fig 10. Herding attack against concatenated hash.

(i) for each $i \in \{1,2,...,(d+1)n_2\}$, the set M_i consists of two distinct message blocks; and

(ii) $f^+{}_1(h_1,x) = f^+{}_1(h_1,x')$ for all $x,x' \in M = M_1M_2\cdots M_{(d+1)n_2}$.

Let us denote $f_1^+(h_1,x) = h_2$, when $x \in M$. The complexity of the construction is approximately $2.5\cdot(d+1)\cdot n_2\cdot 2^{\frac{n_1}{2}}$.

In step (3) the attacker chooses 2^d new chaining hash values $b_1,b_2,\cdots,b_{2^d}$ and creates a multicollision diamond using f_2^+ and the message set $M = M_1M_2\cdots M_{(d+1)n_2}$. So with the complexity of $8\cdot\sqrt{d}\cdot 2^{\frac{n_2+d}{2}}$ the attacker should be able to produce message y_i for each chaining hash value b_i and multicollision diamond with hash value h_3 such that $y_i \in M_{n_2+1}M_{n_2+2}\cdots M_{(d+1)n_2}$ and $f_2^+(b_i,y_i) = h_3$ for all $i \in \{1,2,\cdots,2^d\}$. The attacker now commits to a message length $|p|+1+d+n_2(d+1)$, where $|p|$ is the length of the prefix, and to hash value a $h_2||h_3$ thus completing the offline phase.

Assume now that the attacker is challenged with a prefix p. On step (5) the attacker searches for a message block z_a such that $f_1^+(h_{0,1},pz_a) = a_i$ for some $i \in \{1,2,\cdots,2^d\}$. The complexity of this is approximately 2^{n_1-d}. Now there is a message x_i such that $f_1^+(a_i,x_i) = h_1$. Next on the step (6) the attacker searches for a message z_b such that $z_b \in M_1 M_2 \cdots M_n$ and $f_2^+(h_{0,2}, pz_a x_i z_b) = b_j$ for some $j \in \{1, 2, \cdots, 2^d\}$. The complexity of finding z_b is approximately $2^{n_2 - d + 1}$.

The attacker now chooses the message $pz_a x_i z_b y_j$. The length of the message is $|p| + 1 + d + n_2(d+1)$ while

$$f_1^+(h_{0,1}, pz_a x_i z_b y_j) = f_1^+(a_i, x_i z_b y_j) = f_1^+(h_1, z_b y_j) = h_2$$

and

$$f_2^+(h_{0,2}, pz_a x_i z_b y_j) = f_2^+(b_j, y_j) = h_3.$$

The offline complexity of this process is $4 \cdot \sqrt{d} \cdot 2^{\frac{n_1+d}{2}} + 2.5 \cdot (d+1) n_2 \cdot 2^{\frac{n_1}{2}} + 8 \cdot \sqrt{d} \cdot 2^{\frac{n_2+d}{2}}$ while the online complexity is $2^{n_1 - d} + 2^{n_2 - d + 1}$. This means that by choosing $d = \frac{n_2}{3}$ the total complexity will be at most $(12 \cdot \sqrt{\frac{n_2}{3}} + 3) \cdot 2^{\frac{2n_2}{3}}$. Clearly, this is well below $2^{n_1 + n_2}$ that is the complexity of the random oracle hash function of length $n_1 + n_2$.

3.5.2 Herding Attack against Hash Twice

It is possible to use the Joux's multicollision attack, a diamond structure and a multicollision diamond to create a herding attack against Hash Twice (see subsection 2.5.2) as stated in [1]. Let T be a Hash Twice hash function that uses the compression function f and the initial value h_0. We will now give the informal framework of the attack. The attack proceeds through the following steps in the offline phase. The attacker creates

(1) a diamond structure D, with a diamond value h_1.
(2) a large Joux type multicollision, based on h_1. Assume that the final hash value is h_2.
(3) a multicollision diamond K based on a multicollision created on step (2). Assume that the final hash value is h_3. Commit to the hash value h_3 and the message length $|p| + 1 + d + n(d+1)$.

Then in the online phase, where the attacker is challenged with prefix p, the attack proceeds through following steps. The attacker connects

(4) the prefix p to the diamond structure D.
(5) the diamond structure D to the multicollision diamond K.

Now we will take a look at what happens in more detail. In the first step of the attack, the attacker creates a diamond structure D with 2^d chaining hash values $a_1, a_2, \cdots, a_{2^d}$ and a diamond value h_1. The complexity of the procedure is $4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$.

In the second step, the attacker creates a Joux's type of multicollision of size $2^{(d+1)n}$ for f^+ starting from the initial value h_1. In another words the attacker creates a message set $M_1, M_2, ..., M_{(d+1)n}$ such that:

(i) for each $i \in \{1, 2, ..., (d+1)n\}$, the set M_i consists of two distinct message blocks; and

(ii) $f^+(h_1, x) = f^+(h_1, x')$ for all $x, x' \in M = M_1 M_2 \cdots M_{(d+1)n}$.

Let us denote $f^+(h_1, x) = h_2$, when $x \in M$. The complexity of the construction is at most $2.5 \cdot (d+1) \cdot n \cdot 2^{\frac{n}{2}}$.

In step (3) the attacker chooses 2^d new chaining hash values $b_1, b_2, \cdots, b_{2^d}$ and creates a multicollision diamond using the message set $M = M_1 M_2 \cdots M_{(d+1)n}$. So, with the complexity $8 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$, the attacker should be able to produce a message y_i for each chaining hash value b_i and a multicollision diamond with hash value h_3 such that $y_i \in M_{n+1} M_{n+2} \cdots M_{(d+1)n}$ and $f^+(b_i, y_i) = h_3$ for all $i \in \{1, 2, \cdots, 2^d\}$. The attacker now commits to a message length $|p| + 1 + d + n(d+1)$, where $|p|$ is the length of the prefix and to hash value a h_3, thus completing the offline phase.

Assume now that the attacker is challenged with a prefix p. In step (4) the attacker searches for a message block z_a such that $f^+(h_0, pz_a) = a_i$ for some $i \in \{1, 2, \cdots, 2^d\}$. The complexity of this is around 2^{n-d}. Now there is a message x_i such that $f^+(a_i, x_i) = h_1$. Next on step (5) the attacker searches for a message z_b such that $z_b \in M_1 M_2 \cdots M_n$ and $f^+(h_2, pz_a x_i z_b) = b_j$ for some $j \in \{1, 2, \cdots, 2^d\}$. The complexity of finding z_b is 2^{n-d}.

The attacker now chooses the message $pz_a x_i z_b y_j$. The length of the message is $|p| + 1 + d + n(d+1)$ while

$$f^+(f^+(h_0, pz_a x_i z_b y_j), pz_a x_i z_b y_j) = f^+(f^+(a_i, x_i z_b y_j), pz_a x_i z_b y_j)$$
$$= f^+(f^+(h_1, z_b y_j), pz_a x_i z_b y_j) = f^+(h_2, pz_a x_i z_b y_j) = f^+(b_j, y_j) = h_3.$$

The offline complexity of this process is $12 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}} + 2.5 \cdot (d+1)n \cdot 2^{\frac{n}{2}}$ with the algorithm memory requirement $0.83 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$. The online complexity is $3 \cdot 2^{n-d}$ and the online memory requirement approximately $2^{d+2} + 2(d+1)n$. This means that by choosing $d = \frac{n}{3}$ we get the total complexity of approximately $(12 \cdot \sqrt{\frac{n}{3}} + 3) \cdot 2^{\frac{2n}{3}}$.

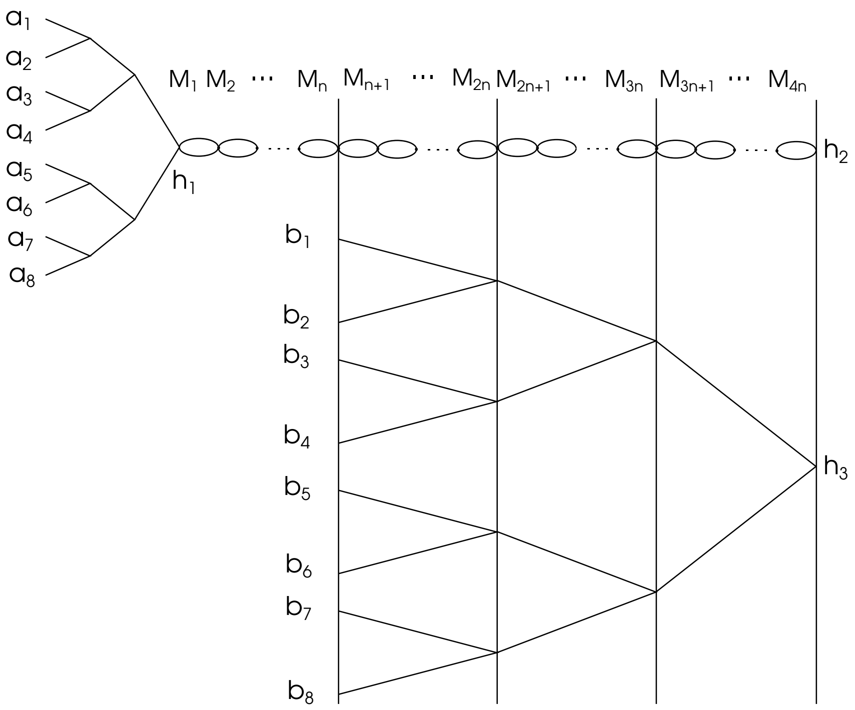


Fig 11. Hash Twice herding attack: Offline phase.

3.5.3 Herding Attack Variant against Zipper Hash

Let f_1, f_2 be compression functions: $\{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ and $h_0 \in \{0,1\}^n$ an initial value. Consider the Zipper Hash Z_{f_1,f_2,h_0} (see subsection 2.5.3). If we assume that f_2 is a random oracle, it is impossible to create a standard herding attack against Z_{f_1,f_2,h_0}. The attacker is not aware of the prefix and the first block of the prefix is the last one used in hashing. If f_2 is a random oracle and we are not aware of the message block used, it is clear that there is no way to predict the produced hash value, even if we could predict the hash value given to f_2.

However, as has been shown in [1] it is possible to mount a variant of herding attack against the Zipper Hash, even when f_1 and f_2 are random oracles. In this variant we assume that the attacker is challenged with a suffix s that is placed at the end of the message instead of a prefix. Otherwise the variant is similar to an original herding attack.

The attack proceeds through the following steps in the offline phase. The attacker creates

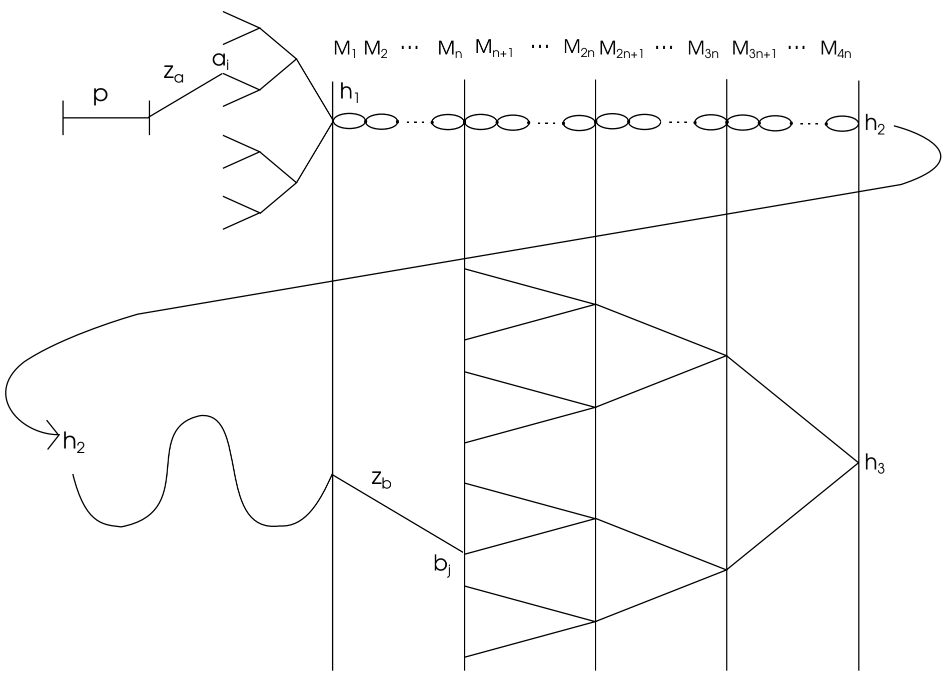


Fig 12. Hash Twice herding attack: Online phase.

(1) a large Joux type multicollision $M_1 M_2 \cdots M_{(d+1)n}$ based on the initial value h_0 and compression function f_1.

(2) a multicollision diamond K based on $M_{(d+1)n} M_{(d+1)n-1} \cdots M_1$ and the compression function f_2.

Then in the online phase the attacker is challenged with suffix s and she/he:

(3) connects the suffix s to the multicollision diamond K.

At the beginning of the attack the attacker creates a Joux's multicollision with size $2^{(d+1)n}$ for f_1^+ starting from initial value h_0. Now the attacker has message sets $M_1, M_2, ..., M_{(d+1)n}$ such that

(i) for each $i \in \{1,2,...,(d+1)n\}$ the set M_i consists of two distinct message blocks; and

(ii) for all $x, x' \in M = M_1 M_2 \cdots M_{(d+1)n}$, $f_1^+(h_0, x) = f_1^+(h_0, x')$.

Let us denote $f_1^+(h_0, x) = h_1$, when $x \in M$. The complexity of the construction is at most $2.5 \cdot (d+1)n \cdot 2^{\frac{n}{2}}$.

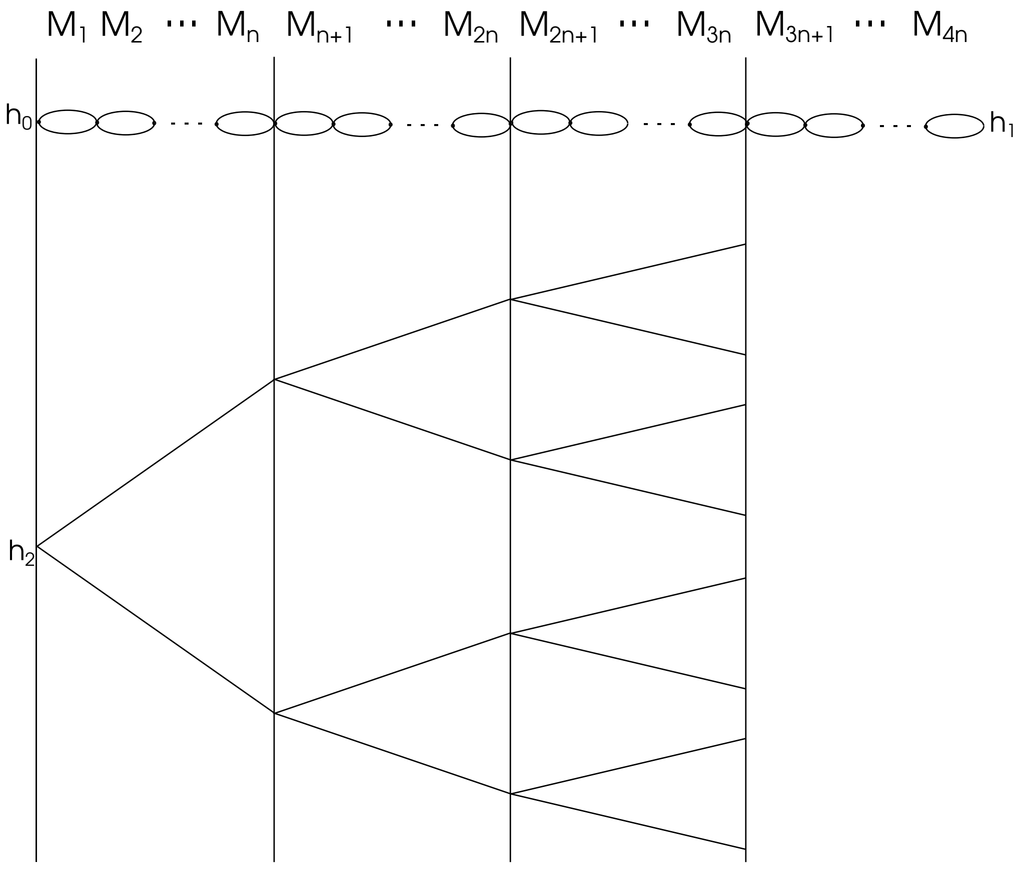
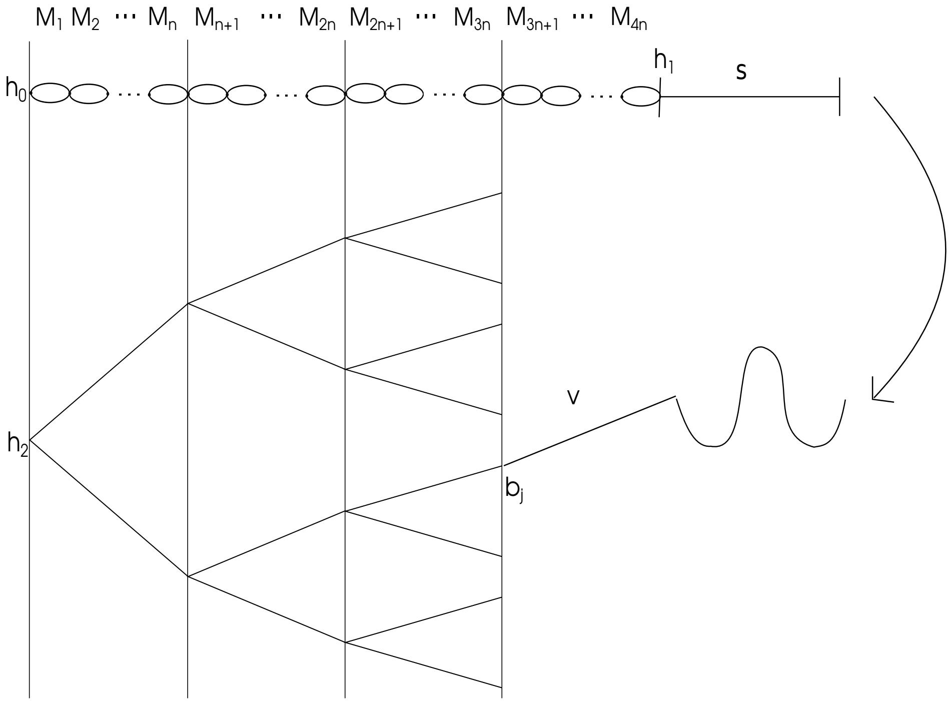


Fig 13. Herding attack variant against Zipper Hash: Offline phase.

Next the attacker chooses 2^d new chaining hash values $b_1, b_2, \cdots, b_{2^d}$ and creates a multicollision diamond using the message set $M_{(d+1)n}M_{(d+1)n-1}\cdots M_1$ for the compression function f_2. The complexity of this is approximately $8 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$ and the attacker now has a message z_i for each chaining hash value b_i and a single hash value h_2 such that $z_i \in M_{dn}M_{dn-1}\cdots M_1$ and $f_2^+(b_i, z_i) = h_2$ for all $i \in \{1, 2, \cdots, 2^d\}$.

The attacker is now ready to commit to a hash value h_2 and a message length $n(d+1) + |s|$, where $|s|$ is the length of the suffix. Assume now that the attacker is challenged with a suffix $s = s_1 s_2 \cdots s_p$, where $|s_i| = 1$ for all $i \in \{1, 2, \cdots, p\}$. The attacker searches for a message $v \in M_{(d+1)n}M_{(d+1)n-1}\cdots M_{dn+1}$ such that $f_2^+(f_1^+(h_1, s), s_p s_{p-1}\cdots s_1 v) = b_j$ for some $j \in \{1, 2, \cdots, 2^d\}$. The complexity of this action is around $2 \cdot 2^{n-d}$.



M_1 M_2 ⋯ M_n M_{n+1} ⋯ M_{2n} M_{2n+1} ⋯ M_{3n} M_{3n+1} ⋯ M_{4n}
h_0
h_1
s
v
h_2
b_j

Fig 14. Herding attack variant against Zipper Hash: Online phase.

Assume that $v = v_1 v_2 \cdots v_n$ and $|v_j| = 1$ for all $j \in \{1, 2, \cdots n\}$, thus $v_n v_{n-1} \cdots v_1 \in M_{dn+1} M_{dn+2} \cdots M_{(d+1)n}$. Furthermore note that $f_2^+(b_j, z_j) = h_2$ and $z_j = z'_1 z'_2 \cdots z'_{nd}$, where $|z'_i| = 1$ for all $i \in \{1, 2, \cdots, nd\}$ and $z_j \in M_{dn} M_{dn-1} \cdots M_1$. It follows that $z'_{nd} z'_{nd-1} \cdots z'_1 \in M_1 M_2 \cdots M_{dn}$.

Now we know that

$$Z_{f_1, f_2, h_0}(z'_{nd} z'_{nd-1} \cdots z'_1 v_n v_{n-1} \cdots v_1 s)$$

$$= f_2^+(f_1^+(h_0, z'_{nd} z'_{nd-1} \cdots z'_1 v_n v_{n-1} \cdots v_1 s), s_p s_{p-1} \cdots s_1 v z_j)$$

$$= f_2^+(f_1^+(h_1, s), s_p s_{p-1} \cdots s_1 v z_j) = f_2^+(b_j, z_j) = h_2.$$

The offline complexity of this attack is $2.5 \cdot (d+1) \cdot n \cdot 2^{\frac{n}{2}} + 8 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$ while the online complexity is $2 \cdot 2^{n-d}$. The algorithm memory requirement of the attack is $0.83 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$. The online memory requirement of the attack is $2^d + 2 \cdot (d+1) \cdot n$. Once again by choosing $d = \frac{n}{3}$ the total complexity is approximately $(8 \cdot \sqrt{\frac{n}{3}} + 2) \cdot 2^{\frac{2n}{3}}$.

3.6 Second Preimage Attack with Diamond Structure

It is also possible to use the diamond structure for a second preimage attack against an iterated hash function, if the original message is long [2]. Assume that we have an iterated hash function f^+ with the initial value h_0 and the original message $y = y_1 y_2 \cdots y_{2^l}$ where y_i is a single message block for each $i \in \{1, 2, \cdots, 2^l\}$.

First the attacker creates a diamond structure with 2^d chaining hash values $a_1, a_2, \cdots, a_{2^d}$. Let us assume that the diamond value of the structure is h'. The attacker then searches for a message block x_v such that $f(h', x_v) = f^+(h_0, y_1 y_2 \cdots y_i)$ for some $i \in \{d+2, d+3, \cdots, 2^l\}$. The complexity of finding such an x_v is around 2^{n-l} as we have seen.

Next the attacker searches for a beginning part x_c such that $f^+(h_0, x_c) = a_j$ for some $j \in \{1, 2, \cdots, 2^d\}$ and $|x_c| = i - d - 1$. The complexity of finding such x_c is 2^{n-d}. The second preimage message is $x_c x_j x_v y_{i+1} y_{i+2} \cdots y_{2^l}$ where x_j is the path from a_j to h'.

The complexity of creating a diamond structure is $4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$ so the total complexity of this second preimage attack is $4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}} + 2^{n-l} + 2^{n-d}$ with the algorithm memory requirement of $0.83 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$ and the online memory requirement of $3 \cdot 2^d$. By choosing $d = \frac{n}{3}$ the attacker gets the total complexity of $(4 \cdot \sqrt{\frac{n}{3}} + 1) \cdot 2^{\frac{2n}{3}} + 2^{n-l}$.

The second preimage attack using expandable messages was discovered before the second preimage attack with the diamond structure and the complexity of the attack applying an expandable message is slightly smaller. However, some applications of an iterated hash function that are not vulnerable to second preimage attack applying an expandable message, such as Dithered hash function, are vulnerable to the second preimage attack that uses a diamond structure [1, 2]

3.6.1 Second Preimage Attack against Hash Twice

A second preimage attack that uses expandable messages does clearly not work against Hash Twice directly. However, it is possible to combine expandable messages, Joux's multicollisions and the multicollision diamond to create a second preimage attack against Hash Twice as was done in [1].

The attack proceeds through the following steps:

(1) Create an expandable message with an initial value h_0 and an expanding value h_1.
(2) Create a large Joux type multicollision $M_1 M_2 \cdots M_{(d+1)n}$ based on h_1.

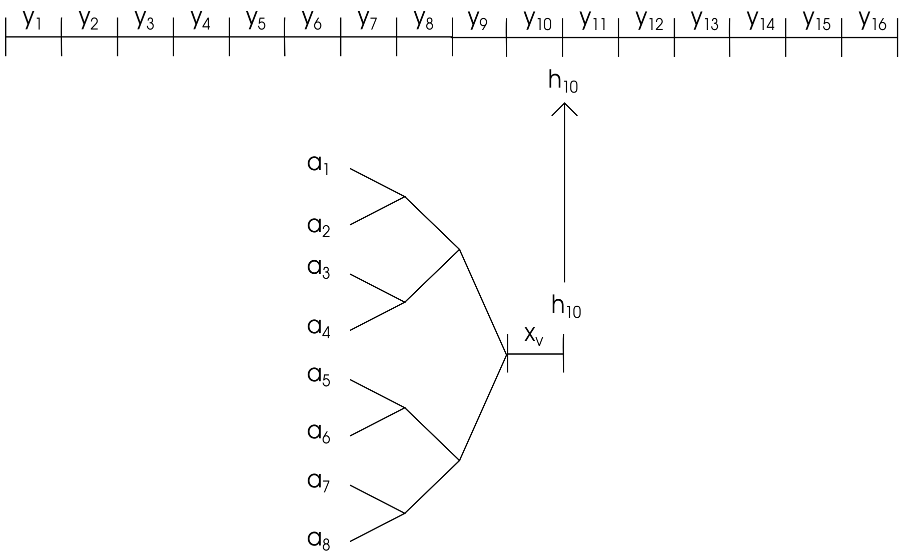


Fig 15. Second preimage attack example with a diamond structure: d=3 l=4.

(3) Create a multicollision diamond K based on $M_1M_2\cdots M_{(d+1)n}$ and a compression function f_2.

(4) Connect the multicollision diamond K to the original message on the “second round” of the hashing process.

(5) Use the expandable message to ensure that the created second preimage has the same length as the original message.

We will now take a more detailed look at the process.

Assume that we have the Hash Twice function T_{f,h_0} and an original message $y = y_1y_2\cdots y_{2^l}$, where y_i is a single message block for each $i \in \{1,2,\cdots,2^l\}$. The attacker creates an expandable message for f^+, with a maximum length $2^l + l - 1$ starting from the initial value h_0. Assume that the expanding value of the expandable message is h_1.

Next the attacker creates a Joux’s $2^{(d+1)n}-$collision for f^+ starting from the initial value h_1. As before this means that the attacker creates message sets $M_1, M_2, ..., M_{(d+1)n}$ such that:

(i) For each $i \in \{1,2,...,(d+1)n\}$, the set M_i consists of two distinct message blocks; and

(ii) $f^+(h_1,x) = f^+(h_1,x')$ for all $x,x' \in M = M_1M_2\cdots M_{(d+1)n}$.

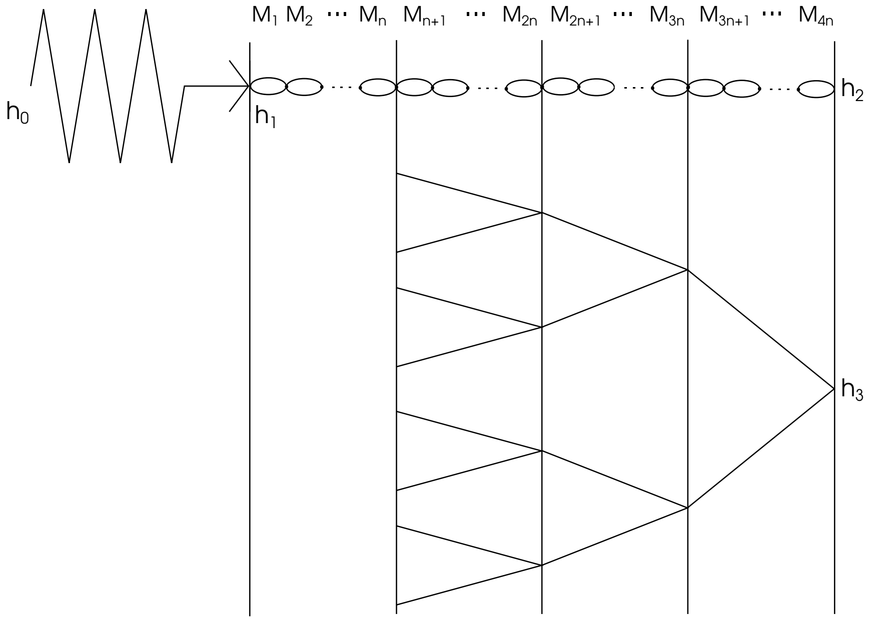


Fig 16. Second preimage attack against Hash Twice: Steps (1), (2) and (3).

Once again we can denote $f^+(h_1,x) = h_2$, when $x \in M$ and conclude that complexity of the construction is at most $2.5 \cdot (d+1) \cdot n \cdot 2^{\frac{n}{2}}$.

Now the attacker chooses 2^d new chaining hash values $b_1, b_2, \cdots, b_{2^d}$ and creates a multicollision diamond, using the message set $M = M_1 M_2 \cdots M_{(d+1)n}$. The attacker should be able to produce a single hash value h_3 and a message z_i for each chaining hash value b_i such that $z_i \in M_{n+1} M_{n+2} \cdots M_{(d+1)n}$ and $f^+(b_i, z_i) = h_3$. The complexity of this is $8 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$.

Next the attacker finds a message block x_v such that $f(h_3, x_v) = f^+(f^+(h_0, y_1 y_2 \cdots y_{2^l}), y_1 y_2 \cdots y_j)$ for some $j \in \{l+(d+1)n+1, l+(d+1)n+2, \cdots, 2^l\}$. Assume now that x_r is the expandable message with length $j-(d+1)n-1$. Now there should be $x_b \in M_1 M_2 \cdots M_n$ such that.

$$f^+(h_2, x_v y_{j+1} y_{j+2} \cdots y_{2^l} x_r x_b) = b_i$$

for some $i \in \{1, 2, \cdots, 2^d\}$. Such an x_b can be found with complexity $2 \cdot 2^{n-d}$.

Note that $f^+(b_i, z_i) = h_3$. Now since

$$T_{f,h_0}(x_r x_b z_i x_v y_{j+1} y_{j+2} \cdots y_{2^l})$$

$$= f^+(f^+(h_0, x_r x_b z_i x_v y_{j+1} y_{j+2} \cdots y_{2^l}), x_r x_b z_i x_v y_{j+1} y_{j+2} \cdots y_{2^l})$$
$$= f^+(f^+(h_1, x_b z_i x_v y_{j+1} y_{j+2} \cdots y_{2^l}), x_r x_b z_i x_v y_{j+1} y_{j+2} \cdots y_{2^l})$$
$$= f^+(f^+(h_2, x_v y_{j+1} y_{j+2} \cdots y_{2^l}), x_r x_b z_i x_v y_{j+1} y_{j+2} \cdots y_{2^l})$$
$$= f^+(b_i, z_i x_v y_{j+1} y_{j+2} \cdots y_{2^l}) = f^+(h_3, x_v y_{j+1} y_{j+2} \cdots y_{2^l})$$
$$= f^+(f(h_3, x_v), y_{j+1} y_{j+2} \cdots y_{2^l}) = T_{f,h_0}(y_1 y_2 \cdots y_{2^l})$$

we have $T_{f,h_0}(x_r x_b z_i x_v y_{j+1} y_{j+2} \cdots y_{2^l}) = T_{f,h_0}(y_1 y_2 \cdots y_{2^l})$. In addition $|x_r x_b z_i x_v y_{j+1} y_{j+2} \cdots y_{2^l}| = 2^l$ so $x_r x_b z_i x_v y_{j+1} y_{j+2} \cdots y_{2^l}$ is the second preimage for $y_1 y_2 \cdots y_{2^l}$.

The total complexity of this procedure is approximately $2.5 \cdot n(d+1) \cdot 2^{\frac{n}{2}} + 8 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}} + 2 \cdot 2^{n-d} + 2^{n-l}$ with the algorithm memory requirement $0.83 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$. This means that by choosing $d = \frac{n}{3}$ we get the total complexity of approximately $(8 \cdot \sqrt{\frac{n}{3}} + 2) \cdot 2^{\frac{2n}{3}} + 2^{n-l}$.

Remark 10. In [1] this attack is divided into the offline and online phase. Since in this work the other second preimage attacks are presented in a single phase, also this attack is presented in this form.

3.7 Trojan Message Attack

The article [1] offered two variants of the Trojan message attacks against the Merkle-Damgård structure: the Collision Trojan Attack and the Herding Trojan Attack. Both of them consist of the following three general phases.

1. The attacker creates a Trojan message t. The complexity of this phase is the offline complexity of the attack.
2. The attacker is challenged with a prefix message p from the prefix set P, where $|P| = 2^r$.
3. The attacker creates a second preimage for pt. The complexity of this phase is the online complexity of the attack.

We assume that the attacker is familiar with the compression function f, the initial hash value h_0 and the prefix set P. Furthermore assume from now on that $h_0 \in \{0,1\}^n$ is the initial hash value and $P = \{p_1, p_2, \cdots, p_{2^r}\}$, $r \in \mathbb{N}_+$ is the set of prefixes. Moreover, denote $h_{0,i} := f^+(h_0, p_i)$ for $i = 1, 2, \cdots, 2^r$. For the sake of simplicity we will assume that all the prefixes in P are of equal length $k \in \mathbb{N}_+$.

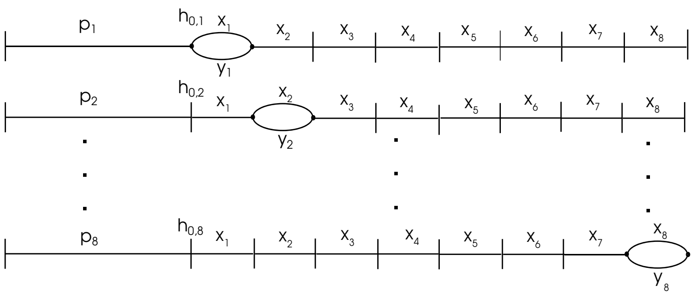


Fig 17. Example of the offline phase in Collision Trojan Attack when r=3.

Next we will present the basic structure of both the Collision Trojan Attack and the Herding Trojan Attack.

3.7.1 Collision Trojan Attack

The first phase of the Collision Trojan Attack consists of 2^r steps. In the first step the attacker creates a message block pair x_1, y_1 such that $f(h_{0,1}, x_1) = f(h_{0,1}, y_1)$, $x_1 \neq y_1$. In the step i of the attack, where $i \in \{2, 3, \cdots, 2^r\}$, the attacker computes the value $h_{i-1} = f^+(h_{0,i}, x_1 x_2 \cdots x_{i-1})$ and creates a message block pair x_i, y_i such that $f(h_{i-1}, x_i) = f(h_{i-1}, y_i)$ and $x_i \neq y_i$. The attacker chooses then the word $t = x_1 x_2 \cdots x_{2^r}$ for the Trojan message and has thus completed the offline phase of the attack.

Assume now that in the second phase the attacker is challenged with p_j and forms the word $p_j t$.

In the third phase the attacker first sets $t' := x_1 x_2 \cdots x_{j-1} y_j x_{j+1} \cdots x_{2^r}$ and then offers the word $p_j t'$ for a second preimage to $p_j t$. The attack is successful, since obviously $f^+(h_0, p_j t) = f^+(h_0, p_j t')$ and $|p_j t| = |p_j t'|$. The offline complexity of this attack is $2.5 \cdot 2^{\frac{n}{2}+r}$ with the algorithm memory requirement $2^{\frac{n}{2}}$. The online complexity is negligible. The online memory requirement of this attack is 2^{r+1}.

3.7.2 Herding Trojan Attack

The article [1] also presents a stronger version of the Trojan message attack called the Herding Trojan Attack. It offers the attacker a greater freedom to choose the contents of the second preimage message. To ensure this we will assume that the attacker is, in addition to the prefix choice p_j from the set P, challenged with any prefix w (for the sake of simplicity we assume that $|w| < k$). The attacker now has to find a suffix s such that $f^+(h_0, ws) = f^+(h_0, pt)$, where t is the Trojan message.

In the first phase the attacker creates a diamond structure with 2^d chaining hash values. The complexity of this is $4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$. Assume now that the final value of the structure is h'. Then the attacker creates a message x_0 such that $|x_0| = d$. Next the attacker searches for a message block pair x_1, y_1 such that $f^+(h_{0,1}, x_0 x_1) = f(h', y_1)$, and then sets $h_1 = f(h', y_1)$. Certainly the attacker can perform this with complexity $\frac{e}{e-1} \cdot 2^{\frac{n}{2}+1}$.

In the step i of the first phase, where $i \in \{2, 3, \cdots, 2^r\}$, the attacker computes the value $h_{i-1,i} := f^+(h_{0,i}, x_0 x_1 \cdots x_{i-1})$ and creates a message block pair x_i, y_i such that $f(h_{i-1,i}, x_i) = f(h_{i-1}, y_i)$. Then the attacker simply sets $h_i := f(h_{i-1}, y_i)$ and is ready to proceed to next step. Finally the attacker creates the Trojan message $t = x_0 x_1 \cdots x_{2^r}$ and has finished the second phase.

Let us assume that the attacker is challenged both with a prefix p_j, $j \in \{1, 2, \ldots, 2^r\}$ and a second prefix w such that $|w| < k$. The attacker now searches for a connection message z such that $|wz| = k$ and $f^+(h_0, wz)$ is equal to some chaining hash value of the created diamond structure. Assume now that message u is the path from this chaining hash value to the root hash value h' of the diamond structure, i.e. $f^+(h_0, wzu) = h'$.

Now we have $f^+(h_0, wzuy_1 y_2 \cdots y_j) = h_j = f^+(h_0, p_j x_0 x_1 \cdots x_j)$, so clearly $wzuy_1 y_2 \cdots y_j x_{j+1} x_{j+2} \cdots x_{2^r}$ is the second preimage for the word $p_j x_0 x_1 \cdots x_{2^r}$.

The complexity of the offline phase is $\frac{e}{e-1} \cdot 2^{\frac{n}{2}+r+1} + 4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$ while the complexity of finding z is 2^{n-d}, which means that the complexity of the online phase is also 2^{n-d}. The algorithm memory requirement of this attack is $0{,}83 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$ while the online memory requirement is $3 \cdot 2^d + 2^{r+1}$. Once again, we can set $d = \frac{n}{3}$ and get the total complexity of approximately $\frac{e}{e-1} \cdot 2^{\frac{n}{2}+r+1} + (4 \cdot \sqrt{\frac{n}{3}} + 1) \cdot 2^{\frac{2n}{3}}$.

It is easy to see that the complexity of this kind of an attack is approximately $(4 \cdot \sqrt{\frac{n}{3}} + 1) \cdot 2^{\frac{2n}{3}}$ as long as the number of possible prefixes is at most $2^{\frac{n}{6}}$, while the length of the created message is $k + d + 2^r$. If the number of possible preimages is larger than $2^{\frac{n}{6}}$, the complexity exceeds $(4 \cdot \sqrt{\frac{n}{3}} + 1) \cdot 2^{\frac{2n}{3}}$.

In comparison the second preimage attack presented in [18] and the second preimage attack based on the diamond structure presented in [2] against a message with length $2^{\frac{n}{6}}$ would have the complexity of approximately $2^{\frac{5n}{6}}$.

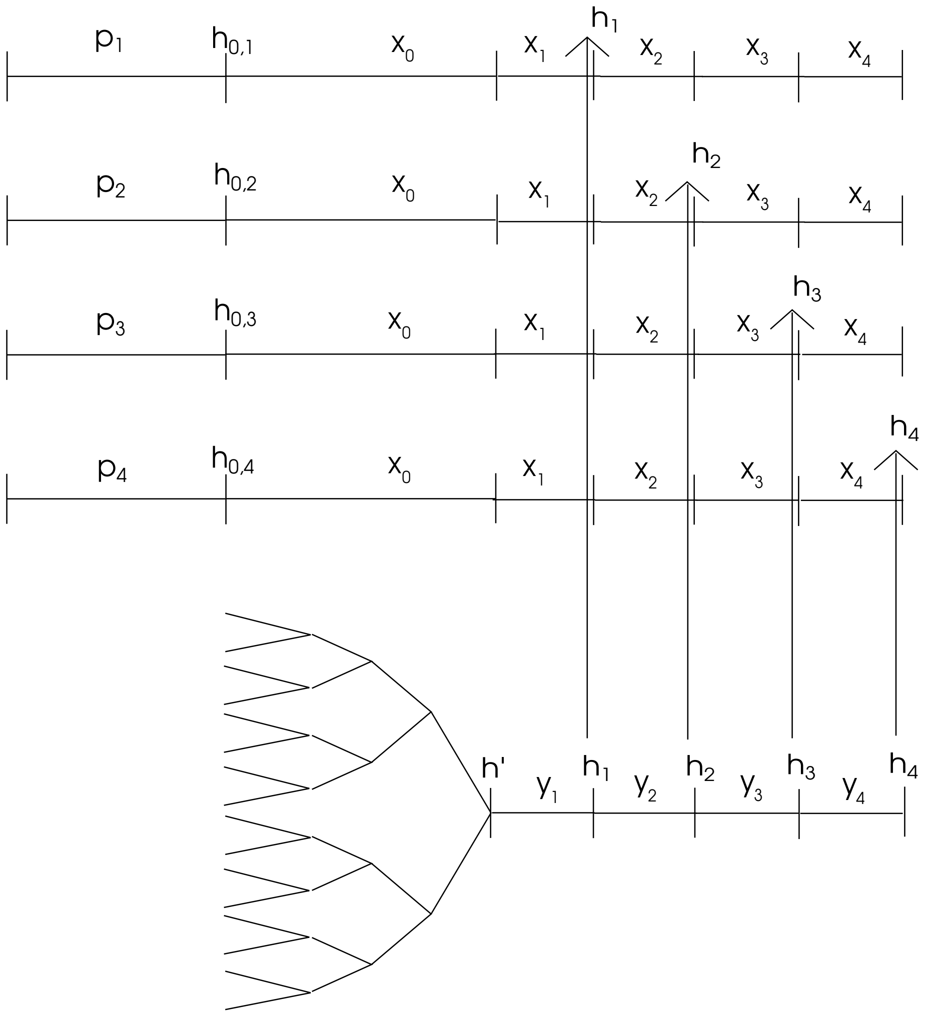


Fig 18. Example of the offline phase in Herding Trojan Attack when r=2, d=4.

3.8 Tables

The most important results of this chapter are summarized on the following tables. The complexity refers to the complexity of the attack carried out in the manner described in this work. AMR refers to the algorithm memory requirement and OMR refers to the online memory requirement of the attack carried out in the manner described in this work.

We wish to point out that the given complexities are rough upper bounds and not strict. The complexities could very well be presented in the asymptotic setting, where we would drop the constant multipliers. The aim of this work has not been to optimize these constants. We have decided to include them in order to be able to give certain upper bounds for the complexity of the attack, when the attacker can satisfy the algorithm memory requirement.

Table 1. Complexities and memory requirements of basic attacks and constructions.

Attack Type	Complexity	AMR
Joux's Multicollision Attack [15]	$2.5 \cdot k \cdot 2^{\frac{n}{2}}$	$2^{\frac{n}{2}}$
Second Preimage Attack [18]	$\frac{e}{e-1} \cdot l \cdot 2^{\frac{n}{2}+1} + 2^{n-l}$	$2^{\frac{n}{2}+1}$
Creating Diamond Structure [5, 17]	$4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$	$0.83 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$
Hash Twice Second Preimage [1, 5]	$(8 \cdot \sqrt{\frac{n}{3}} + 2) \cdot 2^{\frac{2n}{3}} + 2^{n-l}$	$0.83 \cdot \sqrt{\frac{n}{3}} \cdot 2^{\frac{2n}{3}}$

The length of the original message in second preimage attacks is assumed to be 2^l.

Table 2. Offline and online complexities of different attacks.

Attack Type	Offline	Online
Herding Attack [5, 17]	$4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$	2^{n-d}
Hash Twice Herding [1, 5]	$12 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}} + 2.5 \cdot (d+1) \cdot n \cdot 2^{\frac{n}{2}}$	$3 \cdot 2^{n-d}$
Zipper Hash Herding Variant [1, 5]	$8 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}} + 2.5 \cdot (d+1) \cdot n \cdot 2^{\frac{n}{2}}$	$2 \cdot 2^{n-d}$
Collision Trojan Attack [1]	$2.5 \cdot 2^{\frac{n}{2}+r}$	Negl.
Herding Trojan Attack [1]	$\frac{e}{e-1} \cdot 2^{\frac{n}{2}+r+1} + 4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$	2^{n-d}

Negl. means that the online complexity is negligible. The size of the prefix set in Trojan message attack is assumed to be 2^r.

Table 3. Memory requirements of different attacks.

Attack Type	AMR	OMR
Herding Attack [5, 17]	$0.83 \cdot \sqrt{d} \cdot 2^{\frac{d+n}{2}}$	$3 \cdot 2^d$
Hash Twice Herding [1, 5]	$0.83 \cdot \sqrt{d} \cdot 2^{\frac{d+n}{2}}$	$2^{d+2} + 2 \cdot (d+1) \cdot n$
Zipper Hash Herding Variant [1, 5]	$0.83 \cdot \sqrt{d} \cdot 2^{\frac{d+n}{2}}$	$2^d + 2 \cdot (d+1) \cdot n$
Collision Trojan Attack [1]	$2^{\frac{n}{2}}$	2^{r+1}
Herding Trojan Attack [1]	$0.83 \cdot \sqrt{d} \cdot 2^{\frac{d+n}{2}}$	$3 \cdot 2^d + 2^{r+1}$

Table 4. Complexities of the attacks that apply diamond structure when *d=n/3*.

Attack Type	Complexity
Herding Attack [5, 17]	$(4 \cdot \sqrt{\frac{n}{3}} + 1) \cdot 2^{\frac{2n}{3}}$
Hash Twice Herding [1, 5]	$(12 \cdot \sqrt{\frac{n}{3}} + 3) \cdot 2^{\frac{2n}{3}}$
Zipper Hash Herding Variant [1, 5]	$(8 \cdot \sqrt{\frac{n}{3}} + 2) \cdot 2^{\frac{2n}{3}}$
Herding Trojan Attack [1]	$\frac{e}{e-1} \cdot 2^{\frac{n}{2}+r+1} + (4 \cdot \sqrt{\frac{n}{3}} + 1) \cdot 2^{\frac{2n}{3}}$

4 A Variant of Joux's Attack

Next we present a variant of Joux's multicollision attack (see [15]). This attack can be found in [23]. The calculations and results of [23] are kept mostly in their original form. Based on the results of subsection 2.7.2 we could slightly improve the efficiency of this variant. We will look at the details of this in Section 4.6. This variant is a generalization of the Joux's method and uses the information about messages and hash values gathered in previous steps of the attack.

The basic idea of our attack is to create smaller sets of message blocks and instead of searching for the collision only in the current set, we also compare the hash values of the current set with s previous ones. This gives us greater probability of finding collisions with less work. The downside is, that we need more memory space to store the message blocks and hash values.

4.1 About Probabilites

Let $s \in \mathbb{N}_+$. Suppose that we have enough memory to store the message blocks and the respective hash values produced during the previous s steps of the attack. Suppose furthermore that in each step approx. $\frac{1}{\sqrt{2s}} 2^{\frac{n}{2}}$ random message blocks are generated and their respective hash values computed.

Assume that we are in the $(k+1)$st step of the attack ($k \in \mathbb{N}$, $k \geq s$). Let M_j be the set of pairs of message blocks and respective hash values computed in the step $j = k-s+1, k-s+2, \ldots, k$. Next we generate $\frac{1}{\sqrt{2s}} 2^{\frac{n}{2}}$ new random message blocks and hash values. According to [8] the probability $p_1(s)$ of finding a collision within this new set through the birthday paradox is approximately $1 - e^{-\frac{1}{4s}}$.

Let H_i be set of hash values in step i, where $i = k-s+1, k-s+2, \ldots, k$. Let us now evaluate the probability $p_2(s)$ that $H_{k+1} \cap H_i \neq \emptyset$ for at least one $i \in \{k-s+1, k-s+2, \ldots, k\}$. Since $|\bigcup_{i=k-s+1}^{k} H_i| \approx \sqrt{\frac{s}{2}} \cdot 2^{\frac{n}{2}}$ we can now use Lemma 2 to prove that the probability of finding a collision is approximately

$$p_2(s) \approx 1 - e^{-\frac{1}{2}} \approx 0.4$$

when we assume n to be large enough.

Suppose now that $p_3(s)$ is the probability that either there is a collision between hash values calculated in the $(k+1)$ st step or that there exists a common hash value in the sets H_{k+1} and H_i for some $i \in \{k-s+1, k-s+2, \dots, k\}$. Since the events above can be assumed to be statistically independent, the equality $p_3(s) = p_1(s) + p_2(s) - p_1(s)p_2(s)$ holds. This means that

$$p_3(s) \approx (1 - e^{-\frac{1}{4s}}) + (1 - e^{-\frac{1}{2}}) - (1 - e^{-\frac{1}{4s}})(1 - e^{-\frac{1}{2}}).$$

When we let s grow, $p_3(s)$ approaches from above the number $1 - e^{-\frac{1}{2}} \approx 0.4$, the collision probability in the Joux's attack performed in the manner presented in this work. What follows now are the technical description and details of the attack. As stated before the results can be found in [23].

4.2 Probabilistic Attack Algorithm

Let an iterated hash function $f^+ : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$, an initial value $h_0 \in \{0,1\}^n$, and an integer $s \in \mathbb{N}_+$ be given. Denote $d = \frac{1}{\sqrt{2s}} 2^{\frac{n}{2}}$.

Initialization. Let $h := h_0$, $i := 1$, and $C_0 := \{\varepsilon\}$. While $i < s+1$ do the following.

Generate d random message blocks $x_1, x_2, \dots, x_d$. Compute the respective hash values $f(h, x_j)$ for $j = 1, 2, \dots, d$. Let $(x_{i,j}, h_{i,j}) := (x_j, f(h, x_j))$ for $j = 1, 2, \dots, d$, $M_i := \{(x_{i,j}, h_{i,j}) | j = 1, 2, \dots, d\}$ and $H_i = \{h_{i,j} | j = 1, 2, \dots, d\}$. Search for a collision in $h_{i,1}, h_{i,2} \dots, h_{i,d}$.

A. Suppose that a collision is found. Let $j_1, j_2 \in \{1, 2, \dots, d\}$, $j_1 \neq j_2$ be such that $h_{i,j_1} = h_{i,j_2}$. Then set $D := \{x_{i,j_1}, x_{i,j_2}\}$, $h := h_{i,j_1}$, and $C_i := C_{i-1}D$. Finally set $i := i+1$.

B. Suppose that no collision is found in $h_{i,1}, h_{i,2} \dots, h_{i,d}$. If $i = 1$, set $h := h_{i,1}$, $C_1 := C_0\{x_{i,1}\}$, and $i := i+1$. If $i > 1$, search for a collision between the values in the set H_i and the values in the union $\cup_{j=1}^{i-1} H_j$.

   1° Assume that a collision is found. Let $i_1 \in \{1, 2, \dots, i-1\}$ be the largest number such that there exist $l_1, l_2 \in \{1, 2, \dots, d\}$, for which $h_{i,l_1} = h_{i_1,l_2}$. Set $C_i := C_{i-1}\{x_{i,l_1}\} \cup C_{i_1-1}\{x_{i_1,l_2}\}$, and $i := i+1$.

   2° Assume that no collision between the values in the set H_i and the values in the union $\cup_{j=1}^{i-1} H_j$ is found. Then set $C_i := C_{i-1}\{x_{i_1}\}$ and $i := i+1$.

We have now completed the initialization and are ready to describe the real attack. The procedure is exactly the same as before, except that if no collision is found, we shall repeat the generation of the set of random message blocks (and the execution of the step) until a collision is found.

Assumptions for the general step. Let $k \in \mathbb{N}_+$, $k \geq s$. Suppose that the sets $M_{k-i} := \{(x_{k-i,j}, h_{k-i,j}) | j = 1,2,\ldots,d\}$ and $H_{k-i} = \{h_{k-i,j} | j = 1,2,\ldots,d\}$ for $i = 0,1,\ldots,s-1$ are created as well as the collision sets $C_{k-s}, C_{k-s+1}, \ldots, C_k$. Assume that a hash value h is given.

The general (k+1)st step. Generate d random message blocks $x_1, x_2, \ldots,\ x_d$. Compute the respective hash values $f(h, x_i)$ for $i = 1,2,\ldots,d$. Set

$$(x_{k+1,j}, h_{k+1,j}) := (x_j, f(h, x_j))$$

for $j = 1,2,\ldots,d$, $M_{k+1} := \{(x_{k+1,j}, h_{k+1,j}) | j = 1,2,\ldots,d\}$ and $H_{k+1} = \{h_{k+1,j} |\ j = 1,2,\ldots,d\}$.

Search for a collision in $h_{k+1,1},\ h_{k+1,2},\ldots,h_{k+1,d}$.

C. Suppose that a collision is found. Let $j_1, j_2 \in \{1,2,\ldots,d\}$, $j_1 \neq j_2$ be such that $h_{i,j_1} = h_{i,j_2}$. Then set $D := \{x_{k+1,j_1}, x_{k+1,j_2}\}$ and $h := h_{k+1,j_1}$. Finally set $C_{k+1} := C_k D$.

D. Suppose that no collision is found in $h_{k+1,1}, h_{k+1,2} \ldots, h_{k+1,d}$. Search for a collision between the values in the set H_{k+1} and the values in the union $\cup_{j=k-s+1}^{k} H_j$.

   3° Assume that a collision is found. Let $i_1 \in \{k-s+1, k-s+2, \ldots, k\}$ be the largest number such that there exist $l_1, l_2 \in \{1,2,\ldots,d\}$, for which $h_{k+1,l_1} = h_{i_1,l_2}$. Set $C_{k+1} := C_k\{x_{k+1,l_1}\} \cup C_{i_1-1}\{x_{i_1,l_2}\}$.

   4° Assume that no collision between H_{k+1} and $\cup_{j=k-s+1}^{k} H_j$ is found. Then repeat the execution of the $(k+1)$ st step.

Let us now assume that the $(k+1)$ st step is carried out successfully and we have found a match. It is time to look at the size of the created multicollision. If there is a match between hash values $h_{k+1,1},\ h_{k+1,2},\ldots,h_{k+1,d}$, say $h_{k+1,i} = h_{k+1,j}$, $i \neq j$, then we have just doubled the size of our multicollision, because the last block of the colliding messages can be chosen to be $\{x_{k+1,i}, x_{k+1,j}\}$. This means that the value $|C_{k+1}| = 2\,|C_k|$. If this is not the case, then $h_{k+1,l_1} = h_{i_1,l_2}$ for some $i_1 \in \{k-s+1, k-s+2, \ldots, k\}$ and $l_1, l_2 \in \{1,2,\ldots,d\}$, $l_1 \neq l_2$. This means that

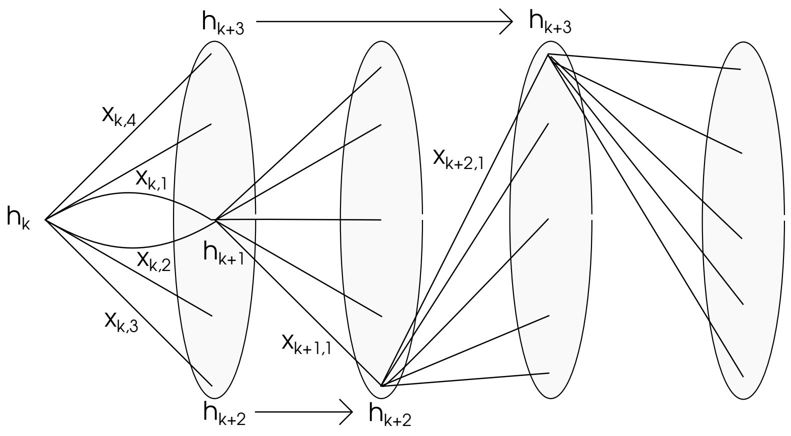


Fig 19. Example of attack variant. Starting from h_k we can choose paths $x_{k,4}$, $x_{k,3}x_{k+2,1}$, $x_{k,1}x_{k+1,1}x_{k+2,1}$ and $x_{k,2}x_{k+1,1}x_{k+2,1}$ to reach hash value h_{k+3}.

almost certainly the equality $|C_{k+1}| = |C_k| + |C_{i_1-1}|$ holds (theoretically we could have $|C_{k+1}| = |C_k| + |C_{i_1}|$ which would of course be a good thing since $|C_{i_1}| > |C_{i_1-1}|$).

Obviously all the messages in our collision set C_{k+1} are not of equal length, which seems to be a problem at first. We shall address this problem later.

4.3 Computing the Expected Value

Let us now evaluate the size of the created multicollision in step $k+1$, $k \geq s$. If there is a match in $h_{k+1,1}, h_{k+1,2}, \ldots, h_{k+1,d}$, the size of the created collision is $|C_{k+1}| = 2|C_k|$ and the probability of this event is $\frac{p_1(s)}{p_3(s)}$. Otherwise the collision is between sets H_{k+1} and some set H_i, where $i \in \{k-s+1, k-s+2, \cdots, k\}$, which means that the size of the created collision is at least $|C_{k+1}| = |C_k| + |C_{i_1-1}|$. The probability that such a collision exists and that there is no match in $h_{k+1,1}, h_{k+1,2}, \ldots, h_{k+1,d}$ is $\frac{p_3(s)-p_1(s)}{p_3(s)}$. We may assume that each of the sets $H_{k-s+1}, H_{k-s+2}, \ldots, H_k$ contains the matching hash value with equal likelihood; this probability is $\frac{p_3(s)-p_1(s)}{sp_3(s)}$, since the number of sets is s.

If we now mark the expected size of the multicollision in step i with E_i, we get the equations

$$E_{k+s+1} = \frac{p_1(s)}{p_3(s)}(2E_{k+s}) + \frac{p_3(s)-p_1(s)}{p_3(s)}(E_{k+s} + \frac{1}{s}(E_{k+s-1} + E_{k+s-2} + \cdots + E_k))$$

$$= \left(1+\frac{p_1(s)}{p_3(s)}\right)(E_{k+s}) + \left(1-\frac{p_1(s)}{p_3(s)}\right)\frac{1}{s}(E_{k+s-1}+E_{k+s-2}+\cdots+E_k) \tag{1}$$

for each $k \in \mathbb{N}$. From the definitions of $p_1(s)$ and $p_3(s)$, it is easy to see that $E_{k+s+1} > E_{k+s} + \frac{1}{s}(E_{k+s-1}+E_{k+s-2}+\cdots+E_k)$ and when s is large $E_{k+s+1} \approx E_{k+s} + \frac{1}{s}(E_{k+s-1}+E_{k+s-2}+\cdots+E_k), k \in \mathbb{N}$.

Let us now evaluate the size of the multicollision in step $k+s+1$ by using the equation

$$E_{k+s+1} = E_{k+s} + \frac{1}{s}(E_{k+s-1}+E_{k+s-2}+\cdots+E_k)$$

when $k \in \mathbb{N}$. Moreover, we set $E_i = |C_i|$ for all $i = 1,2,\ldots,s$, where $|C_1|,|C_2|,\ldots,|C_s|$ are the cardinalities of the sets determined in the initialization step of our attack. The above recursive equation has the characteristic polynomial $f_s(x) = x^{s+1} - x^s - \frac{1}{s}(x^{s-1} + x^{s-2} + \cdots + 1)$. The roots of this polynomial certainly determine the values of E_k, where $k > s$. For the large values of k the root that has the largest absolute value, dominates the values of the sequence and gives us the ratio $\frac{E_{k+1}}{E_k}, k \in \mathbb{N}$.

Now we have to find the solution to the equation $x^{s+1} - x^s - \frac{1}{s}(x^{s-1} + x^{s-2} + \cdots + 1) = 0$ with the largest absolute value. It is easy to see that $x = 0$ is not a root for the equation and so it can be written in the form $1 = x^{-1} + \frac{1}{s}(x^{-2} + x^{-3} + \cdots + x^{-s-1})$. Clearly $x^{-1} + \frac{1}{s}(x^{-2} + x^{-3} + \cdots + x^{-s-1})$ is decreasing, when $x \in \mathbb{R}_+$ and so our equation can have only one positive real root. It is straight forward to see that $x \neq 1$ so the equation can be written as

$$x^s(x-1) + \frac{1-x^s}{s(x-1)} = \frac{x^s[s(1-x)^2-1]+1}{s(x-1)} = 0.$$

Finding the general solution to such an equation is hard if not an impossible task. However finding an approximation is relatively easy. Let us set $g(x) = x^s[s(1-x)^2 - 1] + 1$. Now $g(1+\frac{1}{\sqrt{s}}) = (1+\frac{1}{\sqrt{s}})^s[s(\frac{1}{\sqrt{s}})^2 - 1] + 1 = 1 > 0$. On the other hand $g(1+\frac{1}{\sqrt{s+2}}) = (1+\frac{1}{\sqrt{s+2}})^s[s(\frac{1}{\sqrt{s+2}})^2 - 1] + 1 = (1+\frac{1}{\sqrt{s+2}})^s\frac{(-2)}{s+2} + 1 = 1 - (1+\frac{1}{\sqrt{s+2}})^s\frac{1}{\frac{s}{2}+1}$. It is easy to compute this value for all $s = 2,3,\cdots,19$ and see that in these cases $g(1+\frac{1}{\sqrt{s+2}}) < 0$.

Now assume that $s \geq 20$. In this case we get

$$(1+\frac{1}{\sqrt{s+2}})^s = 1 + \frac{s}{(s+2)^{\frac{1}{2}}} + \frac{s(s-1)}{2(s+2)^1} + \frac{s(s-1)(s-2)}{6(s+2)^{\frac{3}{2}}} + \sum_{i=4}^{s}\binom{s}{i}\frac{1}{\sqrt{s+2}^i}.$$

Now $\frac{s(s-1)(s-2)}{6(s+2)^{\frac{3}{2}}} = \frac{s}{6} \cdot \frac{s-1}{s+2} \cdot \frac{s-2}{\sqrt{s+2}}$. If $s = 20$, $\frac{s-1}{s+2} \cdot \frac{s-2}{\sqrt{s+2}} > 3$ and since $\frac{s-1}{s+2}$ and $\frac{s-2}{\sqrt{s+2}}$ are clearly increasing when $s \geq 20$ we get $\frac{s(s-1)(s-2)}{6(s+2)^{\frac{3}{2}}} > \frac{s}{2}$. Thus $(1 + \frac{1}{\sqrt{s+2}})^s > \frac{s}{2} + 1$ which in turn means that $g(1 + \frac{1}{\sqrt{s+2}}) < 0$ also for values $s = 20, 21, \cdots$.

We have now proven that $g(x)$ possesses a positive real root $x \in]1 + \frac{1}{\sqrt{s+2}}, 1 + \frac{1}{\sqrt{s}}[$. This root appears to be the only positive real root of our equation. We mark this root with x_1. Next we will prove that all other roots of this equation will have smaller absolute values.

Let us now assume that $x \notin \mathbb{R}_+$. This means that $|x+1| < |x| + 1$ Our equation can also be written as $x^{s+1} = x^s + \frac{1}{s}(x^{s-1} + x^{s-2} + \cdots + 1)$. It follows that $|x^{s+1}| < |x^s| + \frac{1}{s}(|x^{s-1}| + |x^{s-2}| + \cdots |x| + 1)$. This in turn means that

$$|x|^{s+1} - |x|^s - \frac{1}{s}(|x|^{s-1} + |x|^{s-2} + \cdots + 1) < 0.$$

Since $|x| \in \mathbb{R}_+$ this leads to $|x| < x_1 = |x_1|$.

We have now proven that $\frac{E_{k+1}}{E_k} = x_1 \in]1 + \frac{1}{\sqrt{s+2}}, 1 + \frac{1}{\sqrt{s}}[$ for the large values of k. This means that by taking k general steps, where k is large, we can create a multicollision with an expected size x_1^k, where $x_1^k > (1 + \frac{1}{\sqrt{s+2}})^k$.

4.4 Comparing the Procedure to Joux's Attack

Calculating the exact complexity, i.e. the expected number of required compression function calls, to create a collision of certain size with our attack variant, seems to be nearly an impossible task. However, it is still possible to compare its effectiveness with standard Joux's attack, when we use Joux's attack as a statistical experiment described in section 3.1. We will do this by calculating the expected sizes of created multicollisions, when the number of compression function calls is assumed to be constant.

Let us now compare the complexity of our attack with the complexity of the Joux's attack. As we have stated before, with $(1 - e^{-\frac{1}{2}})^{-1} \cdot k \cdot 2^{\frac{n}{2}} \approx 2.5 \cdot k \cdot 2^{\frac{n}{2}}$ work, we should get k successful steps in the Joux's attack. Each step of the Joux's attack multiplies the size of the multicollision by two and thus k steps gives us a 2^k-collision.

With $(1 - e^{-\frac{1}{2}})^{-1} \cdot k \cdot 2^{\frac{n}{2}}$ compression function calls our variant should be able to complete the initialization phase and more than

$$\frac{((1 - e^{-\frac{1}{2}})^{-1} \cdot k - s \cdot \frac{1}{\sqrt{2s}}) \cdot 2^{\frac{n}{2}}}{(p_3(s))^{-1} \cdot \frac{1}{\sqrt{2s}} \cdot 2^{\frac{n}{2}}} > \sqrt{2s}\left(k - p_3(s)\sqrt{\frac{s}{2}}\right)$$

successful ordinary steps after this (where $s \cdot \frac{1}{\sqrt{2s}} \cdot 2^{\frac{n}{2}}$ is the number of compression function calls required to complete the initialization phase). Let us mark $t_s = p_3(s)\sqrt{\frac{s}{2}}$. In each general step of our attack, the size of the multicollision is multiplied by a constant greater than $(1 + \frac{1}{\sqrt{s+2}})$. Thus the expected size of the multicollision is greater than

$$(1+\frac{1}{\sqrt{s+2}})^{\sqrt{2s}(k-t_s)} = [(1+\frac{1}{\sqrt{s+2}})^{\sqrt{2s}}]^{k-t_s},$$

when we are not considering Merkle-Damgård strengthening.

When $s \to \infty$ we see that

$$\left(1+\frac{1}{\sqrt{s+2}}\right)^{\sqrt{2s}} = \left[\left(1+\frac{1}{\sqrt{s+2}}\right)^{\sqrt{s}}\right]^{\sqrt{2}} \to e^{\sqrt{2}} \approx 4.113.$$

So if we are creating large multicollisions (k is large) and using a large number of stored sets of message blocks (s is large), our attack creates a $(e^{\sqrt{2}})^{k-t_s}$-collision, when we assume that the length of the message is not a problem.

With the same amount of compression function calls, Joux's attack creates a 2^k-collision. This means that theoretically we can achieve multicollisions with less than half of the work of Joux's attack.

4.4.1 Bypassing Merkle-Damgård Strengthening

As stated before Merkle-Damgård strengthening means that the length of the message is added to the end of the original message before hashing. This forces all of the colliding messages in our method to be of the same length. In Joux's attack this is not a problem, because all the messages have the same length. At first this might seem problematic to our attack, since the lengths of the created messages are not equal. However, when the number of the steps taken is large, we can overcome this obstacle.

If we complete the initialization phase (with s steps) and after this $\sqrt{2s}(k-t_s)$ ordinary steps, where k is large, then there are certainly at most $\sqrt{2s}(k-t_s)+s$ possible lengths for the messages. This means that the expected size for the largest set of messages of the same length is at least

$$\frac{(1+\frac{1}{\sqrt{s+2}})^{\sqrt{2s}(k-t_s)}}{\sqrt{2s}(k-t_s)+s}.$$

In reality the largest collision set with the equal length messages is of course much greater. However, even this evaluation shows that, when k is large, the length of the messages is not really an obstacle, since $(1+\frac{1}{\sqrt{s+2}})^{\sqrt{2s}(k-t_s)}$ grows exponentially with respect to k and $\sqrt{2s}(k-t_s)+s$ grows only linearly with respect to k.

Certainly we know that the expected size of the created multicollision will be greater than $\frac{1}{\sqrt{2s}(k-t_s)+s}(e^{\sqrt{2}})^{k-t_s}$, when we do $(1-e^{-\frac{1}{2}})^{-1}\cdot k\cdot 2^{\frac{n}{2}}$ compression function calls and assume s and k to be large enough.

4.5 Special Cases with Small Parameter Values

In practice the amount of usable memory and the maximum length of the messages limit the use of our attack variant. However, even the small values of s give us quite nice results for large k. We can assume that the case $s=0$ is the standard attack by Joux's.

It is possible to use Maple program to evaluate the expected multicollision sizes with $2.5\cdot k\cdot 2^{\frac{n}{2}}$ compression function calls, when $s=1$ and come up with

$$\frac{1}{\sqrt{2}(k-t_1)+1}(1.74948)^{\sqrt{2}\cdot 2.5\cdot 0.53272\cdot(k-t_1)} \approx \frac{1}{\sqrt{2}(k-t_1)+1}(2.87)^{k-t_1}.$$

Similarly the cases $s=2$ are evaluated, where we get the expected size

$$\frac{1}{2(k-t_2)+2}(1.62322)^{2\cdot 2.5\cdot 0.47050\cdot(k-t_2)} \approx \frac{1}{2(k-t_2)+2}(3.13)^{k-t_2}$$

and in the case $s=3$, we get the expected size of the multicollision to be approximately

$$\frac{1}{\sqrt{6}(k-t_3)+3}(1.54478)^{\sqrt{6}\cdot 2.5\cdot 0.44797\cdot(k-t_3)} \approx \frac{1}{\sqrt{6}(k-t_3)+3}(3.30)^{k-t_3}.$$

4.6 Improved Results

As stated in subsection 2.7.2 we can improve the standard statistical Joux's attack (run in the manner described in this work) by simply choosing to create $\check{s}\cdot 2^{\frac{n}{2}}$ new message blocks in each step. We can also use the same results to improve our attack variant. This happens by choosing the number of message blocks created in each step to be $\frac{\check{s}}{\sqrt{2s}}2^{\frac{n}{2}}$ instead of $\frac{1}{\sqrt{2s}}2^{\frac{n}{2}}$.

Effectively this would change nothing in the sections 4.1, 4.2, 4.3 and 4.4. This means that the efficiency of the variant, compared with the standard statistical attack by

Joux's, remains the same i.e. the expected number of compression function calls needed to create a k-collision with our variant is less than $\frac{\breve{a}}{2} \cdot k \cdot 2^{\frac{n}{2}}$, when s is large and we wish to create large sets of colliding messages.

However, the results of subsection 4.5 are changed. Again using Maple to evaluate the expected size of the created multicollision for $\breve{a} \cdot k \cdot 2^{\frac{n}{2}}$, the expected number of compression function calls is $\frac{1}{\sqrt{2}(k-t_1)+1}(2.68)^{k-t_1}$ for $s=1$, $\frac{1}{2(k-t_2)+2}(3.07)^{k-t_2}$ for $s=2$ and finally $\frac{1}{\sqrt{6}(k-t_3)+3}(3.29)^{k-t_3}$ for $s=3$.

4.7 Further Thoughts

There are three aspects concerning the analysis of this variant that should be taken into consideration. Firstly, the algorithm memory requirement for this attack variant is $\frac{s+1}{\sqrt{2s}} \cdot 2^{\frac{n}{2}}$, while for Joux attack it is $2^{\frac{n}{2}}$, when performed in the manner described in this work.

Secondly, there exists an algorithm that can be used to perform quite efficient space-time tradeoff for the standard Joux's attack [34]. No such algorithm exists, as far as we know, for this variant.

Thirdly, in this work both attacks; the standard Joux attack and this variant, are performed in a model, where the attacker tries to complete a step and if unsuccessful starts the whole step again. As we have seen, if we run Joux's attack in optimal manner, i.e. we do not pay anttention to the amount of memory required and stop the process immediately when a collision occurs, the complexity of Joux's attack drops to $k \cdot \sqrt{\frac{\pi}{2}} \cdot 2^{\frac{n}{2}}$, when creating a 2^k-collision.

Assume now that in the variant $\frac{\breve{s}}{\sqrt{2s}} \cdot 2^{\frac{n}{2}}$ new message blocks are created in each step. With $k \cdot \sqrt{\frac{\pi}{2}} \cdot 2^{\frac{n}{2}}$ compression function calls the variant should be able to perform more than

$$p_4(s) \cdot \frac{\sqrt{\frac{\pi}{2}} \cdot k \cdot 2^{n/2} - s \cdot \frac{\breve{s}}{\sqrt{2s}} \cdot 2^{n/2}}{\frac{\breve{s}}{\sqrt{2s}} \cdot 2^{n/2}} \approx \sqrt{s} \cdot 0.800 \cdot (k - \breve{s} \cdot \sqrt{\frac{s}{\pi}})$$

general steps (and initialization phase), where $p_4(s)$ is the probability that a single step succeeds ($p_4(s) \approx 0.71533$ when s is large).

This means that the expected size of the created collision is more than

$$\frac{(2.23)^{k-\breve{s}\cdot\sqrt{\frac{s}{\pi}}}}{\sqrt{s} \cdot 0.800 \cdot (k - \breve{s} \cdot \sqrt{\frac{s}{\pi}}) + s}.$$

Thus the variant would have a slight edge over the standard version of Joux's attack, when creating really large multicollisions.

However, it is worth noticing that in the comparison above the standard Joux's attack is run the optimal way, while our variant is still caried out in the model, where the attacker tries to complete a step and if unsuccessful, starts the whole step again. Our variant would certainly also benefit from the approach, where the new hash values are created one by one and the process stops immediately, when a collision is found.

The reason, why such an approach has been chosen in [23] and in this work, is simplicity. We need some way to compare the effectiveness of this attack with the efficiency of Joux's attack and it would be extremely hard to compute the complexity of our attack variant in the most efficient form. On the whole, the considerations above give us reason to believe that our attack variant could outperform the standard Joux's attack when there is enough memory available and the attacker is creating really large multicollisions.

5 Diamond Structures and Trojan Messages

In this chapter we will present some new results concerning diamond structures and Trojan messages (see Chapter 3). In the first section we will prove that it is indeed possible to create a diamond structure with 2^d chaining hash values and length d with complexity $O(2^{\frac{n+d}{2}})$. In the second section we will create more efficient versions of the Trojan message attacks. The results of this chapter have been published in [22].

5.1 A New Method for a Diamond Structure Creation

5.1.1 A Pairing Set

We will now give a definition of a pairing set that will later be used in our attack construction.

Let $H \subseteq \{0,1\}^n$ be a finite nonempty set of hash values. A *pairing set* of H is any set $B \subseteq H \times \{0,1\}^m$ (where $\{0,1\}^m$ is the message block alphabet of the hash function) such that

(i) for each $h \in H$ there exists exactly one $x \in \{0,1\}^m$ such that $(h,x) \in B$; and
(ii) for each $(h_1,x_1) \in B$ there exists $(h_2,x_2) \in B$ such that $h_1 \neq h_2$ and $f(h_1,x_1) = f(h_2,x_2)$.

5.1.2 Intuitive Description of the Diamond Structure Construction Method

Our method advances in *jumps*, *phases* and *steps*.

- To complete our method, i.e. to create a diamond structure with breadth 2^d we need to carry out d jumps.
- In each jump we carry out several phases.
- In each phase we carry out numerous steps.
- In each step we find two distinct hash value and message block pairs (h_1,x_1), (h_2,x_2) such that $f(h_1,x_1) = f(h_2,x_2)$.

By dividing the process in an aforementioned manner and recycling the hash value and message block sets we are able to decrease the number of compression function queries.

It is quite easy to see that our method is not optimal, but we have to make a compromise between the completeness and the simplicity of computations. We will now present the pseudocode of the attack to give the reader some insight to the structure of the attack. A more rigorous description of the method will follow. As stated before these results can be found in [22].

Diamond Structure Construction Method: The Pseudocode

```
1. Input: $d \in \mathbb{N}_+$, $(1 < d < \frac{n}{2})$; $H_d \subseteq \{0,1\}^n$, $|H_d| = 2^d$
2. for $i = d$ downto 2 do   {Jumps $jump(d)$, $jump(d-1)$, ..., $jump(2)$.}
      {Input to jump $jump(i)$: a set $H_i$ of $2^i$ distinct hash values.}
   2.1. $A_{i,0} := H_i$
   2.2. Generate a set $M_{i,0} \subseteq \{0,1\}^m$ such that $|M_{i,0}| = 2^{\frac{n-i}{2}-1}$ and $|f(A_{i,0}, M_{i,0})| =$
        $2^{\frac{n+i}{2}-1}$.   {Initialization}
   2.3. $H_{i,0} := f(A_{i,0}, M_{i,0})$; $B_i := \emptyset$
   2.4. for $j = i$ downto 2 do   {Phases $phase(i,i)$, $phase(i,i-1)$, ..., $phase(i,2)$.}
          {Input to phase $phase(i,j)$: The sets $B_i$, $A_{j,0}$, $M_{j,0}$, and $H_{j,0}$}
        2.4.1. for $k = 0$ to $2^{j-2}-1$ do   {Steps $step(i,j,0)$, $step(i,j,1)$, ...,
               $step(i,j,2^{j-2}-1)$.}
                 {Input to $step(i,j,k)$: the sets $A_{j,k} \subseteq \{0,1\}^n$, $M_{j,k} \subseteq \{0,1\}^m$, $H_{j,k} =$
                 $f(A_{j,k}, M_{j,k})$, and $B_i$ such that $|A_{j,k}| = 2^j - 2k$, $|M_{j,k}| = s_{j,k}$, and
                 $|H_{j,k}| = |A_{j,k}| \cdot |M_{j,k}|$.}
               a. Generate a set $M'_{j,k} \subseteq \{0,1\}^m$ of cardinality $\lceil s_{j,k+1} - s_{j,k} \rceil$ (see lemma
                  3) such that $M'_{j,k} \cap M_{j,k} = \emptyset$ and $|f(A_{j,k}, M'_{j,k})| \geq 2^{\frac{n-j}{2}+1}$.
               b. Search distinct hash values $h_{j,k}, h'_{j,k} \in A_{j,k}$ and message blocks $x_{j,k} \in$
                  $M_{j,k}$, $x'_{j,k} \in M'_{j,k}$ such that $f(h_{j,k}, x_{j,k}) = f(h'_{j,k}, x'_{j,k})$.
               c. $A_{j,k+1} = A_{j,k} \setminus \{h_{j,k}, h'_{j,k}\}$; $M_{j,k+1} = M_{j,k} \cup M'_{j,k}$; $H_{j,k+1} =$
                  $f(A_{j,k+1}, M_{j,k+1})$; $B_i = B_i \cup \{(h_{j,k}, x_{j,k}), (h'_{j,k}, x_{j,k})\}$
               d. if $k = 2^{j-2}-1$ then
                  (i) $A_{j-1,0} := A_{j,2^{j-2}-1}$, $M_{j-1,0} := M_{j,2^{j-2}-1}$; $H_{j-1,0} := H_{j,2^{j-2}-1}$
      {Input to phase $phase(i,1)$: the set $A_{1,0} := \{h_{1,0}, h'_{1,0}\}$ of two distinct hash
      values.}
   2.5. Generate a set $M'_{1,0} \subseteq \{0,1\}^m$ of $2^{\frac{n}{2}}$ message blocks such that there exist
        $x_{1,0}, x'_{1,0} \in M'_{1,0}$ for which $f(h_{1,0}, x_{1,0}) = f(h'_{1,0}, x'_{1,0})$.   {Phase $phase(i,1)$.}
```

```
    2.6. $B_i := B_i \cup \{(h_{1,0}, x_{1,0}), (h'_{1,0}, x'_{1,0})\}$; $H_{i-1} := \{f(h,x) \,|\, (h,x) \in B_i\}$
    {Input to jump $jump(1)$: the set $H_1 := \{h_1, h_2\}$ of two distinct hash values.}
3. Generate a set $M_1 \subseteq \{0,1\}^m$ of $2^{\frac{n}{2}}$ message blocks such that there exist $x_1, x_2 \in M_1$
   for which $f(h_1, x_1) = f(h_2, x_2)$.   {Jump $jump(1)$.}
4. $B_1 := \{(h_1, x_1), (h_2, x_2)\}$; $H_0 := \{h_0\}$ where $h_0 = f(h_1, x_1) = f(h_2, x_2)$
5. Output: $B_d, B_{d-1}, \ldots, B_1$
```

Jumps

The construction of a diamond structure D with 2^d chaining hash values $d \geq 2$ is carried out in d jumps $jump(d)$, $jump(d-1)$, ..., $jump(1)$. We proceed from the leaves towards the root of the structure. Let H_d be the set of the 2^d chaining hash values. In the jump $jump(d)$, a pairing set B_d of H_d is created. The set B_d is so constructed that the cardinality of the set $H_{d-1} := \{f(h,x) \,|\, (h,x) \in B_d\}$ is 2^{d-1}. In the jump $jump(d-1)$ a pairing set B_{d-1} of H_{d-1} is created so that the cardinality of the set $H_{d-2} := \{f(h,x) \,|\, (h,x) \in B_{d-1}\}$ is 2^{d-2}. We continue like this until in the last jump $jump(1)$ a pairing set B_1 of H_1 containing only two hash values is generated. The set $H_0 := \{f(h,x) \,|\, (h,x) \in B_1\}$ contains only one element, which is the root of the diamond structure. By each jump the distance to the root of the diamond structure is decreased by one. Obviously, we are herding the chaining hash values towards the final hash value, which labels the root of our structure. We call this final hash value the diamond value of the diamond structure.

Now, each jump consists of several *phases*; since the structures of jumps are mutually identical, we give below an intuitive description of the phases (and steps) of the jump $jump(d)$ only.

Phases

The Jump $jump(d)$ begins with *initialization phase I(d)* that we will describe later. After this the jump $jump(d)$ is made up of d phases

$$phase(d,d), phase(d,d-1), \cdots, phase(d,2), phase(d,1).$$

In the phase $phase(d,d)$ of the jump $jump(d)$ we create a pairing set T_{d-1} of a subset $K_{d-1} \subseteq H_d$ of cardinality 2^{d-1}, in the phase $phase(d,d-1)$ a pairing set T_{d-2} of a subset $K_{d-2} \subseteq H_d \setminus K_{d-1}$ of cardinality 2^{d-2}, and so on, ..., in the phase $phase(d,2)$ a pairing set T_1 of a subset K_1 of $H_d \setminus (K_{d-1} \cup K_{d-2} \cup \cdots \cup K_2)$ of cardinality 2. This means that in each phase, $phase(d,i)$ where $i \in \{d, d-1, \cdots, 2\}$ we halve the number of hash values in H_d still without a pairing.

There are two hash values (forming the set K_0) still without a pairing left in H_d, so in the phase $phase(d,1)$ we search a pairing T_0 of K_0. Then we set $B_d := T_{d-1} \cup T_{d-2} \cup \cdots \cup T_0$. Thus the jump $jump(d)$ consists of d phases after which we have created a pairing set B_d of H_d; moreover the input set $H_{d-1} := \{f(h,x) \,|\, (h,x) \in B_d\}$ of the jump $jump(d-1)$ is of cardinality 2^{d-1}.

Steps

Each phase is made up of several *steps* in the following way. Consider the phase $phase(d,j)$ of the jump $jump(d)$, where $j \in \{2,3,\ldots,d\}$. As told above, in this phase we create a pairing set for a subset K_{j-1} of $H_d \setminus (K_{d-1} \cup K_{d-2} \cup \cdots \cup K_j)$ of cardinality 2^{j-1}. The phase is divided into 2^{j-2} steps

$$step(d,j,0), step(d,j,1), \ldots, step(d,j,2^{j-2}-1) \ .$$

In each step we create a pairing for two hash values in $H_d \setminus (K_{d-1} \cup K_{d-2} \cup \cdots \cup K_j)$ so that together the hash values in the pairs form a set K_{j-1} of cardinality 2^{j-1}.

A more rigorous description of each step with appropriate input and output follows. However, before we begin, we need the following lemma.

Lemma 3. Let $r \geq 2$ and n be positive integers. Define the integers $s_{r,0}, s_{r,1}, s_{r,2}, \ldots, s_{r,2^{r-2}}$ as follows.

$$s_{r,0} = \lceil 2^{\frac{n-r}{2}-1} \rceil \qquad s_{r,k+1} = s_{r,k} + \left\lceil \frac{2^{\frac{n-r}{2}+1}}{2^r - 2k} \right\rceil \quad \text{for } k = 0,1,\ldots,2^{r-2}-1$$

Then $s_{r,j} \geq \frac{2^{\frac{n+r}{2}-1}}{2^r-2j}$ for each $j \in \{0,1,\ldots,2^{r-2}\}$.

Proof. Proceed by induction on j. The case $j=0$ is clear. Suppose that $s_{r,k} \geq \frac{2^{\frac{n+r}{2}-1}}{2^r-2k}$ where $k \in \{0,1,\ldots,2^{r-2}-1\}$. Then, by definition, the inequality

$$s_{r,k+1} \geq \frac{2^{\frac{n+r}{2}-1} + 2^{\frac{n-r}{2}+1}}{2^r - 2k}$$

holds. It suffices to show that

$$\frac{2^{\frac{n+r}{2}-1} + 2^{\frac{n-r}{2}+1}}{2^r - 2k} \geq \frac{2^{\frac{n+r}{2}-1}}{2^r - 2(k+1)} \ .$$

But this follows from the inequality

$$(2^{\frac{n+r}{2}-1}+2^{\frac{n-r}{2}+1})[2^r-2(k+1)] \geq 2^{\frac{n+r}{2}-1}(2^r-2k)$$

is equivalent with $k \leq 2^{r-2}-1$. $\square$

Initialization $\mathtt{I}(d)$

As an input, we have a set $A_{d,0} := H_d$ of 2^d hash values. We first create a message block set $M_{d,0} \subseteq \{0,1\}^m$ such that

1. the cardinality of $M_{d,0}$ is $2^{\frac{n-d}{2}-1}$ and
2. the cardinality of the set $f(A_{d,0},M_{d,0}) = \{f(h,x) \big| h \in A_{d,0}, x \in M_{d,0}\}$ is $2^{\frac{n+d}{2}-1}$.

Let $H_{d,0} = f(A_{d,0},M_{d,0})$. The complexity to construct such an $H_{d,0}$ is approximately $2^{\frac{n+d}{2}-1}$. Note that our assumption on the cardinality of the set $H_{d,0}$ has an insignificant impact on the complexity. When we first create the message set $M_{d,0}$, we will almost certainly have message blocks $x,y \in M_{d,0}$ such that $x \neq y$ and $f(h_1,x) = f(h_2,y)$ for some $h_1,h_2 \in A_{s,0}$. However, we can easily replace the colliding message blocks one by one with new ones. The output of the initiation step is $A_{d,0}$, $M_{d,0}$, $H_{d,0}$.

Let now $j \in \{2,3,\ldots,d\}$ and $k \in \{0,1,2,\ldots,2^{j-2}-1\}$.

Step *step*(d,j,k)
The step takes as an input $A_{j,k}$, $M_{j,k}$, $H_{j,k}$. Here $A_{j,k}$ is a set of 2^j-2k hash values, $M_{j,k}$ is a set of $s_{j,k}$ message blocks, where $s_{j,k} \geq \frac{2^{\frac{n+j}{2}-1}}{2^j-2k}$, and

$$H_{j,k} = \{f(x,h) \,\big|\, x \in A_{j,k}, x \in M_{j,k}\}$$

is a set of hash values such that $|H_{j,k}| = |A_{j,k}| \cdot |M_{j,k}|$. Note that $|H_{j,k}| = (2^j-2k)s_{j,k} \geq 2^{\frac{n+j}{2}-1}$.

A set $M'_{j,k}$ of $\lceil s_{j,k+1} - s_{j,k} \rceil$ new messages is generated so that the cardinality of the set

$$f(A_{j,k},M'_{j,k}) = \{f(h,x) \,\big|\, h \in A_{j,k}, x \in M'_{j,k}\}$$

is at least $2^{\frac{n-j}{2}+1}$. We search for hash values $h_{j_k}, h'_{j_k} \in A_{j.k}$ and message blocks $x_{j_k} \in M_{j,k}$, $x'_{j_k} \in M'_{j,k}$ such that $f(h_{j_k},x_{j_k}) = f(h'_{j_k},x'_{j_k})$. Note that since $|H_{j,k} \times f(A_{j,k},M'_{j,k})| \geq 2^n$, the expected number of hash values h such that $h \in H_{j,k} \cap f(A_{j,k},M'_{j,k})$ is at least one. Furthermore, for the sake of simplicity of computations, we assume that (h_{j_k}, x_{j_k}) and (h'_{j_k}, x'_{j_k}) are the only colliding pairs in $A_{j,k} \times [M_{j,k} \cup M'_{j,k}]$ (more colliding pairs would of course be a good thing).

Now, what is the complexity of the actions and assumptions above? We may create the message set $M'_{j,k}$ as a statistical experiment and then compute the hash values in the set $f(A_{j,k}, M'_{j,k})$. Since $|H_{i,k} \times f(A_{j,k}, M'_{j,k})| \geq 2^n$, the probability that we find a colliding pair is according to Lemma 2 approximately $\frac{e-1}{e}$. This means that the expected number of times we have to repeat the experiment is $\frac{e}{e-1}$, according to Fact 2. Thus the complexity to create the set $M'_{j,k}$, compute the values in $f(A_{j,k}, M'_{j,k})$, and find the colliding pair is at most $\frac{e}{e-1} \cdot 2^{\frac{n-j}{2}+1}$.

Our assumptions on the cardinality of $f(A_{j,k}, M'_{j,k})$ and on the number of colliding pairs do not increase the complexity significantly. This is ensured by replacing the colliding messages in the set $M'_{j,k}$ one by one with new ones.

Let $A_{j,k+1} := A_{j,k} \setminus \{h_{j_k}, h'_{j_k}\}$, $M_{j,k+1} := M_{j,k} \cup M'_{j,k}$, and $H_{j,k+1} := f(A_{j,k+1},\ M_{j,k+1})$. Furthermore we set $B_d := B_d \cup \{(h_{j_k}, x_{j_k}), (h'_{j_k}, x'_{j_k})\}$.

As an output of this step we get $A_{j,k+1}$, $M_{j,k+1}$, $H_{j,k+1}$, and B_d.

The output of the step $step(d, j, 2^{j-2} - 1)$ (the last step of the phase $phase(d, j)$) serves as the input to the $step(d, j-1, 0)$ (the first step of the phase $phase(d, j-1)$) for each $j \in \{3, 4, \ldots, d\}$. We thus define $A_{j-1,0} := A_{j,2^{j-2}-1}$, $M_{j-1,0} := M_{j,2^{j-2}-1}$, and $H_{j-1,0} := H_{j,2^{j-2}-1}$.

We carry out our diamond structure construction by running the jumps $jump(d)$, $jump(d-1)$, $\ldots$, $jump(2)$, $jump(1)$ one after another in this order. We describe the inner realization of the jump $jump(d)$ more accurately; all the other jumps are carried out completely analogously. The jump $jump(d)$ is implemented by running all its phases $\mathtt{I}(d)$, $phase(d,d)$, $phase(d,d-1)$, $\ldots$, $phase(d,2)$, $phase(d,1)$. The last phase $phase(d,1)$ takes as its input only the set $A_{1,0} := A_{2,1}$ of two (remaining) hash values and the pairing set B_d. It searches a pairing set for $A_{1,0}$ on its own. Each phase $phase(d, j)$, $j \in \{2, 3, \ldots, d\}$ is realized by running all its steps $step(d, j, 0)$, $step(d, j, 1)$, $\ldots$, $step(d, j, 2^{j-2} - 1)$ subsequently in this order.

Note that in each phase (step, resp.), the message blocks and hash values generated in the previous phases (steps, resp.) are utilized, recycled, one could say. This means that in our method the excessive growth of the complexity can be prevented. This will be verified in the next subsection.

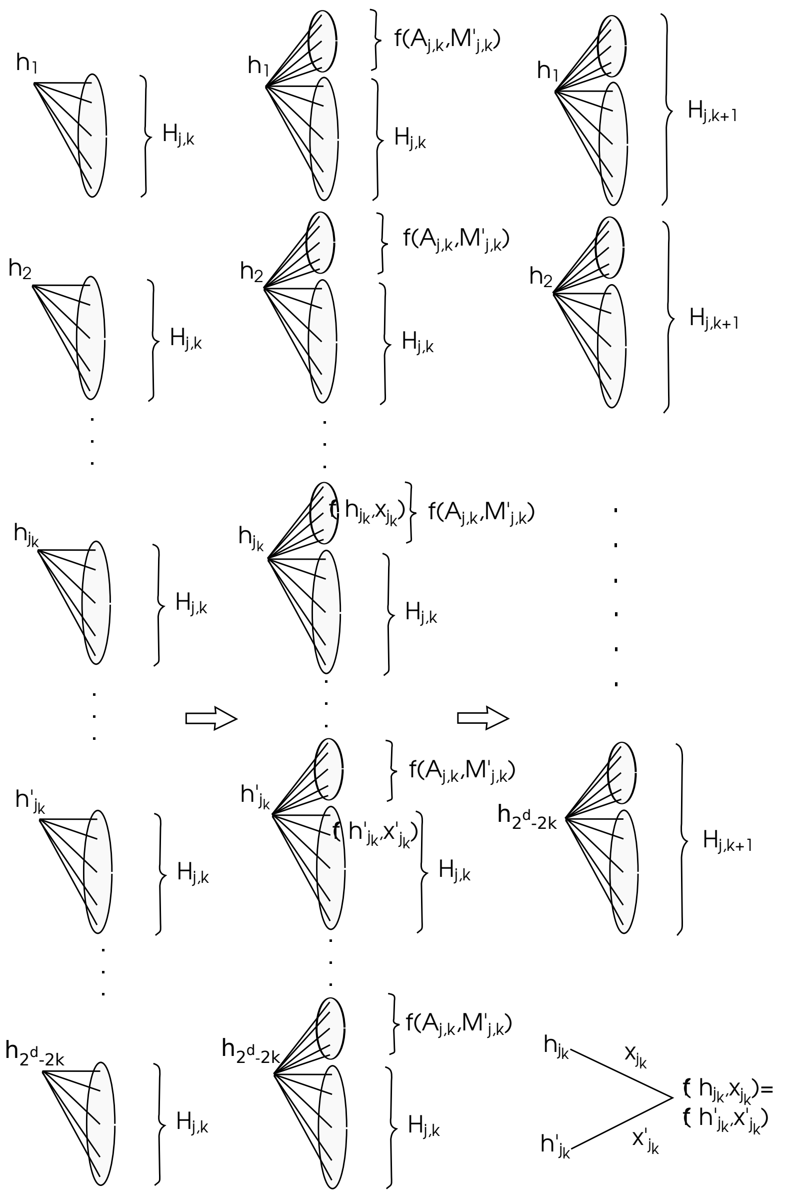


Fig 20. Step $S(d,j,k)$ of diamond structure creation.

5.1.3 The Overall Complexity of the Construction

Let us first compute the complexity of the jump $jump(d)$; recall that it consists of phases $phase(d,d)$, $phase(d,d-1)$, $\ldots$, $phase(d,2)$, $phase(d,1)$. Certainly the complexity of the jump $jump(d)$ is the sum of the expected number of compression function queries in $\mathtt{I}(d)$ and phases $phase(d,d)$, $phase(d,d-1)$, $\ldots$, $phase(d,2)$, $phase(d,1)$. Applying Lemma 3 and induction on j and k we see that $s_{j,k} \geq \frac{2^{\frac{n+j}{2}-1}}{2^{j}-2k}$ holds for each $j \in \{2,3,\ldots,d\}$ and $k \in \{0,1,\ldots,2^{j-2}-1\}$. This means that the complexity analysis given in the description of the step $step(d,j,k)$ holds. This implies that given $j \in \{2,3,\ldots,d\}$, the expected number of compression function queries to carry out the phase $phase(d,j)$ is at most $\frac{e}{e-1}2^{j-2} \cdot 2^{\frac{n-j}{2}+1}$; here 2^{j-2} naturally refers to the number of steps in the phase. The complexity of $\mathtt{I}(d)$ is approximately $2^{\frac{n+d}{2}-1}$ and of $phase(d,1)$ can be approximated to be less than $\frac{e}{e-1}2^{\frac{n}{2}+1}$. The total complexity of the jump $jump(d)$ is thus approximately

$$2^{\frac{n+d}{2}-1} + \frac{e}{e-1} \cdot \left[\sum_{j=2}^{d} [2^{j-2} \cdot 2^{\frac{n-j}{2}+1}] + 2^{\frac{n}{2}+1} \right]$$
$$< 2^{\frac{n+d}{2}-1} + \frac{e}{e-1} \left[(1+\frac{1}{\sqrt{2}})2^{\frac{n+d}{2}} + 2^{\frac{n}{2}+1} \right].$$

By the considerations above we can deduce that the complexity of the jump $jump(i)$ is at most

$$2^{\frac{n+i}{2}-1} + \frac{e}{e-1} \left[(1+\frac{1}{\sqrt{2}})2^{\frac{n+i}{2}} + 2^{\frac{n}{2}+1} \right]$$

for $i = 2,3,\ldots,d$. Once again Lemma 2 and Fact 2 prove that the complexity of the jump $jump(1)$ is at most $2 \cdot \frac{e}{e-1} \cdot 2^{\frac{n}{2}}$ compression function queries. It follows that the overall complexity of our diamond structure construction is not more than

$$\sum_{i=2}^{d} \left[2^{\frac{n+i}{2}-1} + \frac{e}{e-1} \left[(1+\frac{1}{\sqrt{2}})2^{\frac{n+i}{2}} + 2^{\frac{n}{2}+1} \right] \right] + 2 \cdot \frac{e}{e-1} \cdot 2^{\frac{n}{2}}$$
$$< (1+\frac{1}{\sqrt{2}})2^{\frac{n+d}{2}} + 2\frac{e}{e-1}(1+\frac{1}{\sqrt{2}})^2 2^{\frac{n+d}{2}} + 2d\frac{e}{e-1}2^{\frac{n}{2}}.$$

If we assume for example that $d \geq 20$, as is certainly reasonable in the attack constructions that apply a diamond structure, the total complexity will be less than

$$11 \cdot 2^{\frac{n+d}{2}}.$$

Certainly it is clear that the total complexity of the attack is $O(2^{\frac{n+d}{2}})$.

It is worth noticing that this construction method uses a more strict model in the complexity analysis than the one used in [5]. Namely, in our method each step is caried out and if it is unsuccessful, repeated again until it succeeds. Using the model of [5] we could drop the $\frac{e}{e-1}$ multiplier from the complexity of each of our steps. This would reduce the whole complexity to less than $7.6 \cdot 2^{\frac{n+d}{2}}$.

In [17] it is claimed that it is possible to create a diamond structure with approximately $2^{\frac{n+d}{2}+2}$ compression function calls. The intuitive reasoning proved to be incorrect. However, our method brings the complexity quite close to $2^{\frac{n+d}{2}+2}$. Since our method is not optimal (for the sake of simplicity) the original claim may well hold.

5.1.4 Memory Requirements

In order to run this construction method optimally, the attacker has to be able to store $H_{j,k}$ and $f(A_{j,k}, M'_{j,k})$ in each step $step(d, j, k)$. $|H_{j,k}| + |f(A_{j,k}, M'_{j,k})|$ is largest when $j = d$ but even then it is well below $2^{\frac{n+d}{2}}$. Thus the optimal memory requirement of this construction method can be approximated to be at most $2^{\frac{n+d}{2}}$. The online memory requirement of the diamond structure remains of course the same as before, i.e. $3 \cdot 2^d$.

5.1.5 Reducing the Complexity

It is quite easy to slightly reduce the complexity of the first pairing (i.e., $jump(d)$), if we can choose chaining hash values freely. One can choose an arbitrary hash value set A such that $|A| = 2^{\frac{n+d}{2}}$. After this we can fix a single message block x and compute the value $f(h,x)$ for all $h \in A$. Thus we have $2^{\frac{n+d}{2}}$ hash values and the number of possibly colliding pairs is

$$\binom{2^{\frac{n+d}{2}}}{2} = 2^{n+d-1} - 2^{\frac{n+d}{2}-1} \approx 2^{n+d-1}.$$

Since the codomain of f consists of 2^n elements, there should be approximately 2^{d-1} pairs $h, h' \in A$ such that $f(h,x) = f(h',x)$. We have now found 2^{d-1} colliding pairs with the approximated complexity $2^{\frac{n+d}{2}}$ (instead of $2^{\frac{n+d}{2}-1} + \frac{e}{e-1}\left[(1+\frac{1}{\sqrt{2}})2^{\frac{n+d}{2}} + 2^{\frac{n}{2}+1}\right]$).

As stated before, the method presented in this section does not give us the optimal complexity. A more efficient approach would be to create new message blocks one by one, to compute the respective hash values, and to search for colliding pairs after each new message block. However, we certainly still need to apply the compression function at least $2^{\frac{n+d-i}{2}}$ times to create 2^{d-i} pairs and so the total complexity of our diamond structure construction will thus not drop below $\Omega(2^{\frac{n+d}{2}})$.

5.2 New Versions of the Trojan Message Attacks

In this section we will present two new versions of the Trojan message attacks (see [1]). First however we will take a look at the Trojan message attack against the random oracle hash function.

5.2.1 Trojan Message Resistance

Assume for a moment that $\mathtt{H}: \{0,1\}^* \to \{0,1\}^n$ is a random oracle hash function and P is a set of messages with cardinality $k \in \mathbb{N}_+$. Suppose that M is another set of messages such that $|M| = 2^s$ for some $s \in \mathbb{N}_+$. The probability that a random message t satisfies property: for each $p \in P$ there exists $x \in M$ satisfying $\mathtt{H}(pt) = \mathtt{H}(x)$, is at most $\left(\frac{2^s}{2^n}\right)^k$. Assume now that we create a new message set T where $|T| = 2^j$, $j \in \mathbb{N}_+$. The expected number of messages $t \in T$ such that for each $y \in P$ there exists $x \in M$ satisfying $\mathtt{H}(yt) = \mathtt{H}(x)$ is $2^j \cdot \left(\frac{2^s}{2^n}\right)^k = 2^{j+ks-kn}$. In order to successfully complete the attack we should be able to create at least one Trojan message satisfying the given conditions, so the above expected number of messages should be at least one. This means that $j + ks - kn \geq 0$.

The number of hash function queries needed to create the sets M and T is of order $2^j + 2^s$. We can minimize this by setting $j = s = \frac{kn}{k+1}$. So the number of hash function queries needed is bounded from below by $\Omega(2^{\frac{k}{k+1} \cdot n})$. It is interesting to see that this is almost equal to the number of hash function queries needed to create a $k-$collision [32].

5.2.2 Weak Trojan Attack

We shall now present a new variant of the Collision Trojan Attack. The complexity of our construction is lower than that of the original one, while it gives the attacker more freedom to choose the content of the created second preimage. To ensure this we will assume that the attacker is, in addition to the prefix choice p from the set P, challenged with another prefix choice v from a prefix set V such that $|V| \leq 2^r$ in the second phase of the attack. The attacker now has to find a suffix s such that $f^{+}(h_0, vs) = f^{+}(h_0, pt)$, where t is the Trojan message created by the attacker in the first phase.

The attack proceeds through the following steps in the offline phase:

(1) Create a diamond structure D with chaining hash values $f^{+}(h_0, v)$, where $v \in V$. Denote the diamond value with h'.
(2) Create an expandable message with the initial value h' and the expanding value h''.
(3) Create a large set of message blocks Y and calculate $f(h'', y)$ for each $y \in Y$.
(4) Connect the prefix set P to the expandable message, one prefix at a time by searching for collisions to hash values created in the step (3).

In the online phase the attacker uses the expandable message to ensure that the created second preimage has the same length as the original message. We will now describe the process with more details.

In the offline phase the attacker creates a diamond structure with chaining hash values $\{f^{+}(h_0, v) | v \in V\}$. Since $|V| \leq 2^r$, the complexity of this certainly is $O(2^{\frac{n+r}{2}})$. Assume that $r \geq 20$. We can now approximate the complexity to be at most $11 \cdot 2^{\frac{n+r}{2}}$.

Assume now that the diamond value is h'. Next the attacker creates an expandable message, starting from the hash value h', with minimum block length $r+1$ and maximum block length $2^{r+1} + r$. The complexity of this effort is at most $\frac{e}{e-1}(r+1) \cdot 2^{\frac{n}{2}+1}$ [18]. Assume that the final hash value of the expandable message is h''.

The attacker then creates a set Y consisting of $2^{\frac{n+r}{2}}$ random message blocks and computes the respective hash values $f(h'', y)$ for each $y \in Y$. This requires $2^{\frac{n+r}{2}}$ compression function queries. In addition, the attacker now chooses any message x_0 such that the length of x_0 is $2r+1$.

Now the attacker searches for a message block x_1 such that $f^{+}(h_{0,1}, x_0x_1) = f(h'', y)$ for some $y \in Y$. Denote $h_1 := f^{+}(h_{0,1}, x_0x_1)$. The complexity of finding such an x_1 is approximately $2^{\frac{n-r}{2}}$. The attacker sets $y_1 := y$ and is now ready for the second step of the first phase.

Consider the step $i \in \{2, 3, \cdots, 2^r\}$ of the first phase of the attack. The attacker computes $h'_{i-1} := f^{+}(h_{0,i}, x_0x_1x_2 \cdots x_{i-1})$ and searches for a message block x_i such that $f(h'_{i-1}, x_i) = f(h'', y)$ for some $y \in Y$. Once again the complexity of finding x_i is $2^{\frac{n-r}{2}}$. Denote $h_i := f(h'_{i-1}, x_i)$ and $y_i := y$. Once the attacker has completed the 2^r steps, the offline phase is done. The attacker now forms the Trojan message $t = x_0x_1x_2 \cdots x_{2^r}$.

Assume that in the second phase the attacker is challenged with the prefix p_j from the prefix set P, $j \in \{1, 2, \ldots, 2^r\}$ and a second prefix $v \in V$. Assume that z

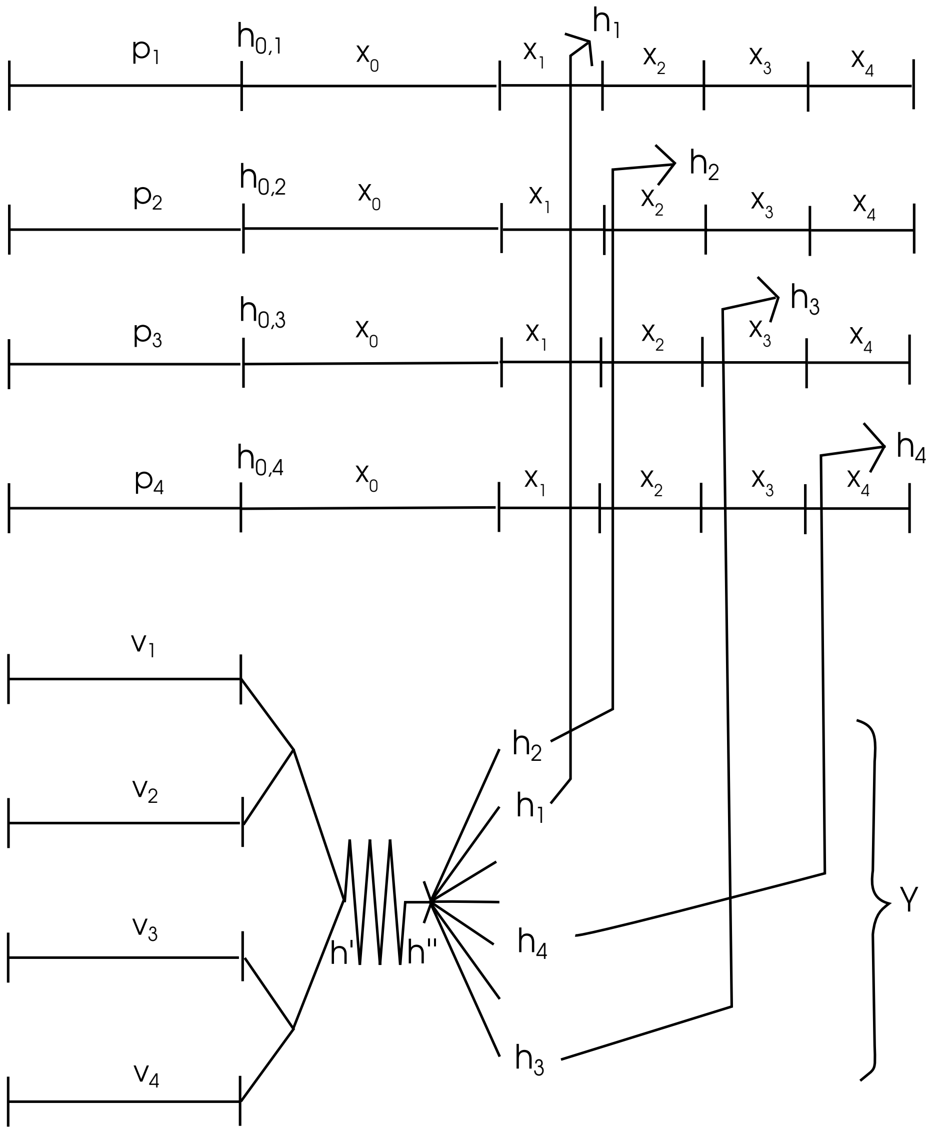


Fig 21. Weak Trojan Attack example when $r=2$**.**

is the expandable message with length $j+r$ and y is the path from $f^+(h_0,v)$ to h', i.e., $f^+(f^+(h_0,v),y)=h'$. Obviously $f^+(h_0,p_jx_0x_1\cdots x_{2^r})=f^+(h_0,vyzy_jx_{j+1}x_{j+2}\cdots x_{2^r})$, so clearly $vyzy_jx_{j+1}x_{j+2}\cdots x_{2^r}$ is a second preimage for p_jt.

The messages created in this way have the length $k+2r+1+2^r$. The offline complexity of this attack is $13\cdot 2^{\frac{n+r}{2}}$, while the online complexity is negligible. Since Collision Trojan Attack has the complexity of approximately $2.5\cdot 2^{\frac{n}{2}+r}$, when completed in the manner described in this work, the advantage of Weak Trojan Attack is obvious, when considering the number of required compression function calls. The algorithm memory requirement of Weak Trojan Attack is however somewhat higher $2^{\frac{n+r}{2}}$. The online memory requirement of Weak Trojan Attack is approximately $6\cdot 2^r$.

5.2.3 Strong Trojan Attack

We shall use both expandable messages [18] and elongated diamond structures [17] to reduce the complexity of the original Herding Trojan Attack.

In the offline phase the attacker goes through the following steps:

(1) Create an elongated diamond structure D_e and denote the final hash value with h'.
(2) Connect the hash value h' to the prefix set P in the same manner as was done in Weak Trojan Attack.

In the online phase the attacker goes through steps:

(3) Connect the new prefix p to the elongated diamond structure D_e.
(4) Use the expandable message to ensure that the created second preimage has the same length as the original one.

The attacker begins the first phase of the attack by creating a random message $z=z_1z_2\cdots z_{2^s}$, where $s\geq r$ and $z_1,z_2,\ldots,z_{2^s}$ are message blocks. Then she/he chooses random hash values $b_1,b_2,\cdots b_{2^d}$ (where $d\in\mathbb{N}_+,d\leq s$), computes $a_i:=f^+(b_i,z)$ for $i=1,2,\ldots 2^d$, and creates a diamond structure with chaining hash values $a_1,a_2,\ldots,a_{2^d}$. The number of compression function queries needed is $11\cdot 2^{\frac{n+d}{2}}$. Assume that the final root hash value of the structure is h'. The attacker then continues by constructing an expandable message, starting from the hash value h' with the minimum length $s+1$ and the maximum length $s+2^{s+1}$. The complexity of the construction is $\frac{e}{e-1}(s+1)\cdot 2^{\frac{n}{2}+1}$. Suppose that the final hash value of the expandable message is h''.

The attacker now creates a set Y containing $2^{\frac{n+r}{2}}$ random message blocks and computes all the hash values $f(h'',y)$, $y \in Y$. She also chooses an arbitrary message x_0 of length $2^s + d + s + 1$, and searches a message block x_1 such that $f^+(h_{0,1}, x_0x_1) = f(h'',y)$ for some $y \in Y$. The complexity of finding such x_1 and y is $2^{\frac{n-r}{2}}$. Denote $h_1 := f^+(h_{0,1}, x_0x_1)$ and $y_1 := y$.

Let $i \in \{2,3,\cdots,2^r\}$. In the step i of the first phase of the attack, the attacker computes $h'_{i-1} := f^+(h_{0,i}, x_0x_1x_2\cdots x_{i-1})$ and searches for a message block x_i such that $f(h'_{i-1}, x_i) = f(h'',y)$ for some $y \in Y$. To find such x_i and y takes approximately $2^{\frac{n-r}{2}}$ compression function queries. Finally the attacker sets $h_i := f(h'_{i-1}, x_i) = f(h'',y)$ and $y_i := y$.

After the 2^r steps our attacker chooses $t := x_0x_1x_2\cdots x_{2^r}$ for the Trojan message and has completed the first (offline) phase of the attack. Since there were altogether 2^r steps above, the complexity of completing them all is $2^{\frac{n+r}{2}}$. This means that the total complexity of the offline phase is approximately $11 \cdot 2^{\frac{n+d}{2}} + 2^{\frac{n+r}{2}+1}$.

Assume now that the attacker is challenged with a prefix $p_j \in P$ and another prefix p with the length smaller than $|p_j|$. The attacker now searches for a connection message x such that the length of px is $|p_j|$, and $f^+(h_0, px) = h'''$ for some hash value h''' that satisfies the condition $h''' = f^+(b_i, z_1z_2\cdots z_k)$ for some $i \in \{1,2,\cdots d\}$ and $k \in \{1,2,\cdots 2^s\}$. Assume now that message y is the path from h''' to the hash value h' in the diamond structure, i.e., $f^+(h_0, pxy) = h'$; the length of y is clearly $2^s + d - k$. Assume furthermore that w is the expandable message chosen so that the total length of the message $pxywy_jx_{j+1}x_{j+2}\cdots x_{2^r}$ is $2^s + 2^r + |p_j| + d + s + 1$.

Now $f^+(h_0, pxywy_j) = f^+(h_0, p_jx_0x_1\cdots x_j) = h_j$ so $pxywy_jx_{j+1}x_{j+2}\cdots x_{2^r}$ is the second preimage for the message p_jt. The length of both messages is $2^s + 2^r + k + d + s + 1$. The complexity of finding x is in 2^{n-d-s}, which means that the complexity of the online phase is also 2^{n-d-s}. The algorithm memory requirement of the attack is $2^{\frac{n+d}{2}}$ or $2^{\frac{n+r}{2}}$, whichever is higher. Thus the algorithm memory requirement is $max\{2^{\frac{n+d}{2}}, 2^{\frac{n+r}{2}}\}$. The attacker has to store the elongated diamond structure and thus the online memory requirement of the attack is approximately $2^{d+s+1} + 2^{r+1}$. If we choose $d = \frac{n-2s}{3}$, the total complexity of the attack is $12 \cdot 2^{\frac{2n-s}{3}} + 2^{\frac{n+r}{2}+1}$.

This of course means that, if we are able to create longer messages and are able to store the elongated diamond structure, we can reduce the total complexity of the attack. Ideally, we could choose $s = \frac{n-3r}{2}$, giving us the total complexity of $14 \cdot 2^{\frac{n+r}{2}}$. For example in SHA-1 the maximum length of the message is 2^{54} message blocks while $n = 160$. This implies that for $r = 20$ we will have $14 \cdot 2^{\frac{n+r}{2}} = 14 \cdot 2^{90}$, if we can

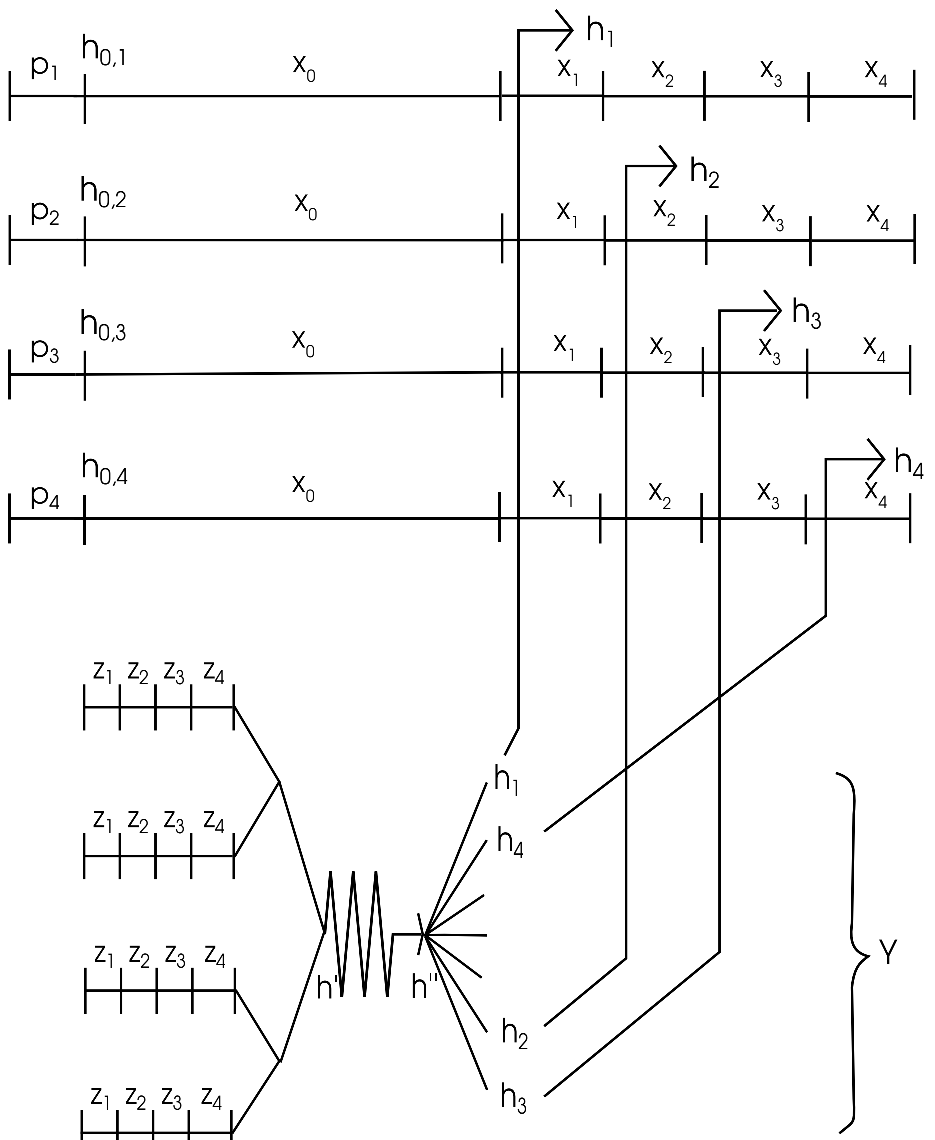


Fig 22. Example of the offline phase in Strong Trojan Attack when $d = s = r = 2$.

choose the length of the message to be 2^{50} message blocks. Creating basic second preimage attacks, presented in [18] and [2], against messages with that length would have complexity greater than 2^{110}.

In practice messages are of course a lot shorter. However, if we are able to choose, for example $s = \frac{n}{5}$, we would have the total complexity of less than $12 \cdot 2^{\frac{3n}{5}}$ in comparison to over $2^{\frac{4n}{5}}$ offered by ordinary second preimage attacks against messages with the length $2^{\frac{n}{5}}$, while $s = \frac{n}{11}$ would give us complexity less than $12 \cdot 2^{\frac{7n}{11}}$ in comparison to complexity over $2^{\frac{10n}{11}}$.

5.3 Tables

The results of this chapter are summarized on the following tables. We have also summarized the effect the new diamond construction method has on the complexities and algorithm memory requirements of some of the attacks analyzed in Chapter 3. It is worth noticing that the model used to evaluate the complexity and algorithm memory requirement of the new construction method for the diamond structure is more demanding than the one used for the original version. Using the same method would decrease the constant multipliers of the complexities of all the attacks that apply the new method of the diamond structure creation.

Once again we wish to point out that the results concern the attacks carried out in the way described in this work. They are thus certain but rough upper bounds.

Table 5. Complexities and memory requirements of diamond structure construction methods.

Diamond Structure Creation Method	Complexity	AMR	OMR
Old Method [5, 17]	$4 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$	$0.83 \cdot \sqrt{d} \cdot 2^{\frac{n+d}{2}}$	$3 \cdot 2^d$
New Method [22]	$11 \cdot 2^{\frac{n+d}{2}}$	$2^{\frac{n+d}{2}}$	$3 \cdot 2^d$

The complexity, AMR and OMR of the old diamond structure creation mehthod are based on analysis presented in [5]. The original work [17] claims the complexity of approximately $2^{\frac{n+d}{2}+2}$. However, the analysis was not correct as was proven in [5].

Table 6. Offline and online complexities of Trojan message attacks.

Trojan Message Attack Type	Offline	Online
Collision Trojan Attack [1]	$2.5 \cdot 2^{\frac{n}{2}+r}$	Negligible
Herding Trojan Attack [1]	$11 \cdot 2^{\frac{n+d}{2}} + \frac{e}{e-1} \cdot 2^{\frac{n}{2}+r+1}$	2^{n-d}
Weak Trojan Attack [22]	$13 \cdot 2^{\frac{n+r}{2}}$	Negligible
Strong Trojan Attack [22]	$11 \cdot 2^{\frac{n+d}{2}} + 2^{\frac{n+r}{2}+1}$	2^{n-d-s}

The length of the created message is 2^s and the size of the prefix set is 2^r, where $r < s$. We assume that the attacks use the diamond structure creation method of this work.

Table 7. Memory requirements and attacker's conrol of Trojan message attacks.

Trojan Message Attack Type	AMR	OMR	Control
Collision Trojan Attack [1]	$2^{\frac{n}{2}+1}$	2^{r+1}	None
Herding Trojan Attack [1]	$2^{\frac{n+d}{2}}$	$3 \cdot 2^d + 2^{r+1}$	Any pr.
Weak Trojan Attack [22]	$2^{\frac{n+r}{2}}$	$6 \cdot 2^r$	Restricted pr.
Strong Trojan Attack [22]	$max\{2^{\frac{n+d}{2}}, 2^{\frac{n+r}{2}}\}$	$2^{d+s+1} + 2^{r+1}$	Any pr.

We assume that the attacks use the diamond structure creation method of this work. Control refers to the control the attacker has over the created second preimage in the online phase. Restricted pr. means that the attacker can choose the prefix of the second preimage from the pregenerated set with 2^r prefixes. Any pr. means that only the length of the prefix is restricted.

Table 8. The effect of the new diamond construction method on complexities and memory requirements of some attacks when *d=n/3*.

Attack Type	Complexity	AMR
Old Herding Attack [17]	$(4\cdot\sqrt{\frac{n}{3}}+1)\cdot 2^{\frac{2n}{3}}$	$0.83\cdot\sqrt{\frac{n}{3}}\cdot 2^{\frac{2n}{3}}$
Old Hash Twice Herding [1]	$(12\cdot\sqrt{\frac{n}{3}}+3)\cdot 2^{\frac{2n}{3}}$	$0.83\cdot\sqrt{\frac{n}{3}}\cdot 2^{\frac{2n}{3}}$
Old Zipper Hash Herding Variant [1]	$(8\cdot\sqrt{\frac{n}{3}}+2)\cdot 2^{\frac{2n}{3}}$	$0.83\cdot\sqrt{\frac{n}{3}}\cdot 2^{\frac{2n}{3}}$
Old Hash Twice Second Preimage [1]	$(8\cdot\sqrt{\frac{n}{3}}+2)\cdot 2^{\frac{2n}{3}}+2^{n-l}$	$0.83\cdot\sqrt{\frac{n}{3}}\cdot 2^{\frac{2n}{3}}$
New Herding Attack [17]	$12\cdot\frac{2n}{3}$	$2^{\frac{2n}{3}}$
New Hash Twice Herding [1]	$36\cdot 2^{\frac{2n}{3}}$	$2^{\frac{2n}{3}}$
New Zipper Hash Herding Variant [1]	$24\cdot 2^{\frac{2n}{3}}$	$2^{\frac{2n}{3}}$
New Hash Twice Second Preimage [1]	$24\cdot 2^{\frac{2n}{3}}+2^{n-l}$	$2^{\frac{2n}{3}}$

Hash Twice second preimage attacks are against original message with length 2^l in message blocks.

6 Generalized Iterated Hash Functions I: Attack Consideration Based on Classical Combinatorics and Permutations

As we have seen, it is possible to use the Joux's method to create large, exponential size multicollisions, while at the same time the complexity of the attack rises only linearly. This means that the standard iterated hash function is unable to achieve the *generalized collision resistance* of the random oracle hash functions. The question arises whether or not the ideas of Joux's can be applied in a broader setting, i.e.: can Joux's approach be used to create multicollisions in generalized iterated hash functions? In the following we shall see that under certain assumptions this is indeed possible.

One possible way to use generalized iterated hash functions would be to apply them to the concatenation structure (see. 2.5.1) so that ${f_1}^+$ and f_2^+ are generalized iterated hash functions instead of standard iterated hash functions. The results of this chapter imply that the collision attack against concatenated structure presented in [15] (see also 3.1.1) can be applied against such a structure as well.

Traditionally, in combinatorics on words one studies unavoidable regularities that appear in sufficiently long strings over a fixed size alphabet. The viewpoint inspired by the generalized iterated hash functions is different. Assume that the sequence $\hat{\alpha} = (\alpha_1, \alpha_2 \ldots)$ and the compression function f define a generalized iterated hash function $H_{\hat{\alpha},f}$ (see Definition 3). Since in practice each message block can certainly be used in the hashing process only a limited number of times, we assume that the number of occurrences of each symbol in $\alpha_i, i \in \{1, 2, \cdots\}$ is restricted by a fixed given constant. In other words, we assume that, for some $q \in \mathbb{N}_+$, the sequence $\hat{\alpha}$ is $q-$bounded. On the other hand, we assume that each message block is used in the hashing process at least once.

So our question is whether $H_{\hat{\alpha},f}$ can achieve the generalized collision resistance of a random oracle hash function when $q > 1$. In this chapter, we will study previously proven results that show that this is not the case.

This question was investigated first for 2-bounded hash functions in [30]. In the article [14] the results of [30] were further generalized and used to study any generalized hash function that is based on a $q-$bounded sequence. The approach and basic ideas of

[14] are correct and brilliant. A mathematically rigorous consideration of the results was provided in [19] (see also [13]).

It was shown that it is possible to create a 2^k-collision in any q-bounded generalized iterated hash function with $O(g(n,q,k)\,2^{\frac{n}{2}})$ queries on f, where $g(n,q,k)$ is a function of n,q and k which is polynomial with respect to n and k, but triple exponential with respect to q.

The results were studied and improved further in [21] (see also [20]), where it was shown that it is possible to create 2^k-collision in any q-bounded generalized iterated hash function with $O(g(n,q,k)\,2^{\frac{n}{2}})$ queries on f, where $g(n,q,k)$ is only double exponential with respect to q. In Chapter 7 we will show that it is possible to lower the $g(n,q,k)$ even further.

6.1 Nested Multicollision Attack Schema (*NMCAS*)

Below we will describe a general (and informal) attack procedure that, given $H_{\hat{\alpha},f}$, $h_0 \in \{0,1\}^n$ and $k \in \mathbb{N}_+$ creates a 2^k-collision on any generalized iterated hash function $H_{\hat{\alpha},f}$ with initial value h_0. The attacks of this chapter and next chapter follow the procedure that was presented in [19].

Procedure Schema *NMCAS*

Input: A generalized iterated hash function $H_{\hat{\alpha},f}$, initial value $h_0 \in \{0,1\}^n$, positive integer k.

Output: A 2^k-multicollision on $H_{\hat{\alpha},f}$.

Step 1: Choose (a large) $l \in \mathbb{N}_+$. Consider the lth element α_l of the sequence $\hat{\alpha}$. Let $\alpha_l = i_1 i_2 \cdots i_s$, where $s \in \mathbb{N}_+$ and $i_j \in \mathbb{N}_l$ for $j = 1,2,\ldots,s$.

Step 2: Fix a (large) set of *active indices* $A \subseteq \mathbb{N}_l = \{1,2,\ \ldots,l\}$.

Step 3: Factorize the word α_l into nonempty strings appropriately , i.e., find $p \in \{1,2,\ldots,s\}$ and $\beta_i \in \mathbb{N}_l^+$ such that $\alpha_l = \beta_1\beta_2\ldots\beta_p$ and $A \subseteq \mathrm{alph}(\beta_i)$ for $i = 1,2...,p$.

Step 4: Based upon the active indices, create a large multicollision on f_{β_1}. More precisely, find message block sets $M_1, M_2, \ldots, M_l$ satisfying the following properties.

(i) If $i \in \mathbb{N}_l \setminus A$, then the set M_i consists of one (constant) message block $x_{i1} = iv$.

(ii) If $i \in A$, then the set M_i consists of two different message blocks x_{i1} and x_{i2}.

(iii) The set $M = M_1 M_2 \cdots M_l = \{u_1 u_2 \cdots u_l \mid u_i \in M_i \text{ for } i = 1,2,\ldots,l\}$ is a $2^{|A|}$-multicollision on f_{β_1} with initial value h_0.

Step 5: Based on the set $C_1 = M$, find message sets $C_2, C_3, \ldots, C_p$ such that

(iv) $C_p \subseteq C_{p-1} \subseteq \cdots \subseteq C_1 = M$.

(v) For each $j \in \{1,2,\ldots,p\}$ the set C_j is a (large) multicollision on $f_{\beta_1\beta_2\cdots\beta_j}$ with initial value h_0.

(vi) $|C_p| = 2^k$.

Step 6: Output C_p.

It should be clear that, if the above procedure is successfully carried out, then

$$H_{\hat{\alpha},f}(h_0,x) = H_{\hat{\alpha},f}(h_0,x')$$

for all $x,x' \in C_p$.

6.2 Basic Structure of the Attack

In Chapter 7, we will present the best known attack against the $q-$bounded generalized iterated hash function. We will now take a closer look at these previous versions of the attacks against $q-$bounded generalized iterated hash functions presented in [14, 19, 21, 30]. For this we need the following definitions.

A binary relation R of the set X is a *partial order* (*in* X), if it is irreflexive ($\forall x \in X : (x,x) \notin R$), antisymmetric ($\forall x,y \in X : (x,y) \in R \Rightarrow (y,x) \notin R$) and transitive ($\forall x,y,z \in X : (x,y) \in R \wedge (y,z) \in R \Rightarrow (x,z) \in R$).

Let $\prec$ be a partial order in X. Call $(X,\prec)$ a *partially ordered set*. The elements $x,y \in X$ are *incomparable* (in $(X,\prec)$) if neither $x \prec y$ nor $y \prec x$ holds. The nonempty finite sequence $x_1,x_2,\ldots,x_r$ of elements of X is a *chain* of $(X,\prec)$ if $x_i \prec x_{i+1}$ for $i \in \{1,2,\ldots,r-1\}$. Above $r \in \mathbb{N}_+$ is the *length* of the chain $x_1 \prec x_2 \cdots \prec x_r$.

Consider now a finite partially ordered set $(X,\prec)$, i.e., a partially ordered set such that X is finite. The *maximum number of incomparable elements* of $(X,\prec)$ is the cardinality of the largest set $Y \subseteq X$ such that the elements of Y are pairwise incomparable. Finally, let the *maximum chain length* of $(X,\prec)$ be the largest number $r \in \mathbb{N}_+$ such that there exists a chain of length r in $(X,\prec)$.

Let us now investigate partial orders induced by words. Let α be a nonempty word. Define the binary relation $\prec_\alpha$ of $\text{alph}(\alpha)$ as follows. For each $a,b \in \text{alph}(\alpha)$, let $a \prec_\alpha b$ hold if and only if $a \neq b$ and each occurrence of a in α happens before each occurrence of b in α. Certainly, if $a \prec_\alpha b$, then there exist words α_1 and α_2 such that $\alpha = \alpha_1\alpha_2$ and $|\alpha_1|_b = |\alpha_2|_a = 0$. Obviously, $\prec_\alpha$ is irreflexive, antisymmetric and transitive, so $(\text{alph}(\alpha),\prec_\alpha)$ is a partially ordered set. Call the elements of a nonempty set $A \subseteq \mathrm{alph}(\alpha)$ *independent* (with respect to $\prec_\alpha$), if they form a chain in $(\mathrm{alph}(\alpha), \prec_\alpha)$.

The *projection morphism* π_B from A^* into B^*, where B is nonempty and $B \subseteq A$, is defined by $\pi_B(c) = c$ if $c \in B$ and $\pi_B(c) = \varepsilon$ if $c \in A \setminus B$. Given a word w over the alphabet A, define the word $(w)_B$ as follows: $(w)_B = \varepsilon$ if $\pi_B(w) = \varepsilon$ and $(w)_B = a_1 a_2 \cdots a_s$ if $\pi_B(w) \in a_1^+ a_2^+ \cdots a_s^+$, where $s \in \mathbb{N}_+$, $a_1, a_2, \ldots, a_s \in B$, and $a_i \neq a_{i+1}$ for $i = 1, 2, \ldots, s-1$.

Example 1. Let $\alpha = 1 \cdot 3 \cdot 3 \cdot 5 \cdot 1 \cdot 2 \cdot 4$ and $B = \{1, 3, 4\}$. Now $\pi_B(\alpha) = 1 \cdot 3 \cdot 3 \cdot 1 \cdot 4$ and $(\alpha)_B = 1 \cdot 3 \cdot 1 \cdot 4$

Let α be a word such that $\mathrm{alph}(\alpha) \subseteq \mathbb{N}_+$ and u a word in $(\{0,1\}^m)^+$. Assume that $\alpha = a_1 a_2 \cdots a_s$, where $s \in \mathbb{N}$ and $a_i \in \mathbb{N}_+$ for $i = 1, 2, \ldots, s$ and $u = u_1 u_2 \cdots u_l$ where u_j is a message block for $j = 1, 2, \ldots, l$. Let $u_k = iv$, where iv is any fixed message block, for all $k \in \mathrm{alph}(\alpha) \setminus \mathbb{N}_l$. Define

$$u(\alpha) = u_{a_1} u_{a_2} \cdots u_{a_s} .$$

Then $u(\alpha)$ is the word in which the message blocks $u_1, u_2, \ldots, u_l$ of u are written in the order determined by α so that, if k is an element of $\mathrm{alph}(\alpha)$, but does not belong to $\mathbb{N}_l = \{1, 2, \ldots, l\}$, then the respective u_k is equal to the constant message block iv.

Example 2. Let $u = u_1 u_2 u_3 u_4 u_5$, where u_i is a message block for $i = 1, 2, \cdots, 5$ and $\alpha = 1 \cdot 3 \cdot 3 \cdot 5 \cdot 1 \cdot 2 \cdot 4$. Now $u(\alpha) = u_1 u_3 u_3 u_5 u_1 u_2 u_4$ and

$$f_\alpha(h_0, u) = f^+(h_0, u_1 u_3 u_3 u_5 u_1 u_2 u_4).$$

The idea behind the successful construction of the attack is the fact, that since $\hat{\alpha}$ is q-bounded, unavoidable regularities start to appear in the word α_l of $\hat{\alpha}$, when l is increased. The attacks presented in the articles [14, 19, 21, 30] create a 2^k-collision by finding a factorization $\alpha_l = \beta_1 \beta_2 \cdots \beta_p$ and a set $A \subseteq \mathrm{alph}(\alpha_l)$, such that $|A| = kn^{p-1}$ and the following properties are satisfied.

(P1) $\alpha_l = \beta_1 \beta_2 \cdots \beta_p$ and the word $(\beta_i)_A$ is a permutation of A for $i = 1, 2, \ldots, p$

(P2) For any $i \in \{1, 2, \ldots, p-1\}$, if $(\beta_i)_A = z_1 z_2 \cdots z_{n^{p-i}k}$ is a factorization of $(\beta_i)_A$ such that $|\mathrm{alph}(z_j)| = n^{i-1}$ for $j = 1, 2, \ldots n^{p-i}k$ and $(\beta_{i+1})_A = u_1 u_2 \cdots u_{n^{p-i-1}k}$ is a factorization of $(\beta_{i+1})_A$ such that $|\mathrm{alph}(u_j)| = n^i$ for $j = 1, 2, \ldots n^{p-i-1}k$, then for each $j_1 \in \{1, 2, \ldots,\ n^{p-i}k\}$, there exists $j_2 \in \{1, 2, \ldots,\ n^{p-i-1}k\}$ such that

$\text{alph}(z_{j_1}) \subseteq \text{alph}(u_{j_2})$. In other words, for each $j_2 \in \{1,2,\ldots,\ n^{p-i-1}k\}$ there exist exactly n integers $j_1 \in \{1,2,\ldots n^{p-i}k\}$ such that $\text{alph}(z_{j_1}) \subseteq \text{alph}(u_{j_2})$, while $\text{alph}(z_{j_1}) \bigcap \text{alph}(u_{j_3}) = \emptyset$ for all $j_3 \in \{1,2,\ldots,\ n^{p-i-1}k\}$, $j_3 \neq j_2$.

It is worth noticing that here l tells us the length of the message in message blocks. When we increase the number of message blocks in the message, we increase l. On the other hand, since we assume that each message block is used in the hashing process at least once, the number of the message blocks is equal to $|\text{alph}(\alpha_l)| = l$.

So our goal is to create a 2^k-collision on f_{α_l}, with messages that are l message blocks long. Now the question is, how large l, i.e. the number of message blocks in the message, must be in order to ensure that the attacker is able to find a set A such that (P1) and (P2) are satisfied.

When we have found a factorization of α and a set A satisfying properties (P1) and (P2) we have also completed steps 1, 2 and 3 of the $NMCAS$. Previous versions of the attacks (found in [14, 19, 21]) all approach the problem in the same manner. First the attacker searches for a huge set $A' \subseteq \text{alph}(\alpha_l)$ that satisfies property (P1). After this, the attacker searches for such $A \subseteq A'$ that A satisfies property (P2) and $|A| = kn^{p-1}$.

In the next chapter, we will abandon this approach and create a new one. However, first we will take a look at the approaches that have been used in articles [14, 19, 21]) to search such an α_l and a set A that properties (P1) and (P2) are satisfied.

6.3 First Step: Creating Unavoidable Permutations

The attacks in articles [14, 19, 30] are all based on the same result, formalized in [19] as Lemma 4.8. We present the result below.

Lemma 4. Let r,n and q be positive integers and α a $q-$bounded word such that $\text{alph}(\alpha) \geq r \cdot s$. Then either (i) the maximum chain length of $(\text{alph}(\alpha), \prec_\alpha)$ is at least r; or (ii) the maximum number of pairwise incomparable elements in $(\text{alph}(\alpha), \prec_\alpha)$ is greater than s.

We omit the proof that is based on Dilworth's Theorem [9]. The proof can be found in [19].

As stated before, [30] studies generalized hash functions where $\hat{\alpha}$ is $2-$bounded. By using the lemma above, it is easy to see that, when $|\text{alph}(\alpha)| = k^2 n$ and $\alpha \in \hat{\alpha}$ either:

1. there exists a set A_1 of size k that forms a chain in α. This is equivalent to claiming that they satisfy the condition: $(\alpha)_{A_1}$ is a permutation of A_1 or
2. there exists a set A_2 of size nk that satisfies condition: There exists a factorization $\alpha = \beta_1\beta_2$, where each symbol of A_2 appears exactly once in β_1 and β_2. This means that $(\beta_1)_{A_2}$ and $(\beta_2)_{A_2}$ are permutations of A_2 for some $\alpha = \beta_1\beta_2$.

It is worth noticing that here $|A_1| \neq |A_2|$, while clearly both A_1 and A_2 satisfy property (P1). In case 1, we have only one permutation and thus also (P2) is satisfied. In case 2, the attacker simply divides $(\beta_2)_{A_2}$ into k equal length (n) intervals and $(\beta_1)_{A_2}$ into nk equal length (1) intervals, in order to satisfy property (P2). Later Theorem 3 will show that we are now able to produce a 2^k-collision.

In their work [14] Hoch and Shamir generalized this problem and used chains (or independent elements) and intervals in sequence to solve it. They showed that, assuming that α_l is q-bounded and $|\text{alph}(\alpha_l)|$ is large enough, the attacker is able to create a set A and factorization $\alpha_l = \beta_1\beta_2\cdots\beta_p$ satisfying properties (P1) and (P2). In the article [19], the question was formalized as a problem of word combinatorics and the boundaries of the proofs of [14] were corrected.

Assume that $|\text{alph}(\alpha_l)| > rs$. The basic idea of the attack is first to search for a chain with r elements in α_l. If such a chain with a large enough number of elements is found, we can directly use its elements to create a set A' such that $(\text{alph}(\alpha_l))_{A'}$ is a permutation of A'. If such a chain is not found, we know, by using Lemma 4 that there is at least s pairwise incomparable elements in α_l and factorize the word $\alpha_l = \alpha_1\alpha_2$ in such a way that each of these pairwise incomparable elements appear in α_1 and α_2. Then we proceed to attack both α_1 and α_2 separately in the same manner.

While this "divide and conquer" idea is sound, it turns out that the upper bound of the required message length is rather high. To ensure that the attacker is able to find a set A' such that $|A'| \geq t$ and A' satisfies property (P1) the length of the word α_l should be at least $r_q t^{s_q}$ where s_q is $O(2^{2q-1})$ and r_q is $O(2^{2^{2q-3}})$. Effectively, this means that when creating large collisions, the length of the word is triple exponential in respect to q. This of course raises the question: could this upper bound be lowered?

In the article [20] the problem was studied even further from the point of view of word combinatorics. The proof of the following theorem offered a new and much lower upper bound for the required message length.

Theorem 1. For all positive integers t and q there exists a (minimal) positive integer $N(t,q)$ such that the following is true. Let α be a word for which $|\text{alph}(\alpha)| \geq N(t,q)$

and $|\alpha|_a \leq q$ for each $a \in \mathrm{alph}(\alpha)$. There then exists $A \subseteq \mathrm{alph}(\alpha)$ with $|A| = t$ and $p \in \{1,2,\ldots,q\}$ as well as words $\alpha_1, \alpha_2, \ldots, \alpha_p$ such that $\alpha = \alpha_1\alpha_2\cdots\alpha_p$ and for all $i \in \{1,2,\ldots,p\}$, the word $(\alpha_i)_A$ is a permutation of A.

Proof. We proceed by induction on q.

Consider first the case $q = 1$. Assume that $t \in \mathbb{N}_+$ and α is a word such that $|\alpha| \geq t$ and $|\alpha|_a = 1$ for each $a \in \mathrm{alph}(\alpha)$. Choosing $p = 1$ and letting $A \subseteq \mathrm{alph}(\alpha)$ be any such set that $|A| = t$, we note that $(\alpha)_A$ is a permutation of A. Thus $N(t,1)$ exists and is less than or equal to t. Trivially $N(t,1) \geq t$, so we conclude that $N(t,1) = t$.

Let us now turn to the general case. Let $q \geq 2$ and α be a word such that $|\mathrm{alph}(\alpha)| \geq N(t^2 - t + 1, q-1)$ and $|\alpha|_a \leq q$ for all $a \in \mathrm{alph}(\alpha)$.

Suppose that β is the word achieved from α when, for each $a \in \mathrm{alph}(\alpha)$ such that $|\alpha|_a = q$, the q^{th} occurrence of the symbol a is erased. Since $|\mathrm{alph}(\beta)| = |\mathrm{alph}(\alpha)| \geq N(t^2 - t + 1, q-1)$, there exist, by the induction hypothesis, $B \subseteq \mathrm{alph}(\alpha)$, $|B| = t^2 - t + 1$, $p \in \{1,2,\ldots,q-1\}$, and words $\beta_1, \beta_2, \ldots, \beta_p$ such that $\beta = \beta_1\beta_2\cdots\beta_p$ and for each $i \in \{1,2,\ldots,p\}$, the word $(\beta_i)_B$ is a permutation of B. Now $B \subseteq \mathrm{alph}(\beta_i) \subseteq \mathrm{alph}(\beta)$ for $i = 1,2,\ldots,p$.

Let $\alpha_1, \alpha_2, \ldots, \alpha_p$ be words such that $\alpha = \alpha_1\alpha_2\cdots\alpha_p$ and β_i is a subword of α_i for $i = 1,2,\ldots,p$ such that for each $i \in \{1,2,\ldots,p-1\}$ and $b \in B$ we have $|\alpha_i|_b = |\beta_i|_b$. Since β is a subword of α and β_p contains an occurrence of b for each $b \in B$, the words $\alpha_1, \alpha_2, \ldots, \alpha_p$ clearly exist.

Note that, by the facts above, we have $\pi_B(\beta_i) = \pi_B(\alpha_i)$ for $i = 1,2,\ldots,p-1$.

Claim 1. Let $b \in B$. If $|\alpha|_b < q$, then $|\alpha_i|_b = |\beta_i|_b$ for $i = 1,2,\ldots,p$. If $|\alpha|_b = q$, then $|\alpha_i|_b = |\beta_i|_b$ for $i = 1,2,\ldots,p-1$ and $|\alpha_p|_b = |\beta_p|_b + 1$.

Proof. By the definition of the words β and $\alpha_1, \alpha_2, \ldots, \alpha_p$, the following hold:

$$(1) \qquad |\alpha_i|_b = |\beta_i|_b \text{ for } i = 1,2,\ldots p-1 \text{ and } |\alpha_p|_b \geq |\beta_p|_b$$

$$(2) \qquad |\beta|_b + 1 = (\Sigma_{i=1}^{p}|\beta_i|_b) + 1 \geq \Sigma_{i=1}^{p}|\alpha_i|_b = |\alpha|_b \geq |\beta|_b$$

If $|\alpha|_b < q$, then $|\alpha|_b = |\beta|_b$ and, by (1) and (2), the equality $|\alpha_p|_b = |\beta_p|_b$ holds. If $|\alpha|_b = q$, then $|\alpha|_b = |\beta|_b + 1$ and, again by (1) and (2), we have $|\alpha_p|_b = |\beta_p|_b + 1$. $\square$

Assume, without loss of generality that $\pi_B(\beta_p) = a_1^{d_1}a_2^{d_2}\cdots a_{t^2-t+1}^{d_{t^2-t+1}}$, where $d_j \in \mathbb{N}_+$ for $j = 1,2,\ldots,t^2-t+1$ and $a_1, a_2, \ldots, a_{t^2-t+1}$ are the $t^2 - t + 1$ (pairwise distinct) symbols of B. Write $\pi_B(\beta_p)$ in the form $\pi_B(\beta_p) = \gamma_1\gamma_2\cdots\gamma_t$, where $\gamma_i =$

$a_{(i-1)t+1}^{d_{(i-1)t+1}} a_{(i-1)t+2}^{d_{(i-1)t+2}} \cdots a_{i\cdot t}^{d_{i\cdot t}}$ for $i = 1, 2, \ldots, t-1$ and $\gamma_t = a_{(t-1)t+1}^{d_{(t-1)t+1}}$. Then $|\mathrm{alph}(\gamma_i)| = t$ for $i = 1, 2, \ldots t-1$ and $|\mathrm{alph}(\gamma_t)| = 1$. Let $\delta_1, \delta_2, \ldots, \delta_t$ be words such that $\alpha_p = \delta_1 \delta_2 \ldots \delta_t$, γ_i is a subword of δ_i and $\mathrm{alph}(\delta_i) \cap \mathrm{alph}(\gamma_{i+1}\gamma_{i+2} \cdots \gamma_t) = \emptyset$ for $i = 1, 2, \ldots, t-1$. Since, by the definition of β, the d_i^{th} occurrence of a_i in α_p lies before the first occurrence of a_{i+1} for each $i \in \{1, 2, \ldots, t^2 - t\}$, the words $\delta_1, \delta_2, \ldots, \delta_t$ certainly can be found.

Claim 2. Let $i \in \{1, 2, \ldots, t-1\}$ and $a \in \mathrm{alph}(\gamma_i)$. Then $|\delta_{i+1}\delta_{i+2} \cdots \delta_t|_a \leq 1$.

Proof. Suppose that Claim 2 does not hold, i.e., $|\delta_{i+1}\delta_{i+2} \cdots \delta_t|_a \geq 2$. Then

$$|\alpha_p|_a = |\delta_i|_a + |\delta_{i+1}\delta_{i+2} \cdots \delta_t|_a \geq |\gamma_i|_a + 2 = |\beta_p|_a + 2$$

and we have a contradiction with Claim 1. □

For each $i \in \{1, 2, \ldots, t\}$, let $b_i \in \mathrm{alph}(\gamma_i)$ be a symbol that does not occur in $\delta_{i+1}\delta_{i+2} \cdots \delta_t$, if such a symbol exists. By the definition of $\delta_1, \delta_2, \ldots \delta_t$, such a b_i does not occur in $\delta_1 \delta_2 \cdots \delta_{i-1}$. Are we able to find t symbols $b_1, b_2, \ldots, b_t$? If so, choose $A = \{b_1, b_2, \ldots, b_t\}$ and note that $\alpha = \alpha_1 \alpha_2 \ldots \alpha_p$ and $(\alpha_i)_A$ is a permutation of A for $i = 1, 2, \ldots, p$.

Suppose that $j \in \{1, 2, ..., t-1\}$ is such that each symbol in $\mathrm{alph}(\gamma_j)$ occurs in $\delta_{j+1}\delta_{j+2} \cdots \delta_t$. By Claim 2, all these occurrences are the last ones of the respective symbol in α. Recall that $|\mathrm{alph}(\gamma_j)| = t$. Choose $A = \mathrm{alph}(\gamma_j)$, $\alpha'_p = \delta_1 \delta_2 \cdots \delta_j$, and $\alpha'_{p+1} = \delta_{j+1}\delta_{j+2} \cdots \delta_t$. Then $\alpha = \alpha_1 \alpha_2 \ldots \alpha_{p-1} \alpha'_p \alpha'_{p+1}$ and

$$(\alpha_1)_A, (\alpha_2)_A, \ldots, (\alpha_{p-1})_A, (\alpha'_p)_A, (\alpha'_{p+1})_A$$

are permutations of A. Since $p < q$, we have $p + 1 \leq q$. We may deduce that $N(t, q)$ exists and is less than or equal to $N(t^2 - t + 1, q - 1)$. The induction is thus extended and our proof is complete. □

This approach is much more efficient than the one presented in [14]. It allows the attacker to decrease the minimum length of α and so also the complexity of the attack significantly. Now $N(t, q)$ gives us the upper bound for the message length required to create a set A' such that A' satisfies property (P1) and $|A'| \geq t$. It is easy to see that $N(t, q) \leq t^{2^q}$. Thus the upperbound of the message length has been lowered from triple exponenetial to double exponential. In the next chapter, we will take a closer look at $N(t, q)$, lower the upper bound and create a lower bound for it.

6.4 Second Step: Using the Permutations

Once the attacker has found a set A' such that $|A'| = t$ and factorized $\alpha_l = \beta_1\beta_2\cdots\beta_p$ so that $(\beta_i)_{A'}$ is a permutation of A' she/he is ready to use this factorization to produce a set $A \subseteq A'$ that satisfies property (P2). This will result in a set A that will satisfy both properties (P1) and (P2).

In the general case, where α_l is $q-$bounded the situation is quite complex. In order to create A, the article [14] uses a result labeled as Lemma 1. The article [19] uses the following theorem that contains the same result.

Theorem 2. Let $k \in \mathbb{N}_+$ and A be a finite nonempty set such that k divides $|A|$. Furthermore, let $\{B_i\}_{i=1}^{k}$ and $\{C_j\}_{j=1}^{k}$ be partitions of A such that $|B_i| = |C_j|$ for $i, j = 1, 2, \ldots, k$. Then for each $x \in \mathbb{N}_+$ such that $|A| \geq k^3 \cdot x$, there exists a bijection $\sigma : \{1, 2, \ldots, k\} \to \{1, 2, \ldots, k\}$ for which $|B_i \cap C_{\sigma(i)}| \geq x$ for $i = 1, 2, \ldots, k$.

We omit the proof of this Theorem. Detailed proof can be found in [19]. For the original proof see [14].

The attacker proceeds by dividing β_p and β_{p-1} to k intervals and pairing them using Theorem 2. Then both β_{p-1} and β_{p-2} are divided to kn intervals and paired and so on. It is shown in [19] that, if $|A'| \geq k^{2q-3} \cdot n^{(q-1)^2}$, then the attacker will also be able to create a set $A \subseteq A'$ that satisfies also property (P2) and has $|A| = kn^{q-1}$.

This means that the articles [14, 19] prove that the attacker can create a set A that satisfies (P1) and (P2), by choosing the length of the message to be
$2^{2^{2^q-3}} k^{(2q-3)2^{2^q-1}} n^{(q-1)^2 2^{2^q-1}}$. This upper bound of the message length is lowered in [20] to $k^{(2q-3)2^q} n^{(q-1)^2 2^q}$.

6.5 Third Step: Completing the Attack

Assume now that the attacker has found a set A and a factorization $\alpha_l = \beta_1\beta_2\cdots\beta_p$ that satisfy properties (P1) and (P2) and thus first three steps of *NMCAS* are done. Now the attacker has to create a 2^k-collision. In order to do this we will need a couple of new results. These results can be found in [19]. They are included here for the sake of completeness. The following lemma allows the attacker to complete step 4 of the *NMCAS*.

In the following proofs, we assume that the attacker is able to find a collision in a set of 2^n messages with probability equal to one. If we assume that n is constant, this is strictly speaking not true, although the probability that this is not the case, is extremely small (the attacker requires 2^n+1 messages to find a collision certainly). The attacker can avoid this problem by simply starting the attack again, if it unsuccessful after 2^n-compression function calls or by replacing n with $n+1$ in the constructions. We have chosen to use n for the sake of notational simplicity.

Lemma 5. Let α be a word over the alphabet $\mathbb{N}_l$, r a positive integer and $a_1, a_2, \ldots, a_r$ in $\operatorname{alph}(\alpha)$ symbols such that $a_1 \prec_\alpha a_2 \prec_\alpha \ldots \prec_\alpha a_r$. Furthermore, let $\alpha = \alpha_1\alpha_2\cdots\alpha_r$ be a factorization of α such that for each $i \in \{1,2,\ldots,r\}$, all occurrences of the symbol a_i in α lie in α_i. Given an initial value $h_0 \in \{0,1\}^n$, we can, with probability equal to one, find message block sets $M_1, M_2, \ldots, M_l \subseteq \{0,1\}^m$ as well as values $h_1, h_2, \ldots, h_r \in \{0,1\}^n$ such that

(1) $M_b = \{iv\}$ for each $b \in \mathbb{N}_l \setminus A$, where $A = \{a_1, a_2, \ldots, a_r\}$;
(2) $M_{a_i} = \{u_i, u_i'\}$, where $u_i \neq u_i'$ for each $i \in \{1,2,\ldots,r\}$;
(3) for each $i \in \{1,2,\ldots,r\}$ the set $M = M_1M_2\cdots M_l$ is such that $\forall u,u' \in M$:

$$\begin{aligned} h_i &= f_{\alpha_i}(h_{i-1},u) = f_{\alpha_i}(h_{i-1},u') \\ &= f_{\alpha_1\alpha_2\cdots\alpha_i}(h_0,u) = f_{\alpha_1\alpha_2\cdots\alpha_i}(h_0,u'). \end{aligned}$$

Moreover, the expected number of queries on f needed for finding M is less than $2.5|\alpha|2^{\frac{n}{2}}$.

Proof. Let initially $M_i = \{iv\}$ for $i = 1,2,\ldots,l$. Proceed by induction on $i \in \{1,2,\ldots,r\}$. Suppose that, given the initial value $h_0 \in \{0,1\}^n$, we are, with probability equal to one, able to find message block sets $M_{a_j} = \{u_j, u_j'\}$, $j = 1,2,\ldots,i-1$, as well as values $h_1, h_2, \ldots, h_{i-1} \in \{0,1\}^n$ such that after updating $M := M_1M_2\cdots M_l$ the following holds: for each $j \in \{1,2,\ldots,i-1\}$ the set M satisfies property

$$\forall u,u' \in M : h_j = f_{\alpha_j}(h_{j-1},u) = f_{\alpha_j}(h_{j-1},u').$$

Furthermore, assume that the expected number of queries on f has been $2.5|\alpha_1\alpha_2 \cdots\alpha_{i-1}|2^{\frac{n}{2}}$.

Assume now that we have updated M. Create then a set $T = T_1T_2\cdots T_l$ such that $T_j = \{iv\}$ for all $j \neq a_i$ and T_{a_i} is a set of $2^{\frac{n}{2}}$ random message blocks. Among the messages, a 2-collision on f_{α_i} with initial value h_{i-1}, is tried to be found. Using results from subsection 2.7.2 it is easy to deduce that the expected number of times that the generation of the set T_{a_i} of $2^{\frac{n}{2}}$ random message blocks has to be repeated is less than 2.5. Thus the expected number of queries on f is less than $2.5|\alpha_i|2^{\frac{n}{2}}$ (one could lower the multiplier 2.5 here to $\check{a} \approx 2.2$).

Note two things in the construction of the collision on f_{α_i}:

(i) only message blocks from those sets T_j for which $j \in \mathrm{alph}(\alpha_i)$ are used; and
(ii) for each $j \in \mathrm{alph}(\alpha_i)$, if $j \neq a_i$, then $M_j = \{iv\}$.

Let $x, y \in T$, $x \neq y$, be such that $f_{\alpha_i}(h_{i-1}, x) = f_{\alpha_i}(h_{i-1}, y)$. Let $x = x_1 x_2 \cdots x_l$ and $y = y_1 y_2 \cdots y_l$, where $x_j, y_j \in \{0,1\}^m$ for $j = 1, 2, \ldots, l$. By properties (i) and (ii) above, $x_{a_i} \neq y_{a_i}$. Choose $u_i = x_{a_i}$ and $u'_i = y_{a_i}$. Let $M_{a_i} = \{u_i, u'_i\}$ and $h_i = f_{\alpha_i}(h_{i-1}, x)$. Update $M = M_1 M_2 \cdots M_l$ and deduce that $\forall u, u' \in M$:

$$\begin{aligned} h_i &= f_{\alpha_i}(h_{i-1}, u) = f_{\alpha_i}(h_{i-1}, u') \\ &= f_{\alpha_1 \alpha_2 \cdots \alpha_i}(h_0, u) = f_{\alpha_1 \alpha_2 \cdots \alpha_i}(h_0, u') \ . \end{aligned}$$

The expected number of queries on f is altogether $2.5|\alpha_1 \alpha_2 \cdots \alpha_i| 2^{\frac{n}{2}}$. The induction is now extended. $\square$

Assume now that the attacker has created a set A and a factorization $\alpha_l = \beta_1 \beta_2 \cdots \beta_p$ that satisfies properties (P1) and (P2). The attacker will use Lemma 5 at the beginning of the attack against the word β_1. This Lemma allows the attacker to create a set $M = M_1 M_2 \cdots M_l$ in such a way that M_a is some fixed message block (referred as iv) for all $a \notin A$, while for all $a \in A$ there are two possible message blocks for M_a. In other words, the attacker searches $2-$collisions for all kn^{p-1} elements of A.

The attacker has now created an advantage that she/he can use later. This advantage is that the attacker knows that $f_{\beta_1}(h_0, u) = f_{\beta_1}(h_0, u')$ for all $u, u' \in M$ while at the same time the attacker can freely choose the M_a between two different options for all $a \in A$. This will allow the attacker to mount an attack against β_2 and so on. Before continuing however, we need the following result.

Lemma 6. Let α be a word over the alphabet $\mathbb{N}_l$, d and r positive integers, $A \subseteq \mathrm{alph}(\alpha)$ a set of cardinality $|A| = dnr$, and $\alpha = \beta_1 \beta_2 \cdots \beta_{nr} \gamma_1 \gamma_2 \cdots \gamma_r$ a factorization of α with the following properties.

(1) $A \subseteq \mathrm{alph}(\beta) \cap \mathrm{alph}(\gamma)$ where $\beta = \beta_1 \beta_2 \cdots \beta_{nr}$ and $\gamma = \gamma_1 \gamma_2 \cdots \gamma_r$;
(2) $|\mathrm{alph}(\beta_i) \cap A| = d$ for $i = 1, 2, \ldots, nr$, and $|\mathrm{alph}(\gamma_j) \cap A| = nd$ for $j = 1, 2, \ldots, r$; and
(3) for each $i \in \{1, 2, \ldots, nr\}$ there exists $j \in \{1, 2, \ldots, r\}$ such that $\mathrm{alph}(\beta_i) \subseteq \mathrm{alph}(\gamma_j)$.

Furthermore, let $u_1, u'_1, u_2, u'_2, \ldots, u_{nr}, u'_{nr} \in \{0,1\}^{ml}$ be messages and $h_0, h_1, \ldots\ h_{nr}$ in $\{0,1\}^n$ be values such that for each $i \in \{1, 2, \ldots, nr\}$:

(4) $\forall b \in \mathbb{N}_l \setminus A : u_i(b) = u'_i(b) = iv$; and

(5) $u_i(\beta_i) \neq u'_i(\beta_i)$ and $h_i = f_{\beta_i}(h_{i-1}, u_i) = f_{\beta_i}(h_{i-1}, u'_i)$.

The set S of all messages $u \in \{0,1\}^{ml}$, such that for each $b \in \mathbb{N}_l \setminus A$: $u(b) = iv$ and for each $i \in \{1,2,\dots,nr\}$: $u(\beta_i) \in \{u_i(\beta_i), u'_i(\beta_i)\}$ is then well-defined and satisfies for each $i \in \{1,2,\dots,nr\}$ and $u \in S$ the equality $h_i = f_{\beta_i}(h_{i-1}, u)$. Moreover we can, with probability equal to one, find messages $v_1, v'_1, v_2, v'_2, \dots, v_r, v'_r$ in S and values $h'_0, h'_1, \dots h'_r$, $h'_0 = h_{nr}$, such that for each $j \in \{1,2,\dots,r\}$:

(6) $v_j(\gamma_j) \neq v'_j(\gamma_j)$ and $h'_j = f_{\gamma_j}(h'_{j-1}, v_j) = f_{\gamma_j}(h'_{j-1}, v'_j)$.

The expected number of queries on f, needed to carry out the task, is less than $2.5|\gamma|2^{\frac{n}{2}}$. Finally, the set T of all messages $v \in \{0,1\}^{ml}$ such that for each $b \in \mathbb{N}_l \setminus A$: $v(b) = iv$ and for each $j \in \{1,2,\dots,r\}$: $v(\gamma_j) \in \{v_j(\gamma_j), v'_j(\gamma_j)\}$ is then a well-defined subset of S and forms a 2^r-collision on f_α with the initial value h_0.

Proof. Note first that, since $|A| = dnr$, $A \subseteq \text{alph}(\beta)$, and $|\text{alph}(\beta_i) \cap A| = d$ for each $i \in \{1,2,\dots,nr\}$, the indexed family of sets $\{\text{alph}(\beta_i) \cap A\}_{i=1}^{nr}$ forms a partition of A. With analogous reasoning, $\{\text{alph}(\gamma_j) \cap A\}_{j=1}^{r}$ is a partition of A, too.

Now, let $x_i \in \{u_i, u'_i\}$ for $i = 1,2,\dots,nr$. Consider the sequence $x_1(\beta_1)$, $x_2(\beta_2)$,...,$x_{nr}(\beta_{nr})$. Define $t_1, t_2, \dots, t_l \in \{0,1\}^m$ as follows. For each $b \in \mathbb{N}_l \setminus A$, let $t_b = iv$. For each $a \in A$ and $i \in \{1,2,\dots,nr\}$, if $a \in \text{alph}(\beta_i) \cap A$, then $t_a = x_i(a)$. Since $\{\text{alph}(\beta_i) \cap A\}_{i=1}^{nr}$ is a partition of A, the message block t_a is uniquely determined. Thus the sequence $x_1(\beta_1)$, $x_2(\beta_2)$,...,$x_{nr}(\beta_{nr})$ uniquely defines the message $t_1 t_2 \cdots t_l$. We deduce that the set S is well-defined.

Consider now the sets $\{u(\gamma_1) | u \in S\}$, $\{u(\gamma_2) | u \in S\}$,..., $\{u(\gamma_r) | u \in S\}$. Since $\{\text{alph}(\gamma_j) \cap A\}_{j=1}^{r}$ is a partition of A and the property (3) holds, the cardinality of the set $\{u(\gamma_j) | u \in S\}$ is 2^n for each $j \in \{1,2,\dots,r\}$. Furthermore, since $\gamma = \gamma_1 \gamma_2 \cdots \gamma_r$, the equality

$$\{u(\gamma) | u \in S\} = \{u(\gamma_1) | u \in S\}\{u(\gamma_2) | u \in S\} \cdots \{u(\gamma_r) | u \in S\}$$

holds, so the cardinality of the set $\{u(\gamma) | u \in S\}$ is 2^{nr}.

Let $u \in S$. Then

$$f_\beta(h_0, u) = f^+(h_0, u(\beta)) = f^+(h_0, u(\beta_1) u(\beta_2) \cdots u(\beta_{nr})) = h_{nr}.$$

Thus S is a 2^{nr}-collision on f_β with the initial value h_0.

Now set, $h'_0 = h_{nr}$. Continue by induction; assume that k is in $\{1,2, \ldots,r\}$ and with the probability equal to one, messages $v_1,v'_1,v_2,v'_2,\ldots,v_k,v'_k$ in S and values $h'_1,h'_2,\ldots,h'_k$ in $\{0,1\}^n$ have been found such that for each $j \in \{1,2,\ldots,k\}$

$$v_j(\gamma_j) \neq v'_j(\gamma_j) \text{ and } h'_j = f_{\gamma_j}(h'_{j-1},v_j) = f_{\gamma_j}(h'_{j-1},v'_j) \,.$$

Furthermore, the expected number of queries on f is $2.5|\gamma_1\gamma_2\cdots\gamma_k|2^{\frac{n}{2}}$. Since, for each $u \in S$, the equality

$$f_{\gamma_{k+1}}(h'_k,u) = f^+(h'_k,u(\gamma_{k+1}))$$

holds and the cardinality of the set $\{u(\gamma_{k+1})|u \in S\}$ is 2^n, we can, choosing message sets of cardinality $2^{\frac{n}{2}}$ randomly from the set S, find messages v_{k+1},v'_{k+1} in $\{0,1\}^{ml}$ and a value h'_{k+1} in $\{0,1\}^n$ such that $v_{k+1}(\gamma_{k+1}) \neq v'_{k+1}(\gamma_{k+1})$ and $h'_{k+1} = f_{\gamma_{k+1}}(h'_k,v_{k+1}) = f_{\gamma_{k+1}}(h'_k,v'_{k+1})$. The expected number of queries on f is less than $2.5|\gamma_{k+1}|2^{\frac{n}{2}}$.

The induction is now extended and messages v_1,v'_1, v_2,v'_2 ,..., v_r,v'_r in S and values $h'_0,h'_1,\ldots,h'_r$ in $\{0,1\}^n$ satisfying (6) found with the expected number $2.5|\gamma|2^{\frac{n}{2}}$ of queries on f. The task is successful with probability equal to one.

Reasoning as with the set S and noting that v_j,v'_j are in S for each $j \in \{1,2,\ldots,r\}$, it is straightforward to see that T is a well-defined subset of S. Since $v_j(\gamma_j) \neq v'_j(\gamma_j)$ for each $j \in \{1,2,\ldots,r\}$ and $v_j(b) = v'_j(b)$ for all $b \in \mathbb{N}_l \setminus A$, the cardinality of T is 2^r. Certainly $f_\alpha(h_0,u) = h'_r$ for each $u \in T$. The proof is now complete. $\square$

This lemma allows the attacker to proceed in the attack step by step. First the attacker proceeds from β_1 to β_2, then from β_2 to β_3 and so on. The following theorem combines the results of the two previous lemmata thus creating a tool that proves that the attack works and allows the attacker to complete the step 5 of *NMCAS*.

Theorem 3. Let α be a word over the alphabet $\mathbb{N}_l$, k and p positive integers, A a subset of the alphabet alph(α) of cardinality $|A| = n^{p-1}k$, and $\alpha = \alpha_1\alpha_2\cdots\alpha_p$ a factorization of α such that for each $i \in \{1,2,\ldots,p\}$, the elements of A form a chain in the partially ordered set (alph$(\alpha_i),\prec_{\alpha_i}$) (i.e., the elements of A are independent with respect to $\prec_{\alpha_i}$). Furthermore, assume that for each $i \in \{1,2,\ldots,p\}$, there exists a factorization $\alpha_i = \alpha_{i1}\alpha_{i2}\cdots\alpha_{i,n^{p-i}k}$ of the word α_i such that the following conditions are satisfied.

(1) $|\text{alph}(\alpha_{ij}) \cap A| = n^{i-1}$ for each $i \in \{1,2,\ldots,p\}$ and $j \in \{1,2,\ldots,n^{p-i}k\}$; and
(2) for each $i \in \{1,2,\ldots,p\}$ and $j \in \{1,2,\ldots,n^{p-i}r\}$ there exists $r \in \{1,2,\ldots, n^{p-i-1}k\}$ such that alph$(\alpha_{ij}) \cap A$ is a subset of alph$(\alpha_{i+1,r}) \cap A$.

Then, given an initial value $h_0 \in \{0,1\}^n$ we can, with probability equal to one, find a 2^k-collision on f_α with the messages of length l message blocks. Moreover, the expected number of queries on f_α needed to carry out the task, is $2.5|\alpha|2^{\frac{n}{2}}$.

Proof. We first apply Lemma 5 to generate a $2^{n^{p-1}k}$-collision B_1 on f_{α_1} and then, by using Lemma 6 repeatedly, show that there exists a $2^{n^{p-i}k}$-collision B_i on $f_{\alpha_1\alpha_2\cdots\alpha_i}$ for $i = 2,3,\ldots,p$ such that $B_1 \supsetneqq B_2 \supsetneqq \cdots \supsetneqq B_p$.

Choose in Lemma 5 parameters as follows: α is equal to α_1 and r is equal to $n^{p-1}k$. Let $A = \{a_1, a_2, \ldots, a_{n^{p-1}k}\}$ and $a_1 \prec_{\alpha_1} a_2 \prec_{\alpha_1} \ldots \prec_{\alpha_1} a_{n^{p-1}k}$. By property (1), these assumptions can be made.

Then $\alpha_1 = \alpha_{11}\alpha_{12}\cdots\alpha_{1,n^{p-1}k}$ is a factorization of α_1 such that for each $j \in \{1,2,\ldots,n^{p-1}k\}$, all occurrences of the symbol a_j in α_1 lie in α_{1j}. Let $h_0 \in \{0,1\}^n$ be given. Applying Lemma 5 one can, with probability equal to one and with the expected number $2.5|\alpha_1|2^{\frac{n}{2}}$ of queries on f, find message block sets $M_1, M_2, \ldots, M_l \subseteq \{0,1\}^m$ as well as values $h_1, h_2, \ldots, h_{n^{p-1}k}$ such that

(a) $M_b = \{iv\}$ for each $b \in \mathbb{N}_l \setminus A$;
(b) $M_{a_i} = \{w_i, w_i'\}$, where $w_i \neq w_i'$ for each $i \in \{1,2,\ldots,n^{p-1}k\}$; and
(c) for each $i \in \{1,2,\ldots,n^{p-1}k\}$, the set $M = M_1M_2\cdots M_l$ is such that $\forall w, w' \in M$:

$$h_i = f_{\alpha_{1i}}(h_{i-1}, w) = f_{\alpha_{1i}}(h_{i-1}, w') = f_{\alpha_{11}\alpha_{12}\cdots\alpha_{1i}}(h_0, w) = f_{\alpha_{11}\alpha_{12}\cdots\alpha_{1i}}(h_0, w').$$

For all $i \in \{1,2,\ldots,n^{p-1}k\}$, let messages $u_i, u_i' \in \{0,1\}^{ml}$ be defined as follows. For each $b \in \mathbb{N}_l \setminus A$, let $u_i(b) = u_i'(b) = iv$; for each $a \in A$, let $u_i(a) = u_i'(a) = w_i$, if $a = a_j$ for some $j \in \{1,2,\ldots,n^{p-1}k\}$, $j \neq i$, and $u_i(a) = w_i$ and $u_i'(a) = w_i'$ if $a = a_i$. The set $B_1 = M$ is a $2^{n^{p-1}k}$-collision (of complexity $2.5|\alpha_1|2^{\frac{n}{2}}$) on f_{α_1} with an initial value h_0.

Choose the parameters of Lemma 6 as follows. Let β be α_1, γ be α_2 and r be $n^{p-2}k$. Let d be equal to 1, β_i equal to α_{1i} for $i = 1,2,\ldots,n^{p-1}k$, and γ_j equal to α_{2j} for $j = 1,2,\ldots,n^{p-2}k$. Then the assumptions of Theorem 3 for α_1 and α_2 imply that all the assumptions of Lemma 6 (with parameters chosen as above) are valid. Thus we can, with a probability equal to one and expected number $2.5|\alpha_2|2^{\frac{n}{2}}$ of queries on f, find messages v_1, v_1', v_2, v_2', $\ldots$, $v_{n^{p-2}r}, v'_{n^{p-2}r}$ in $\{0,1\}^{ml}$ and values $h_0', h_1', \ldots, h'_{n^{p-2}r}$ in $\{0,1\}^n$, $h_0' = h_{n^{p-1}r}$, such that for each $j \in \{1,2,\ldots,n^{p-2}r\}$, $\forall b \in \mathbb{N}_l \setminus A : v_j(b) = v_j'(b) = iv$ and $v_j(\alpha_{2j}) \neq v_j'(\alpha_{2j})$ and $h_j' = f_{\alpha_{2j}}(h'_{j-1}, v_j) = f_{\alpha_{2j}}(h'_{j-1}, v_j')$. The set S of Lemma 6 is clearly our set $B_1 = M$. Choose B_2 to be the set T guaranteed by Lemma 6. Then $B_2 \subsetneqq B_1$ is a $2^{n^{p-2}k}$-collision on $f_{\alpha_1\alpha_2}$ with an initial value h_0.

Continue by induction and let $r \in \{2,3,\ldots,p-1\}$. Let x_1,x'_1, x_2,x'_2, $\ldots$, $x_{n^{p-r}k}$, $x'_{n^{p-r}k}$ in $\{0,1\}^{ml}$ and values $d_0,d_1,\ldots,d_{n^{p-r}k}$ in $\{0,1\}^n$ be such that for each $i \in \{1,2,\ldots,n^{p-r}k\}$, $\forall b \in \mathbb{N}_l \setminus A : x_i(b) = x'_i(b) = iv$ and $x_i(\alpha_{ri}) \neq x'_i(\alpha_{ri})$ and $d_i = f_{\alpha_{ri}}(d_{i-1},x_i) = f_{\alpha_{ri}}(d_{i-1},x'_i)$. Let B_r be the set of all messages $u \in \{0,1\}^{ml}$ such that for each $b \in \mathbb{N}_l \setminus A$: $u(b) = iv$ and for each $j \in \{1,2,\ldots,n^{p-r}k\}$: $u(\alpha_{rj})$ is in $\{x(\alpha_{rj}),x'(\alpha_{rj})\}$. Suppose that B_r is a subset of B_{r-1} and that B_r is a $2^{n^{p-r}k}$-collision on $f_{\alpha_1\alpha_2\ldots\alpha_r}$ with an initial value h_0. Choose the parameters of Lemma 6 as follows. Let d be equal to n^{r-1}, β be equal to α_r, β_i be equal to α_{ri} for $i = 1,2,\ldots,n^{p-r}k$, and γ_j be equal to $\alpha_{r+1,j}$ for $j = 1,2,\ldots,n^{p-r-1}k$. By the assumptions of Theorem 3, all the assumptions of Lemma 6 are valid (with the chosen parameter values). Lemma 6 implies that one may, with a probability equal to one and expected number $2.5|\alpha_{r+1}|2^{\frac{n}{2}}$ of queries on f, find messages y_1,y'_1, y_2,y'_2, $\ldots$, $y_{n^{p-r-1}k},y'_{n^{p-r-1}k}$ in $\{0,1\}^{ml}$ and values $d'_0,d'_1,\ldots,d'_{n^{p-r-1}k}$ in $\{0,1\}^n$, $d'_0 = d_{n^{p-r}k}$, such that for each $j \in \{1,2,\ldots,n^{p-r-1}k\}$, $\forall b \in \mathbb{N}_l \setminus A : y_j(b) = y'_j(b) = iv$ and $y_j(\alpha_{r+1,j}) \neq y'_j(\alpha_{r+1,j})$ and $d'_j = f_{\alpha_{r+1,j}}(d'_{j-1},y_j) = f_{\alpha_{r+1,j}}(d'_{j-1},y'_j)$. The set T of all messages y in $\{0,1\}^{ml}$ such that for each $b \in \mathbb{N}_l \setminus A$: $y(b) = iv$ and for each $j \in \{1,2,\ldots,n^{p-r-1}k\}$: $y(\alpha_{r+1,j})$ is in $\{y_j(\alpha_{r+1,j}),y'_j(\alpha_{r+1,j})\}$ is then a well-defined subset of B_r and forms a $2^{n^{p-r-1}k}$-collision on $f_{\alpha_r\alpha_{r+1}}$ with an initial value d_0. By the induction assumption, T is a $2^{n^{p-r-1}k}$-collision on $f_{\alpha_1\alpha_2\cdots\alpha_{r+1}}$ with an initial value h_0. Choose $B_{r+1} = T$ and the induction is extended. We deduce that we can, with probability equal to one, find a 2^k-collision on f_α with an initial value h_0. The expected number of queries on f is altogether $2.5|\alpha|2^{\frac{n}{2}}$. $\square$

Since against the generalized $2-$bounded hash function we had to choose the message length to be at least k^2n, the total complexity of the 2^k-collision attack against it will be approximately $2.5 \cdot 2 \cdot k^2 \cdot n \cdot 2^{\frac{n}{2}}$ according to our complexity analysis. The complexity bound offered in [30] is $O(k^2 \cdot (lnk) \cdot (n + ln(ln2k)) \cdot 2^{n/2})$.

Since α is $q-$bounded $|\alpha| \leq q|\text{alph}(\alpha)|$. This means that in [14, 19] the total complexity of the attack will be less than

$$2.5 \cdot q \cdot 2^{2^{2q-3}} k^{(2q-3)2^{2q-1}} n^{(q-1)^2 2^{2q-1}} 2^{\frac{n}{2}}.$$

This is enough to imply that, given $q,k \in \mathbb{N}_+$ for sufficiently large n, for all q-bounded generalized iterated hash functions, the complexity of creating a 2^k-collision is much lower than, the complexity required when using a random search. However, if we assume that for example $n = 256$, $k = 4$ and $q = 3$, the only guarantee we get is that the expected number of compression function calls is less than

$$2.5 \cdot 3 \cdot 2^{2^{2^3}-3} 4^{(2^3-3)2^{2^3}-1} 256^{(3-1)^2 2^{2^3}-1} 2^{\frac{256}{2}} = 7.5 \cdot 2^{8630}.$$

Since the complexity of the preimage attack is approximately 2^{256} this is not very reassuring.

As we have seen Theorem 1 (presented first in [20]) offers a more efficient way to create permutations and lowers the minimum length of the required word and thus also the complexity drops to

$$2.5 \cdot q \cdot k^{(2q-3)2^q} n^{(q-1)^2 2^q} 2^{\frac{n}{2}}.$$

However, even this better way to create multicollisions against generalized iterated hash functions still gives us huge complexity. If we once again assume that $n = 256$, $k = 4$ and $q = 3$, we get the upper bound of the attack to be $7.5 \cdot 2^{432}$. This bound is much better than the one offered by the previous version, but still clearly more than the complexity of finding a preimage for any given message.

Consider now a situation, where the inner state of the hash function is larger or equal to the message block size (in bits) i.e. $m \leq n$ in Definition 2. In this situation, the attacker can ensure that it is possible to find the required collisions in f_{α_1} of Theorem 3, by ensuring that each $\alpha_{1,j}$ contains at least i elements of A, where i is the smallest possible integer such that $im > n$. This increases the complexity of the attack with $i(kn^{p-1})^2$ in our complexity considerations. We omit the details of this consideration, since the next chapter offers us a better way to create multicollision attacks and handles this kind of situation more efficiently.

Remark 11. The complexity bounds are presented here in the same form as they have been presented in [14, 19, 21]. The results of subsection 2.7.2 would allow us to drop the 2.5 multiplier of the complexity to $\breve{a} \approx 2.22$.

7 Generalized Iterated Hash Functions II: Attacks Based on Nonuniform Words

We will now create a new way of attacking generalized $q-$bounded hash functions. Later we will take a look at the word combinatorial properties and results in a more general sense. These results are refined versions of the results in [24].

We begin by pointing out that property (P1) (see Section 6.3) can be somewhat loosened and the proof of the Theorem 3 will still remain valid. Let us define two new properties:

(Q1) α_l possesses a factorization $\alpha_l = \beta_1\beta_2\cdots\beta_p$, where $A \subseteq \mathrm{alph}(\beta_i)$ for $i = 1,2,\ldots,p$, $(\beta_1)_A$ is a permutation of A; and

(Q2) for any $i \in \{1,2,\ldots,p-1\}$, if $(\beta_i)_A = z_1 z_2\cdots z_{n^{p-i}k}$ is a factorization of $(\beta_i)_A$ such that $|\mathrm{alph}(z_j)| = n^{i-1}$ for $j = 1,2,\ldots n^{p-i}k$ and $(\beta_{i+1})_A = u_1u_2\cdots u_{n^{p-i-1}k}$ is a factorization of $(\beta_{i+1})_A$ such that $|\mathrm{alph}(u_j)| = n^i$ for $j = 1,2,\ldots,n^{p-i-1}k$, then for each $j_1 \in \{1,2,\ldots, n^{p-i}k\}$ there exists $j_2 \in \{1,2,\ldots, n^{p-i-1}k\}$ such that $\mathrm{alph}(z_{j_1}) \subseteq \mathrm{alph}(u_{j_2})$.

As before, property (Q2) means also that for each $j_2 \in \{1,2,\ldots, n^{p-i-1}k\}$ there exist exactly n integers $j_1 \in \{1,2,\ldots n^{p-i}k\}$ such that $\mathrm{alph}(z_{j_1}) \subseteq \mathrm{alph}(u_{j_2})$, while $\mathrm{alph}(z_{j_1}) \bigcap \mathrm{alph}(u_{j_3}) = \emptyset$ for all $j_3 \in \{1,2,\ldots, n^{p-i-1}k\}$, $j_3 \neq j_2$.

It is easy to see that the only difference between properties (P1), (P2) (see Section 6.2) and (Q1), (Q2) is that (Q1) does not force $(\beta_i)_A$ to be a permutation of A for $i \in \{2,3,\cdots,p\}$. In addition it is also easy to see that (Q1) and (Q2) mean that $|A| = kn^{p-1}$.

We are now able to formulate a new theorem based on Theorem 3.

Theorem 4. Let α_l be a word, $\mathrm{alph}(\alpha_l) = \mathbb{N}_l$, k and p positive integers and A a subset of the alphabet $\mathrm{alph}(\alpha_l)$ of cardinality $|A| = n^{p-1}k$. Assume furthermore that A and the factorization $\alpha_l = \beta_1\beta_2\cdots\beta_p$ satisfy properties (Q1) and (Q2). Then, given an initial value $h_0 \in \{0,1\}^n$ we can, with probability equal to one, find a 2^k-collision on f_{α_l} with the messages of length l message blocks. Moreover, the expected number of queries on f needed to carry out the attack is at most $2,5|\alpha|2^{\frac{n}{2}}$.

Proof. The proof of the theorem is analogous to the proof of Theorem 3. □

In the previous chapter, we created our attack by constructing a set A' that satisfied (P1) and then a set $A \subseteq A'$ satisfying (P2).

We will now abandon this approach and proceed to create a single tool that will allow us to create a set A and a factorization $\alpha_l = \beta_1\beta_2\cdots\beta_p$ that satisfies both (Q1) and (Q2) for large enough integer l. This goal will be finally reached, when we formulate Theorem 9. For this we need a large set of results concerning the so called uniform words that we will define next.

7.1 Uniform and Distinct Words

Given any word β, it seems intuitively evident that there exists a factorization $\beta = \gamma\delta$ of β such that either ratio $\frac{|\mathrm{alph}(\gamma)\cap\mathrm{alph}(\delta)|}{|\mathrm{alph}(\beta)|}$ is fairly large or both $\frac{|\mathrm{alph}(\gamma)\setminus\mathrm{alph}(\delta)|}{|\mathrm{alph}(\beta)|}$ and $\frac{|\mathrm{alph}(\delta)\setminus\mathrm{alph}(\gamma)|}{|\mathrm{alph}(\beta)|}$ are fairly large. We shall now formalize these two properties and study their relationship in detail.

Let thus α be a word, A a subset of $\mathrm{alph}(\alpha)$, and s an integer. We say that the word α is (A,s)*-uniform*, if there exists a factorization $\alpha = \alpha_1\alpha_2$ of α such that $|\mathrm{alph}(\alpha_1)\cap\mathrm{alph}(\alpha_2)\cap A| \geq s$. Given an (A,s)-uniform word α, any word β such that α is a subword of β is (A,s)-uniform as well. Moreover, α itself is (B,d)-uniform for each set B and integer d such that $A \subseteq B \subseteq \mathrm{alph}(\alpha)$ and $d \leq s$.

A dual concept to uniformity is that of distinction. The word α is (A,s)*-distinct* if there exists a factorization $\alpha = \alpha_1\alpha_2$ of α such that $|[\mathrm{alph}(\alpha_1)\setminus\mathrm{alph}(\alpha_2)]\cap A| \geq s$ and $|[\mathrm{alph}(\alpha_2)\setminus\mathrm{alph}(\alpha_1)]\cap A| \geq s$. Given an (A,s)-distinct word α any subword β of α such that $A \subseteq \mathrm{alph}(\beta)$ is also (A,s)-distinct. Moreover, the word α is (B,d)-distinct always, when $A \subseteq B \subseteq \mathrm{alph}(\alpha)$ and $d \leq s$.

Note that, for notational reasons, uniformity and distinction are defined also on the negative values of the parameter s. The following theorem tells us, how the two concepts are connected.

Theorem 5. Let α be a word, $A \subseteq \mathrm{alph}(\alpha)$, and $s \in \mathbb{Z}$.

(a) If α is not (A,s)-uniform, then α is $(A, \lfloor\frac{|A|-s+1}{2}\rfloor)$-distinct.

(b) If α is not (A,s)-distinct, then α is $(A, |A|-2s+1)$-uniform.

Proof. If $s \leq 0$ or $s \geq |A|$, then both claims certainly hold.

Suppose thus that s is an integer such that $0 < s < |A|$ and that α is a word, which is not (A,s)-uniform. Let $\alpha = \beta\gamma$ be a factorization of α such that β is the longest prefix of α for which $|[\mathrm{alph}(\beta)\setminus\mathrm{alph}(\gamma)]\cap A|\le\lfloor\frac{|A|-s+1}{2}\rfloor$. Since we assumed that s is a positive integer, such a factorization always can be found. Suppose that $|[\mathrm{alph}(\beta)\setminus\mathrm{alph}(\gamma)]\cap A|<\lfloor\frac{|A|-s+1}{2}\rfloor$. Then certainly the word γ is nonempty; let $\gamma=c\rho$ where $c\in\mathrm{alph}(\gamma)$. If $c\notin A$ or $c\in\mathrm{alph}(\rho)$, then $|[\mathrm{alph}(\beta)\setminus\mathrm{alph}(\gamma)]\cap A|=|[\mathrm{alph}(\beta c)\setminus\mathrm{alph}(\rho)]\cap A|$ and we are in contradiction with the definition of β. Suppose thus that $c\in A$ and $c\notin\mathrm{alph}(\rho)$. Then

$$\begin{aligned}|[\mathrm{alph}(\beta c)\setminus\mathrm{alph}(\rho)]\cap A| &= |[\mathrm{alph}(\beta)\setminus\mathrm{alph}(\gamma)]\cap A|+1\\ &\le\lfloor\tfrac{|A|-s+1}{2}\rfloor\end{aligned}$$

and we are still in contradiction with the length property of β. Thus $|[\mathrm{alph}(\beta)\setminus\mathrm{alph}(\gamma)]\cap A|=\lfloor\frac{|A|-s+1}{2}\rfloor$. Moreover, since α is not (A,s)-uniform, the following holds:

$$\begin{aligned}|[\mathrm{alph}(\gamma)\setminus\mathrm{alph}(\beta)]\cap A| &= |A|-|[\mathrm{alph}(\beta)\setminus\mathrm{alph}(\gamma)]\cap A|-|\mathrm{alph}(\beta)\cap\mathrm{alph}(\gamma)\cap A|\\ &\ge |A|-\lfloor\tfrac{|A|-s+1}{2}\rfloor-s+1=\lceil\tfrac{|A|-s+1}{2}\rceil\\ &\ge\tfrac{|A|-s+1}{2}\ge\lfloor\tfrac{|A|-s+1}{2}\rfloor\ .\end{aligned}$$

We deduce that α is $(A,\lfloor\frac{|A|-s+1}{2}\rfloor)$-distinct.

Consider then claim (b). We could use the result of (a) and prove (b) by contraposition, but we prefer direct verification. If $s>\lceil\frac{|A|}{2}\rceil$, then (b) certainly holds. Assume thus that s is an integer such that $0<s\le\lceil\frac{|A|}{2}\rceil$ and that the word α is not (A,s)-distinct. Let α_1 be the longest prefix of α such that $|[\mathrm{alph}(\alpha_1)\setminus\mathrm{alph}(\alpha_2)]\cap A|<s$ where $\alpha=\alpha_1\alpha_2$. Since $s\in\mathbb{N}_+$, such an α_1 can always be found.

We first show that $|[\mathrm{alph}(\alpha_2)\setminus\mathrm{alph}(\alpha_1)]\cap A|\le s$. Assume the contrary, i.e., $|[\mathrm{alph}(\alpha_2)\setminus\mathrm{alph}(\alpha_1)]\cap A|>s$. Let $\alpha_2=a\beta_2$ where $a\in\mathrm{alph}(\alpha)$ and $\beta_2\in(\mathrm{alph}(\alpha))^*$. Denote $\beta_1=\alpha_1 a$. By the definition of α_1, the inequality $|[\mathrm{alph}(\beta_1)\setminus\mathrm{alph}(\beta_2)]\cap A|\ge s$ must hold. On the other hand,

$$|[\mathrm{alph}(\beta_2)\setminus\mathrm{alph}(\beta_1)]\cap A|\ge|[\mathrm{alph}(\alpha_2)\setminus\mathrm{alph}(\alpha_1)]\cap A|-1\ ,$$

so $|[\mathrm{alph}(\beta_2)\setminus\mathrm{alph}(\beta_1)]\cap A|\ge s$. This means that α is (A,s)-distinct, a contradiction.

Thus, let $|[\mathrm{alph}(\alpha_2)\setminus\mathrm{alph}(\alpha_1)]\cap A|\le s$. This implies that

$$\begin{aligned}|[\mathrm{alph}(\alpha_1)\cap\mathrm{alph}(\alpha_2)]\cap A| &= ||\mathrm{alph}(\alpha)\cap A|-|[\mathrm{alph}(\alpha_1)\setminus\mathrm{alph}(\alpha_2)]\cap A|\\ &\quad -|[\mathrm{alph}(\alpha_2)\setminus\mathrm{alph}(\alpha_1)]\cap A|\\ &> |A|-2s\ .\end{aligned}$$

Since $|[\mathrm{alph}(\alpha_1)\cap\mathrm{alph}(\alpha_2)]\cap A|\ge|A|-2s+1$, the word α is $(A,|A|-2s+1)$-uniform. The proof is now complete. $\square$

Remark 12. Let $r, s \in \mathbb{N}_+$ and A the alphabet that consists of the $2r+s-2$ distinct symbols $a_1, a_2, \ldots, a_{r-1}$, $b_1, b_2, \ldots, b_{s-1}$, $c_1, c_2, \ldots, c_r$. Furthermore, let u be the following word:

$$u = b_1 b_2 \cdots b_{s-1} a_1 a_2 \cdots a_{r-1} c_1 c_2 \cdots c_r b_1 b_2 \cdots b_{s-1}.$$

A short inspection shows that u is $(A, s-1)$-uniform and u is not (A, s)-uniform. Furthermore, u is $(A, r-1)$ distinct and u is not (A, r)-distinct. Now $r-1 = \lfloor \frac{|A|-s+1}{2} \rfloor$ and $s-1 = |A|-2r+1$, so the parameter values $\lfloor \frac{|A|-s+1}{2} \rfloor$ in Theorem 5 (a) and $|A|-2r+1$ in Theorem 5 (b) are optimal.

We say that the word α is (A, s)-*nonuniform* if $A \subseteq \text{alph}(\alpha)$ and α is not (A, s)-uniform. If α is an (A, s)-nonuniform word, then for any subword β of α, any subset B of $A \cap \text{alph}(\beta)$ and any integer r such that $r \geq s$, the word β is clearly (B, r)-nonuniform.

Lemma 7. Let d, s and t be positive integers, A an alphabet such that $|A| \geq d(s+t-1)$ and α an (A, s)-nonuniform word. Then there exists a factorization $\alpha = \alpha_1 \alpha_2 \cdots \alpha_d$ of α and subsets $A_1, A_2, \ldots, A_d$ of A such that for each $i \in \{1, 2, \ldots, d\}$:

1. $A_i \subseteq \text{alph}(\alpha_i)$;
2. $|A_i| \geq t$; and
3. $A_i \cap \text{alph}(\alpha_j) = \emptyset$ for all $j \in \{1, 2, \ldots, d\}$, $j \neq i$.

Proof. Let $\alpha = \alpha_1 \alpha_2 \cdots \alpha_d$ be a factorization of α such that for all $i \in \{1, 2, \ldots, d\}$, α_i contains at least $s+t-1$ symbols that do not occur in the word $\alpha_1 \alpha_2 \cdots \alpha_{i-1}$. Since $|A| \geq d(s+t-1)$, the factorization can always be found. Since α is (A, s)-nonuniform, each factor of it is as well. Thus for $i \in \{1, 2, \ldots, d\}$, there exists a subset A_i of $\text{alph}(\alpha_i) \setminus \text{alph}(\alpha_1 \alpha_2 \cdots \alpha_{i-1})$ of cardinality at least t such that $A_i \cap (\cup_{j=i+1}^{d} \text{alph}(\alpha_j)) = \emptyset$. The claim follows. $\square$

Now, we wish to generalize the result of Theorem 5 (a) to two directions; first for any number of nonuniform words. For this purpose we introduce the function T_s, by which we are able to evaluate the cardinalities of the disjoint subsets in distinct words. It appears that the function $T_s : x \to \lfloor \frac{x-s+1}{2} \rfloor$ provided by Theorem 5 (a) is exactly what we need.

Thus, let s be any positive integer. Define the function $T_s : \mathbb{Z} \to \mathbb{Z}$ by

$$T_s(x) = \left\lfloor \frac{x-s+1}{2} \right\rfloor.$$

The *powers* of T_s are defined recursively as follows:

$$\begin{aligned} T_s^1(x) &= T_s(x) \\ T_s^{k+1}(x) &= T_s^1(T_s^k(x)) = \left\lfloor \tfrac{T_s^k(x)-s+1}{2} \right\rfloor \quad (k \in \mathbb{N}_+) \end{aligned}$$

Our next task is to determine a closed form for $T_s^k(x)$.

Lemma 8. Let x be a real number, $z \in \mathbb{Z}$ and $p, q \in \mathbb{N}_+$. Then

$$\left\lfloor \frac{x}{p} \right\rfloor + z = \left\lfloor \frac{x+pz}{p} \right\rfloor \quad \text{and} \quad \left\lfloor \frac{\lfloor \frac{x}{p} \rfloor}{q} \right\rfloor = \left\lfloor \frac{x}{pq} \right\rfloor .$$

Proof. There exist unique $k_1, k_2 \in \mathbb{Z}$ and real numbers r_1, r_2, $0 \le r_1 < p$, $0 \le r_2 < pq$ such that $x = k_1 p + r_1$ and $x = k_2(pq) + r_2$. Certainly $\left\lfloor \frac{x}{p} \right\rfloor + z = \left\lfloor \frac{x+pz}{p} \right\rfloor = k_1 + z$ and $\left\lfloor \frac{\lfloor \frac{x}{p} \rfloor}{q} \right\rfloor = \left\lfloor \frac{x}{pq} \right\rfloor = k_2$. □

Theorem 6. For each $r \in \mathbb{N}_+$ and $x \in \mathbb{Z}$, the equality $T_s^r(x) = \left\lfloor \frac{x-(s-1)(2^r-1)}{2^r} \right\rfloor$ holds.

Proof. We proceed by induction on r. Let $x \in \mathbb{Z}$. Consider the case $r = 1$. Clearly $T_s^1(x) = \lfloor \frac{x-s+1}{2} \rfloor = \left\lfloor \frac{x-(s-1)(2^1-1)}{2^1} \right\rfloor$.

Let then $k \in \mathbb{N}_+$ and assume that the claim holds, when $r = k$:

$$T_s^k(x) = \left\lfloor \frac{x-(s-1)(2^k-1)}{2^k} \right\rfloor .$$

Consider finally the case $r = k+1$. We have

$$\begin{aligned} T_s^{k+1}(x) &= T_s(T_s^k(x)) = T_s(\left\lfloor \tfrac{x-(s-1)(2^k-1)}{2^k}) \right\rfloor) = \left\lfloor \frac{\lfloor \frac{x-(s-1)(2^k-1)}{2^k} \rfloor - s + 1}{2} \right\rfloor \\ &= \left\lfloor \tfrac{x-(s-1)(2^k-1)-(s-1)2^k}{2^{k+1}} \right\rfloor = \left\lfloor \tfrac{x-(s-1)(2^{k+1}-1)}{2^{k+1}} \right\rfloor . \end{aligned}$$

Above, the second equality holds by the induction hypothesis, the third by the definition of T_s, and the fourth by Lemma 8. The induction is now extended. □

Note that, given a large $x \in \mathbb{N}_+$, the sequence $(T_s^r(x))_{r=1}^{\infty}$ decreases exponentially with respect to r. Eventually, the elements of the sequence will be negative and $T_s^r(x) \to -s+1$ as $r \to \infty$.

What follows now is a series of quite technical lemmata in which the structural properties of (a sequence) of nonuniform words are gradually fortified. To simplify the appearance of the results, one more notion is needed.

Let β be a word, r a positive integer and $B_1, B_2, \ldots, B_r$ pairwise disjoint sets of symbols. The word β is $\{B_i\}_{i=1}^{r}$*-distinct*, if there exist a factorization $\beta = \beta_1\beta_2\cdots\beta_r$ of β and a permutation σ of $1,2,\ldots,r$ such that for each $i \in \{1,2,\ldots,r\}$

1. $B_i \subseteq \mathrm{alph}(\beta_{\sigma(i)})$; and
2. $B_i \cap \mathrm{alph}(\beta_j) = \emptyset$ for each $j \in \{1,2,\ldots,r\}$ such that $j \neq \sigma(i)$.

The first result is a generalization of Theorem 5 (a) for a sequence of nonuniform words.

Lemma 9. Let p and s be positive integers, A an alphabet and $\alpha_1, \alpha_2, \ldots, \alpha_p$ a sequence of (A,s)-nonuniform words. Then there exists (pairwise disjoint) sets $A_1, A_2 \subseteq A$ such that each of the words $\alpha_1, \alpha_2, \ldots, \alpha_p$ are $\{A_i\}_{i=1}^{2}$-distinct. Furthermore, $|A_1| \geq T_s^p(|A|)$ and $|A_2| \geq T_s^p(|A|)$.

Proof. We proceed by induction on p. Consider the case $p = 1$. The word α_1 is (A,s)-nonuniform, so Theorem 5 (a) implies that it is $(A, \lfloor \frac{|A|-s+1}{2} \rfloor)$-distinct. By definition, there thus exists pairwise distinct sets $A_1, A_2 \subseteq A$ such that α_1 is $\{A_i\}_{i=1}^{2}$-distinct and $|A_1|, |A_2| \geq \lfloor \frac{|A|-s+1}{2} \rfloor$. Certainly $\lfloor \frac{|A|-s+1}{2} \rfloor = T_s^1(|A|)$, so our lemma is true when $p = 1$.

Suppose then that $p = k+1$, $k \in \mathbb{N}_+$. Consider the sequence of words $\alpha_1, \alpha_2, \ldots, \alpha_k$. By the induction hypothesis, there exist pairwise disjoint sets $A_1', A_2' \subseteq A$ such that each of the words $\alpha_1, \alpha_2, \ldots, \alpha_k$ is $\{A_i'\}_{i=1}^{2}$-distinct and $|A_1'|, |A_2'| \geq T_s^k(|A|)$.

Let $A' = A_1' \cup A_2'$. Clearly $|A'| \geq 2T_s^k(|A|)$. Since α_{k+1} is (A,s)-nonuniform and $A' \subseteq A$, the word α_{k+1} is also (A',s)-nonuniform. By Theorem 5 (a), α_{k+1} is (A',t)-distinct, where $t = \lfloor \frac{|A'|-s+1}{2} \rfloor$. Then there exists a factorization $\alpha_{k+1} = \beta_1\beta_2$ of α_{k+1} and sets $X, Y \subseteq A'$ such that $X = [\mathrm{alph}(\beta_1) \setminus \mathrm{alph}(\beta_2)] \cap A'$, $Y = [\mathrm{alph}(\beta_2) \setminus \mathrm{alph}(\beta_1)] \cap A'$, and $|X|, |Y| \geq \lfloor \frac{2T_s^k(|A|)-s+1}{2} \rfloor$. Then let $X = B_1 \cup C_1$ and $Y = B_2 \cup C_2$, where $B_i \subseteq A_1'$ and $C_i \subseteq A_2'$ for $i = 1,2$. Moreover, let $D = \mathrm{alph}(\beta_1) \cap \mathrm{alph}(\beta_2) \cap A'$. Since α_{k+1} is (A',s)-nonuniform, the cardinality of D is at most $s-1$ and $|B_1| + |B_2|, |C_1| + |C_2| \geq T_s^k(|A|) - s + 1$.

We claim that either $|B_1|, |C_2| \geq \frac{T_s^k(|A|)-s+1}{2}$ or $|B_2|, |C_1| \geq \frac{T_s^k(|A|)-s+1}{2}$. Suppose that, for instance, $|B_2| < \frac{T_s^k(|A|)-s+1}{2}$. Then, by the facts above, $|B_1| > \frac{T_s^k(|A|)-s+1}{2}$ and

$$|C_2| > \frac{2\,T_s^k(|A|)-s}{2} - \frac{T_s^k(|A|)-s+1}{2} = \frac{T_s^k(|A|)-1}{2}\,.$$

We deduce that $|C_2| \geq \frac{T_s^k(|A|)-s+1}{2}$. Thus $|B_2| < \frac{T_s^k(|A|)-s+1}{2}$ implies that $|B_1|, |C_2| \geq \frac{T_s^k(|A|)-s+1}{2}$, so our claim holds.

Assume that $|B_1|, |C_2| \geq \frac{T_s^k(|A|)-s+1}{2}$ and choose $A_1 = B_1$ and $A_2 = C_2$. Then α_{k+1}, as well as the words $\alpha_1, \alpha_2, \ldots, \alpha_k$ are $\{A_i\}_{i=1}^2$-distinct and moreover, $|A_1|, |A_2| \geq \left\lfloor \frac{T_s^k(|A|)-s+1}{2} \right\rfloor = T_s^{k+1}(|A|)$. The induction is extended. □

In our future considerations, we need a much stronger tool than the previous lemma can provide. The factorization of each nonuniform word $\alpha_1, \alpha_2, \ldots, \alpha_p$ in Lemma 9 should be refined and the common alphabets for refinements created.

Theorem 7. Let p and s be positive integers, A an alphabet, and $\alpha_1, \alpha_2, \ldots, \alpha_p$ a sequence of (A,s)-nonuniform words. Then, given $d \in \mathbb{N}_+$, there exist (pairwise disjoint) sets $A_1, A_2, \ldots, A_{2^d} \subseteq A$ such that the word α_j is $\{A_i\}_{i=1}^{2^d}$-distinct for $j = 1, 2, \ldots, p$ and $|A_i| \geq T_s^{p\cdot d}(|A|)$ for $i = 1, 2, \ldots, 2^d$.

Proof. Proceed by induction on d. In the case $d = 1$ our theorem restates the result of Lemma 9.

Consider the case $d = k+1$, where $k \in \mathbb{N}_+$. By Lemma 9 there exist disjoint alphabets $D_1, D_2 \subseteq A$ such that each of the words $\alpha_1, \alpha_2, \ldots,\ \alpha_p$ is $\{D_i\}_{i=1}^2$-distinct and $|D_1|, |D_2| \geq T_s^p(|A|)$. By definition, for each $j \in \{1, 2, \ldots, p\}$ there exists a permutation σ_j of 1,2 and a factorization $\alpha_j = \alpha_{j,1}\alpha_{j,2}$ of α_j such that, for each $i \in \{1,2\}$, (i) $D_i \subseteq \mathrm{alph}(\alpha_{j,\sigma_j(i)})$; and (ii) $D_i \cap \mathrm{alph}(\alpha_{j,k}) = \emptyset$ for $k \in \{1,2\}$ such that $k \neq \sigma_j(i)$. Let $\beta_j = \alpha_{j,\sigma_j(1)}$ and $\gamma_j = \alpha_{j,\sigma_j(2)}$ for $j = 1, 2, \ldots, p$. Obviously $\alpha_j = \beta_j\gamma_j$ ($\alpha_j = \gamma_j\beta_j$, resp.) if $\sigma_j(1) = 1$ and $\sigma_j(2) = 2$ ($\sigma_j(1) = 2$ and $\sigma_j(2) = 1$, resp.).

Certainly, for each $j \in \{1, 2, \ldots, p\}$ the word β_j is (D_1, s)-nonuniform and the word γ_j is (D_2, s)-nonuniform.

Apply the induction hypothesis to the words $\beta_1, \beta_2, \ldots, \beta_p$ to obtain alphabets $B_1, B_2, \ldots, B_{2^k} \subseteq D_1$ such that the word β_j is $\{B_i\}_{i=1}^{2^k}$-distinct for $j = 1, 2, \ldots, p$ and $|B_i| \geq T_s^{p\cdot k}(|D_1|)$ for $i = 1, 2, \ldots, 2^k$.

Apply the induction hypothesis once more, now to the words $\gamma_1, \gamma_2, \ldots, \gamma_p$, to obtain alphabets $C_1, C_2, \ldots, C_{2^k} \subseteq D_2$ such that the word γ_j is $\{C_i\}_{i=1}^{2^k}$-distinct for $j = 1, 2, \ldots, p$, and $|C_i| \geq T_s^{p\cdot k}(|D_2|)$ for $i = 1, 2, \ldots, 2^k$.

Note that $|B_i|, |C_i| \geq T_s^{p\cdot k}(T_s^p(|A|)) = T_s^{p\cdot(k+1)}(|A|)$ for $i = 1,2,\dots,2^k$. Let $A_i = B_i$ and $A_{2^k+i} = C_i$ for $i = 1,2,\dots,2^k$. Since the word β_j is $\{B_i\}_{i=1}^{2^k}$-distinct, the word γ_j is $\{C_i\}_{i=1}^{2^k}$-distinct and $B_i \cap C_j = \emptyset$ for $i,j \in \{1,2,\dots,2^k\}$, the word α_j is $\{A_i\}_{i=1}^{2^{k+1}}$-distinct, when $j = 1,2,\dots,p$. Moreover, $|A_i| \geq T_s^{p\cdot(k+1)}(|A|)$ for $i = 1,2,\dots,2^{k+1}$.

The induction is thus extended and the proof is complete. □

Note that in the previous theorem, the sets $A_1, A_2, \dots, A_{2^d}$ are pairwise disjoint. The theorem is evidently a generalization of Theorem 5 (a): in case $p = d = 1$ it provides the boundary $\left\lfloor \frac{|A|-s+1}{2} \right\rfloor$ for the cardinalities of the sets A_1 and A_2.

Given $p \in \mathbb{N}_+$, we are interested in those (positive) $x \in \mathbb{Z}$ that $T_s^p(x) \geq 1$. For each $r \in \mathbb{N}_+$, denote $D_s(r) = 2^r + (s-1)(2^r - 1)$. Obviously $T_s^p(x) = \left\lfloor \frac{x-(s-1)(2^p-1)}{2^p} \right\rfloor$ is an increasing function; since $T_s^p(D_s(p)) = 1$, we deduce that $D_s(p)$ is the smallest integer z such that $T_s^p(z) > 0$.

The defintion of $D_s(r)$ straightforwardly implies that, for each $i \in \mathbb{N}_+$, we have

$$D_s(p+i) = 2^p D_s(i) + (s-1)(2^p - 1) \quad \text{and} \quad T_s^p(D_s(p+i)) = D_s(i)\ .$$

The previous theorem describes the division of p nonuniform words $\alpha_1, \alpha_2, \dots, \alpha_p$ into factors containing pairwise distinct alphabets (which, however, are common to all words $\alpha_1, \alpha_2, \dots, \alpha_p$). The main result of this section follows and further generalizes Theorem 7. In the first phase Theorem 7 is applied to (nonuniform) words $\alpha_1, \alpha_2, \dots, \alpha_p$, then in $p-1$ further phases Theorem 7 is applied to words $\alpha_1, \alpha_2, \dots, \alpha_{p-i}$, where $i = 1,2,\dots,p-1$.

Theorem 8. Let p and s be positive integers, A an alphabet, and $\alpha_1, \alpha_2, \dots, \alpha_p$ a sequence of (A,s)-nonuniform words. Given $d_1, d_2, \dots, d_p \in \mathbb{N}_+$, there exist sets $A_{i,j} \subseteq A$ ($i = 1,2,\dots,p$; $j = 1,2,\dots,2^{\sum_{l=i}^p d_l}$) such that

1. for each $i \in \{1,2,\dots,p\}$, the word α_i is $\{A_{i,j}\}_{j=1}^{2^{\sum_{l=i}^p d_l}}$-distinct;
2. for each $i \in \{1,2,\dots,p\}$ and $j \in \{1,2,\dots,2^{\sum_{l=i}^p d_l}\}$
$$|A_{i,j}| \geq T_s^{\sum_{l=i}^p l\cdot d_l}(|A|)\ ;$$
and
3. for each $i \in \{2,3,\dots,p\}$ and $j \in \{1,2,\dots,2^{\sum_{l=i}^p d_l}\}$ there exist $2^{d_{i-1}}$ indices $j' \in \{1,2,\dots,2^{\sum_{l=i-1}^p d_l}\}$ such that $A_{i-1,j'} \subseteq A_{i,j}$.

Proof. We proceed by induction on p. Our claims in case $p = 1$ are a direct consequence of Theorem 7.

Consider the case $p=k+1$. Apply again Theorem 7 to $p=k+1$ and $d=d_{k+1}$ to discover sets $B_1,B_2,\dots,B_{2^{d_{k+1}}} \subseteq A$ such that the word α_i is $\{B_j\}_{j=1}^{2^{d_{k+1}}}$-distinct for $i=1,2,\dots,k+1$, and $|B_j| \geq T_s^{(k+1)\cdot d_{k+1}}(|A|)$ for $j=1,2,\dots,2^{d_{k+1}}$.

For each $i \in \{1,2,\dots,k+1\}$, let $\alpha_i = \beta_{i,1}\beta_{i,2}\cdots\beta_{i,2^{d_{k+1}}}$ be a factorization of α_i, and ρ_i a permutation of $1,2,\dots,2^{d_{k+1}}$ such that for each $j \in \{1,2,\dots,2^{d_{k+1}}\}$

(a) $B_j \subseteq \mathrm{alph}(\beta_{i,\rho_i(j)})$
(b) $B_j \cap \mathrm{alph}(\beta_{i,j'}) = \emptyset$ for each $j' \in \{1,2,\dots,2^{d_{k+1}}\}$ such that $j' \neq \rho_i(j)$; and
(c) $|B_j| \geq T_s^{(k+1)\cdot d_{k+1}}(|A|)$.

Let $t \in \{1,2,\dots,2^{d_{k+1}}\}$ be arbitrary, but fixed. Then $B_t \subseteq \mathrm{alph}(\beta_{i,\rho_i(t)})$ for each $i \in \{1,2,\dots,k\}$. Furthermore, each of the words $\beta_{1,\rho_1(t)}$, $\beta_{2,\rho_2(t)}$, …, $\beta_{k,\rho_k(t)}$ is (B_t,s)-nonuniform. Denote, for the sake of simplicity, $\tau_i = \beta_{i,\rho_i(t)}$ for $i=1,2,\dots,k$. Apply the induction hypothesis to the set B_t, the words $\tau_1,\tau_2,\dots,\tau_k$ and the integers $d_1,d_2,\dots,d_k$.

Let $B_{i,j}^t \subseteq B_t$ $(i=1,2,\dots,k;\ j=1,2,\dots,2^{\sum_{l=i}^k d_l})$ be pairwise disjoint sets such that

1° for each $i \in \{1,2,\dots,k\}$, the word β_i is $\{B_{i,j}^t\}_{j=1}^{2^{\sum_{l=i}^k d_l}}$-distinct;
2° for each $i \in \{1,2,\dots,k\}$ and $j \in \{1,2,\dots,2^{\sum_{l=i}^k d_l}\}$

$$|B_{i,j}^t| \geq T_s^{\sum_{l=i}^k l\cdot d_l}(|B_t|) ;$$

and
3° for each $i \in \{2,3,\dots,k\}$ and $j \in \{1,2,\dots,2^{\sum_{l=i}^k d_l}\}$ there exist exactly $2^{d_{i-1}}$ indices $j' \in \{1,2,\dots,2^{\sum_{l=i-1}^k d_l}\}$ such that $B_{i-1,j'}^t \subseteq B_{i,j}^t$.

For each $i \in \{1,2,\dots,k\}$, let $A_{i,1}, A_{i,2}, \dots, A_{i,2^{\sum_{l=i}^{k+1} d_l}}$ be the sets

$$B_{i,1}^1, B_{i,2}^1, \dots, B_{i,2^{\sum_{l=i}^k d_l}}^1, B_{i,1}^2, B_{i,2}^2, \dots, B_{i,2^{\sum_{l=i}^k d_l}}^2, \\ \dots, B_{i,1}^{2^{d_{k+1}}}, B_{i,2}^{2^{d_{k+1}}}, \dots, B_{i,2^{\sum_{l=i}^k d_l}}^{2^{d_{k+1}}},$$

respectively. Furthermore, let $A_{k+1,1}, A_{k+1,2}, \dots, A_{k+1,2^{d_{k+1}}}$ be the sets $B_1, B_2, \dots, B_{2^{d_{k+1}}}$, respectively.

It is clear that the three claims of our theorem hold, when $p=k+1$. The induction is thus extended and the proof is complete. □

7.2 Factorizing q-Bounded Words

We shall next prove a result which is the most significant from the viewpoint of our attack construction. Recall that, given $q \in \mathbb{N}_+$, a word α is *q-bounded* if $|\alpha|_a \leq q$ for all $a \in \text{alph}(\alpha)$.

Let us first intuitively contemplate the method that is applied in the proof of the subsequent theorem. Let α be a (hypothetical) q-bounded word. Our goal is to create a factorization of α and a set A that satisfy properties (Q1) and (Q2). This can be done by applying Theorem 8 when $\alpha = \alpha_1\alpha_2\cdots\alpha_p$, $A \subseteq \text{alph}(\alpha_i)$ when $i \in \{1,2,\ldots,p\}$, words $\alpha_1,\alpha_2,...,\alpha_p$ are $(A,s)-$nonuniform and positive integers $d_1,d_2,\ldots,d_p$ are chosen appropriately (we shall later see how). In addition the set A has to be large enough when compared with s to ensure that all the created sets $A_{i,j}$ are nonempty.

To begin with, imagine that we have factorized α into the form $\alpha = \alpha_1\alpha_2\cdots\alpha_q$. Suppose that $\cap_{j=1}^{q}\text{alph}(\alpha_j) \neq \emptyset$. Certainly each α_i is $(\cap_{j=1}^{q}\text{alph}(\alpha_j),\ 1)$-nonuniform. How large should the set $\cap_{j=1}^{q}\text{alph}(\alpha_j)$ be to guarantee that all the sets $A_{i,j}$ in Theorem 8 are nonempty when $p=q$? The definition of the constant $D_s(r)$ implies that if the condition $|\cap_{j=1}^{q}\text{alph}(\alpha_j)| \geq D_1(\sum_{j=1}^{q} j\cdot d_j)$ holds the sets are nonempty. Let $s_q = D_1(\sum_{j=1}^{q} j\cdot d_j) = 2^{\sum_{j=1}^{q} j\cdot d_j}$.

In the second step, imagine that we have factorized α into the form $\alpha = \alpha_1\alpha_2\cdots\alpha_{q-1}$. Suppose that $\cap_{j=1}^{q-1}\text{alph}(\alpha_j) \neq \emptyset$ and each word $\alpha_1,\alpha_2,\ldots,\alpha_{q-1}$ is $(\cap_{j=1}^{q-1}\text{alph}(\alpha_j),s_q)$-nonuniform. How large should the set $\cap_{j=1}^{q}\text{alph}(\alpha_j)$ be to guarantee that all the sets $A_{i,j}$ in Theorem 8 were nonempty when $p=q-1$? Again, the definition of the constant $D_s(r)$ implies that if the condition $|\cap_{j=1}^{q-1}\text{alph}(\alpha_j)| \geq D_{s_q}(\sum_{j=1}^{q-1} j\cdot d_j)$ holds, then the sets are nonempty. Let $s_{q-1} = D_{s_q}(\sum_{j=1}^{q-1} j\cdot d_j)$.

Continuing like this, we reach in the $(q-1)$st step the situation where we have factorized α into the form $\alpha = \alpha_1\alpha_2$. Suppose that $\text{alph}(\alpha_1)\cap\text{alph}(\alpha_2) \neq \emptyset$ and both of the words α_1,α_2 are $(\text{alph}(\alpha_1)\cap\text{alph}(\alpha_2),s_3)$-nonuniform. If $|\text{alph}(\alpha_1)\cap\text{alph}(\alpha_2)| \geq D_{s_3}(d_1+2d_2)$ the sets $A_{i,j}$ in Theorem 8 for $p=2$ are nonempty . Let $s_2 = D_{s_3}(d_1+2d_2)$.

Finally, in the q^{th} step, we are in the situation where α is $(\text{alph}(\alpha),s_2)$-nonuniform. If $|\text{alph}(\alpha)| \geq D_{s_2}(d_1)$, then certainly the sets of $A_{1,j}$ of Theorem 8 are all nonempty. We then put $s_1 = D_{s_2}(d_1)$ and approximate the size of the alphabet $\text{alph}(\alpha)$ with s_1.

The procedure above suggests the following: If $d_1,d_2,\ldots,d_q$ are positive integers and α is a q-bounded word for which $|\text{alph}(\alpha)| \geq s_1$, where the true value of s_1 is achieved through the sequence $s_i = D_{s_i+1}(\sum_{j=1}^{i} j\cdot d_j)$ for $i = 1,2,\ldots,q-1$, $s_q =$

$D_1(\sum_{j=1}^{q} j \cdot d_j)$, then, for some $p \in \{1,2,\dots,q\}$, Theorem 8 can be applied and a factorization for the word α and a set A satisfying properties (Q1) and (Q2) created. The next theorem is a rigorous reasoning that our intuition works.

Theorem 9. Let $q \geq 2$ and $d_1,d_2,\dots,d_q \geq 1$ be integers and $s_1,s_2,\dots,s_q$ parameters defined as follows: $s_q = 2^{\sum_{j=1}^{q} jd_j}$, $s_k = D_{s_{k+1}}(\sum_{i=1}^{k} i \cdot d_i)$, for $k = 2,3,\dots,q-1$, and $s_1 = 2^{d_1}s_2$. Furthermore, let α be a q-bounded word such that $|\mathrm{alph}(\alpha)| \geq s_1$. Then there exist $p \in \{1,2,\dots,q\}$, a factorization $\alpha = \alpha_1\alpha_2\cdots\alpha_p$ of α, and sets $A_{i,j} \subseteq \mathrm{alph}(\alpha)$ $(i = 1,2,\dots p;\ j = 1,2,\dots,2^{\sum_{k=i}^{p} d_k})$ such that

1. for each $i \in \{1,2,\dots,p\}$, the word α_i is $\{A_{i,j}\}_{j=1}^{2^{\sum_{k=i}^{p} d_k}}$-distinct;
2. for each $i \in \{1,2,\dots p\}$ and $j \in \{1,2,\dots,2^{\sum_{k=i}^{p} d_k}\}$:
$$|A_{i,j}| \geq 2^{\sum_{k=1}^{i-1} kd_k}\ ;$$
and
3. for each $i \in \{2,3,\dots,p\}$ and $j \in \{1,2,\dots,2^{\sum_{k=i}^{p} d_k}\}$ there exist exactly $2^{d_{i-1}}$ indices $l \in \{1,2,\dots,2^{\sum_{k=i-1}^{p} d_k}\}$ such that $A_{i-1,l} \subseteq A_{i,j}$.

Proof. We proceed stepwise as follows.

In the `first step` we ask whether or not the word α is $(\mathrm{alph}(\alpha),s_2)$-uniform.

Assume first that α is $(\mathrm{alph}(\alpha),s_2)$-nonuniform. Let then $\alpha = \alpha_1\alpha_2\cdots\alpha_{2^{d_1}}$ be a factorization of α such that for each $i \in \{1,2,\dots,2^{d_1}\}$, the word α_i contains (at least) s_2 different symbols that do not occur in $\alpha_1\alpha_2\cdots\alpha_{i-1}$. Since $|\mathrm{alph}(\alpha)| \geq 2^{d_1}s_2$, the factorization always can be found. Let $A_{1,i}$ be the set of all symbols in $\mathrm{alph}(\alpha_i)$ that do not occur in $\mathrm{alph}(\alpha_1\cdots\alpha_{i-1}\alpha_{i+1}\cdots\alpha_{2^{d_1}})$. Since α is $(\mathrm{alph}(\alpha),s_2)$-nonuniform, each of the sets $A_{1,1}$, $A_{1,2}$, …, $A_{1,2^{d_1}}$ is nonempty. By choosing $p = 1$, we note that the claims of the theorem hold. We use here the convention $\sum_{k=1}^{p-1} kd_k = \sum_{k=1}^{0} kd_k = 0$, i.e., $2^{\sum_{k=1}^{p-1} kd_k} = 1$.

Suppose then that the word α is $(\mathrm{alph}(\alpha),s_2)$-uniform. Let then $\alpha = \alpha_1\alpha_2$ be a factorization of α such that $|\mathrm{alph}(\alpha_1) \cap \mathrm{alph}(\alpha_2)| \geq s_2$. Recall that, by definition, $s_2 = D_{s_3}(d_1 + 2d_2)$.

In the `second step` we ask whether there exists an alphabet $B \subseteq \mathrm{alph}(\alpha_1) \cap \mathrm{alph}(\alpha_2)$ such that one of the words α_1 and α_2 is (B,s_3)-uniform.

Assume first that each of the words α_1 and α_2 is (A,s_3)-nonuniform, where $A = \mathrm{alph}(\alpha_1) \cap \mathrm{alph}(\alpha_2)$. Since $|A| \geq D_{s_3}(d_1 + 2d_2)$, the claims hold by Theorem 8.

Suppose then in the `second step` that there exists a set $B \subseteq \mathrm{alph}(\alpha_1) \cap \mathrm{alph}(\alpha_2)$ such that one of the words α_1,α_2, say α_2, is (B,s_3)-uniform. Let $\alpha_2 = \gamma_1\gamma_2$ be a factorization of α_2 such that $|\text{alph}(\gamma_1)\cap\text{alph}(\gamma_2)\cap B|\geq s_3$. Redenoting $\alpha_2:=\gamma_1$ and $\alpha_3:=\gamma_2$, we have $\alpha=\alpha_1\alpha_2\alpha_3$ and $|\cap_{i=1}^{3}\text{alph}(\alpha_i)|\geq s_3$.

Continuing like this, we describe the general `step` k, $k\in\{1,2,\ldots,q\}$ as follows. Let $\alpha=\alpha_1\alpha_2\cdots\alpha_k$ be a factorization of α such that $|\cap_{i=1}^{k}\text{alph}(\alpha_i)|\geq s_k$. Assume first that $k<q$. Recall that $s_k=D_{s_{k+1}}(\sum_{i=1}^{k} i\cdot d_i)$. We pose the question whether there exists an alphabet $B\subseteq\cap_{i=1}^{k}\text{alph}(\alpha_i)$ such that one of the words $\alpha_1,\alpha_2,\ldots,\alpha_k$ is (B,s_{k+1})-uniform.

Assume first that in `step` k, the answer to the question is negative. Then each of the words $\alpha_1,\alpha_2,\ldots,\alpha_k$ is (A,s_{k+1})-nonuniform; here we have $A=\cap_{i=1}^{k}\text{alph}(\alpha_i)$. Since $|A|\geq D_{s_{k+1}}(\sum_{i=1}^{k} i\cdot d_i)$, the claims hold by Theorem 8.

Suppose then that the answer to the question is positive. Let then B be a subset of $\cap_{i=1}^{k}\text{alph}(\alpha_i)$ such that one of the words $\alpha_1,\alpha_2,\ldots,\alpha_k$, say α_k is (B,s_{k+1})-uniform. Let $\alpha_k=\omega_1\omega_2$ be a factorization of α_k such that $|\text{alph}(\omega_1)\cap\text{alph}(\omega_2)\cap B|\geq s_{k+1}$. Redenoting $\alpha_k:=\omega_1$ and $\alpha_{k+1}:=\omega_2$, we have $\alpha=\alpha_1\alpha_2\cdots\alpha_{k+1}$ and $|\cap_{i=1}^{k+1}\text{alph}(\alpha_i)|\geq s_{k+1}$.

Suppose that in the general step we have $k=q$. Then, since α is q-bounded, we know that all the words $\alpha_1,\alpha_2,\ldots,\alpha_k$ are $(A,1)$-nonuniform, where $A=\cap_{i=1}^{q}\text{alph}(\alpha_i)$. Since $s_q=2^{\sum_{i=1}^{q} i s_i}$, we are again through by Theorem 8. □

Corollary 1. Let $q\geq 2$ and $d_1,d_2,\ldots,d_q\geq 1$ be integers and α a q-bounded word such that $|\text{alph}(\alpha)|\geq 2^{\sum_{i=1}^{q}(q-i+1)id_i}$. Furthermore, let $s_1,s_2,\ldots,s_q$ be as in Theorem 9. Then $|\text{alph}(\alpha)|\geq s_1$ and all the claims of Theorem 9 hold.

Proof. Recall that $s_q=2^{\sum_{j=1}^{q} jd_j}$, $s_k=D_{s_{k+1}}(\sum_{i=1}^{k} i\cdot d_i)$, for $k=2,3,\ldots,q-1$, and $s_1=2^{d_1}s_2$. It is easy to see that $s_k=D_{s_{k+1}}(\sum_{i=1}^{k} i\cdot d_i))\leq s_{k+1}2^{\sum_{i=1}^{k} id_i}$ for $k=1,2,\ldots,q-1$. We get

$$\begin{aligned} s_1 &= 2^{d_1}s_2\leq 2^{d_1}\cdot 2^{d_1+2d_2}\cdot s_3\leq\cdots \\ &\leq 2^{d_1}\cdot 2^{d_1+2d_2}\ldots 2^{d_1+2d_2+\cdots+qd_q} \\ &= 2^{\sum_{i=1}^{q}(q-i+1)id_i}\;. \end{aligned}$$

Then $|\text{alph}(\alpha)|\geq s_1$ and the claims of Theorem 9 are valid. □

7.3 Attacking Bounded Generalized Iterated Hash Functions

Finally, we have found an upper bound l_1 which will ensure that, if α is $q-$bounded and $|\text{alph}(\alpha)|>l_1$, then it is possible to create a set A and a factorization $\alpha=\beta_1\beta_2\cdots\beta_p$ that will satisfy both (Q1) and (Q2).

We are now ready to state the main result of this chapter.

Theorem 10. Let m,n and q be positive integers such that $m>n$ and $q\geq 2$, $f:\{0,1\}^n\times\{0,1\}^m\to\{0,1\}^n$ a compression function, and $\hat{\alpha}=(\alpha_1,\alpha_2,\dots)$ a q-bounded sequence of words such that $\mathrm{alph}(\alpha_l)=\mathbb{N}_l$ for each $l\in\mathbb{N}_+$. Then, for each $k\in\mathbb{N}_+$, there exists a 2^k-collision attack on the generalized iterated hash function $\mathtt{H}_{\hat{\alpha},f}$ such that the expected number of queries on f is at most $2.5\cdot q\cdot 2^{\lceil\log_2 n\rceil\frac{(q+4)q(q-1)}{6}+\lceil\log_2 k\rceil q}\cdot 2^{\frac{n}{2}}$.

Proof. Choose $d_i=\lceil\log_2 n\rceil$ for $i=1,2,\dots,q-1$ and $d_q=\lceil\log_2 k\rceil$. Then

$$\begin{aligned}\textstyle\sum_{i=1}^{q}(q-i+1)\,i\,d_i &= \textstyle\sum_{i=1}^{q-1}(q-i+1)\,i\lceil\log_2 n\rceil+q\lceil\log_2 k\rceil\\ &= \tfrac{(q+4)q(q-1)}{6}\lceil\log_2 n\rceil+q\lceil\log_2 k\rceil\,.\end{aligned}$$

Let $|\mathrm{alph}(\alpha)|\geq 2^{\lceil\log_2 n\rceil\frac{(q+4)q(q-1)}{6}+\lceil\log_2 k\rceil q}$. Now Corollary 1 (and Theorem 9) imply that the properties (Q1) and (Q2) hold. Thus Theorem 4 guarantees that we are able to find a 2^k-collision. □

We can combine this result to the one offered in subsection 2.7.2. Thus once again lowering the upper bound of the attack to $\breve{a}\cdot q\cdot 2^{\lceil\log_2 n\rceil\frac{(q+4)q(q-1)}{6}+\lceil log_2 k\rceil q}\cdot 2^{\frac{n}{2}}$.

Our new upper bound gives us, for $n=256$, $k=4$ and $q=3$, the complexity of $7{,}5\cdot 2^{190}$. This is clearly well below the complexity $\sqrt[16]{16!}\cdot 2^{240}$ offered by the brute force attack (see [32]).

Consider once again a situation, where the inner state of the hash function is larger or equal to the message block size (in bits) i.e. $m\leq n$ in Definition 2. To guarantee that the attack succeeds, the attacker has to ensure that, when applying Theorem 9, $|A_{1,j}|\geq i$ for all $j\in\{1,2,\cdots,2^{\sum_{k=1}^{p}d_k}\}$, where i is the smallest positive integer such that $im>n$. This can be done by simply replacing d_1 in Theorem 9 with $d_1+log_2 i$. This multiplies the complexity of the attack with i^q.

Table 9. Complexities of different attack versions.

Published article	Complexity
[14]	$O(g(n,k,q)2^{\frac{n}{2}})$
[19]	$2.5 \cdot q \cdot 2^{2^{2^q-3}} k^{(2q-3)2^{2^q-1}} n^{(q-1)^2 2^{2^q-1}} 2^{\frac{n}{2}}$
[21]	$2.5 \cdot q \cdot k^{(2q-3)2^q} n^{(q-1)^2 2^q} 2^{\frac{n}{2}}$
[24]	$2.5 \cdot q \cdot 2^{\lceil \log_2 n \rceil \frac{(q+4)q(q-1)}{6} + \lceil \log_2 k \rceil q} \cdot 2^{\frac{n}{2}}$

The attacks create a 2^k-collision on $q-$bounded generalized iterated hash function. In [14] $g(n,q,k)$ is a function of n,q and k which is polynomial with respect to n and k, but triple exponential with respect to q.

7.4 Supplementary Combinatorial Considerations

Often the problems of combinatorics on words consider a situation, where the alphabet is fixed (finite) and the length of a word can be arbitrarily long [26]. Then, as the length of the word increases, unavoidable regularities start to appear and some classic results of combinatorics on words, like Ramsey's, Shirshov's and van der Waerden's Theorems, can be applied. The unavoidable regularities thus result from the fact that the number of symbols that can appear in the word is restricted.

The approach induced by $q-$bounded generalized iterated hash functions is somewhat different. The number of times a symbol can appear in a word is restricted and we increase the number of symbols that appear in a word. As we have already seen, unavoidable regularities start to appear in this case as well. We will now take a short look at these regularities.

Let us further study permutations inside q-bounded words. The results are not needed in our attack construction, but have an independent combinatorial interest. We wish to find a (good) upper and lower bound for the number $N(t,q)$ defined in Theorem 1. As we have seen in [21] the upper bound $N(t,q+1) \leq N(t^2-t+1,q) \leq t^{2^q}$ was attained. The existence of $N(t,q)$ was first (implicitely) proved in [14], where also the first (very large) upper bound for it was evolved.

Using the results of this chapter we are now able to create more efficient tools and thus lower the previous upper bound.

Theorem 11. Let $t \geq 2$ and $q \geq 2$ be integers and α be a q-bounded word such that $|\mathrm{alph}(\alpha)| \geq 2^{\lceil \log_2 t \rceil \cdot \frac{q^2-q+2}{2}}$. Then there exist $p \in \{1,2,\cdots q\}$, a factorization $\alpha = \alpha_1\alpha_2\cdots\alpha_p$ of α and a set $A \subseteq \mathrm{alph}(\alpha)$ such that $|A| = t$ and $(\alpha_i)_A$ is a permutation of A for all $i \in \{1,2,\cdots,p\}$.

Proof. Analogous (although simpler) to that of Theorem 9. $\square$

As a consequence of Theorem 11, we get the following upper bound.

Corollary 2. For all integers $t \geq 2$ and $q \geq 2$ the inequality

$$N(t,q) \leq 2^{\lceil \log_2 t \rceil \cdot \frac{q^2-q+2}{2}}$$

holds.

In the following, we shall search for a lower bound for $N(t,q)$.

Given $p,q \in \mathbb{N}_+$, call a word α a $P(t,q)$*-word*, if α is q-bounded and there exists an alphabet $A \subseteq \mathrm{alph}(\alpha)$, $|A| = t$, integer $p \in \{1,2,\ldots,q\}$, and permutations $\sigma_1,\sigma_2\ldots,\sigma_p$ of $1,2,\ldots,t$ such that

$$\pi_A(\alpha) \in a^+_{\sigma_1(1)}a^+_{\sigma_1(2)}\cdots a^+_{\sigma_1(t)}\cdots a^+_{\sigma_p(1)}a^+_{\sigma_p(2)}\cdots a^+_{\sigma_p(t)}.$$

We have shown that there exists a smallest positive integer $N(t,q)$ such that if α is q-bounded and $|\mathrm{alph}(\alpha)| \geq N(t,q)$, then α is a $P(t,q)$-word. Let $T(t,q) = N(t,q)-1$. Then there exists a word β such that β is q-bounded and $|\mathrm{alph}(\beta)| = T(t,q)$, and β *is not* a $P(t,q)$-word.

Let r be a positive integer, $a_1,a_2,\ldots,a_r$ pairwise distinct symbols, and α a word such that $\mathrm{alph}(\alpha) = \{a_1,a_2,\ldots,a_r\}$. Furthermore, assume that for all $i \in \{1,2,\ldots,r-1\}$, the first occurrence of a_i in α happens before the first occurrence of a_{i+1} in α. To emphasize this, we write $\alpha = \alpha(a_1,\ldots,a_r)$. For any symbol b, let $v[\alpha(a_1,\ldots,a_r),b] = a_1a_2\cdots a_rb\alpha(a_1,\ldots,a_r)$. For any r new symbols $b_1,b_2,\ldots,b_r$, denote by $\alpha(b_1,\ldots,b_r)$ the word achieved from α by replacing each occurrence of a_i with b_i for $i = 1,2,\ldots,r$.

Lemma 10. Let $t \geq 2$ and q be positive integers and $\alpha = \alpha(a_1,\ldots,a_r)$ a q-bounded word, which is not a $P(t,q)$-word. Let a_{r+1} be a new symbol. Then for the word $v[\alpha(a_1,\ldots,a_r),a_{r+1}] = a_1a_2\cdots a_ra_{r+1}\alpha(a_1,\ldots,a_r)$ the following holds:

1. $v[\alpha(a_1,\ldots,a_r),a_{r+1}]$ is not a $P(t,q+1)$-word; and

2. for all $A \subseteq \{a_1, a_2, \ldots, a_{r+1}\}$, $|A| > 1$ the word $(v[\alpha(a_1, \ldots, a_r), a_{r+1}])_A$ is not a permutation of A.

Proof. The second claim is clearly true, since the letters $a_1, a_2, \ldots, a_r$ occur in α.

Let us assume that $x = v[\alpha(a_1, \ldots, a_r), a_{r+1}]$ is a $P(t, q+1)$-word. Let $A \subseteq \text{alph}(\alpha)$, be an alphabet and $x = x_1 x_2 \cdots x_k$, where $k \in \{1, 2, \ldots, q+1\}$, a factorization of x such that $|A| = t$ and $(x_1)_A, (x_2)_A, \ldots, (x_k)_A$ are all permutations of A. Note that a_{r+1}, because of its role as a separator, cannot be in A. If $(x_1)_A$ is a factor of $a_1 a_2 \ldots a_l$ for some $l \in \{1, 2, \ldots, r+1\}$, then the symbols $a_{l+1}, a_{l+2}, \ldots, a_{r+1}$ do not occur in $(x_1)_A, (x_2)_A, \ldots, (x_k)_A$, so, by removing the prefix $a_1 a_2 \ldots a_{r+1}$ from t, we notice that α is a $P(t, q)$-word, a contradiction.

Let us then assume that

$$x_1 = a_1 a_2 \ldots a_{r+1} b_1 b_2 \ldots b_l$$

where l is a positive integer, $b_1, b_2, \ldots, b_l$ are in $\text{alph}(\alpha)$ ($b_1 b_2 \ldots b_l$ thus being a prefix of α with $b_1 = a_1$) and $(b_1 b_2 \cdots b_l)_A \neq \varepsilon$. Now each symbol in $\{b_1, b_2, \ldots, b_l\}$ occurs in x_1 before the symbol a_{r+1} and $(x_1)_A$ is a permutation, so the following holds:

(i) $(b_1 b_2 \cdots b_l)_A \in a_s^+$, where $s \in \{1, 2, \ldots, r\}$; and
(ii) $(x_1)_A$ contains only symbols in $\{a_1, a_2, \ldots, a_s\}$.

Suppose that $(x_1)_A$ contains a symbol a_j such that $j \in \{1, 2, \ldots, s-1\}$. Then, since a_j occurs in the prefix $b_1 b_2 \cdots b_l$ of α before a_s, we are in contradiction with (i) above. The previous facts imply that $|A| = 1$, again a contradiction, since $|A| = t \geq 2$. We thus deduce that x is not a $P(t, q+1)$-word. □

We can make the following conclusion.

Corollary 3. Let $t, q \in \mathbb{N}_+$ and α be as in the previous lemma. Moreover, let

$$a_{1,1}, a_{1,2}, \ldots, a_{1,r+1}, a_{2,1}, a_{2,2}, \ldots, a_{2,r+1} \ldots, a_{t-1,1}, a_{t-1,2} \ldots, a_{t-1,r+1}$$

be $(t-1)(r+1)$ pairwise distinct symbols. Then the word

$$v[\alpha(a_{1,1}, \ldots, a_{1,r}), a_{1,r+1}] v[\alpha(a_{2,1}, \ldots, a_{2,r}), a_{2,r+1}] \cdots v[\alpha(a_{t-1,1}, \ldots, a_{t-1,r}), a_{t-1,r+1}]$$

is not a $P(t, q+1)$ word.

Proof. Denote $v_i = v[\alpha(a_{i,1}, \dots, a_{i,r}), a_{i,r+1}]$ for $i = 1, 2, \dots, t-1$ and $x = v_1 v_2 \cdots v_{t-1}$. Assume that $A \subseteq \text{alph}(x)$, $|A| = t$ and $x = x_1 x_2 \cdots x_k$, where $(x_i)_A$ is a permutation of A.

Assume first that $k = 1$ and thus $(x)_A$ is a permutation of A. Now $(x)_A = (v_1)_A (v_2)_A \cdots (v_{t-1})_A$, so we have $i \in \{1, 2, \dots, t-1\}$ such that $A_1 = \text{alph}(v_i) \cap A$ contains at least two elements and $(v_i)_{A_1}$ is a permutation of A_1. This is a contradiction with Lemma 10.

Assume now that $k > 1$. Let $x_1 = v_1 v_2 \cdots v_s b$, where $s \in \{0, 1, ..., t-2\}$ and b is a prefix of v_{s+1}. Since no elements of $\text{alph}(v_1 v_2 \cdots v_s)$ appear in the word $x_2 x_3 \cdots x_k$ we can deduce that $(v_i)_A = \varepsilon$ for $i = 1, 2, \dots, s$. This means that $A \subseteq \text{alph}(b) \subseteq \text{alph}(v_{s+1})$. It follows that $(x)_A = (v_{s+1})_A$ and v_{s+1} is thus a $P(t, q+1)$-word, a contradiction with Lemma 10. $\square$

Corollary 4. For all $t, q \in \mathbb{N}_+$, the inequality

$$T(t, q+1) \geq (t-1)(T(t,q)+1)$$

holds.

Proof. In the construction of the lemma we add one letter to the word and then (in the corollary) we make $t-1$ copies of the word. $\square$

Theorem 12. For all $t, q \in \mathbb{N}_+$, $t \geq 2$, the following inequality holds:

$$N(t,q) \geq t(t-1)^{q-1} + (t-1)^{q-2} + \cdots + (t-1) + 1$$

Proof. Let us prove by induction on q that

$$T(t,q) \geq [t(t-1)^{q-1} + (t-1)^{q-2} + \cdots + (t-1) + 1] - 1 \ .$$

If $q = 1$, then $T(t,q) = t - 1 = t(t-1)^0 - 1$. Suppose that $T(t,q) \geq [t(t-1)^{q-1} + (t-1)^{q-2} + \cdots + (t-1) + 1] - 1$. Then

$$\begin{aligned} T(t, q+1) &\geq (t-1)[t(t-1)^{q-1} + (t-1)^{q-2} + \cdots + (t-1) + 1] \\ &= t(t-1)^q + (t-1)^{q-1} + \cdots + (t-1) + 1 - 1 \ . \end{aligned}$$

Thus the induction is extended and $N(t,q) \geq t(t-1)^{q-1} + (t-1)^{q-2} + \cdots + (t-1) + 1$ for all positive integers t and q such that $t \geq 2$. $\square$

Corollary 5. For all integers $t \geq 2$ and $q \geq 2$ the inequalities

$$t(t-1)^{q-1} + (t-1)^{q-2} + \cdots + (t-1) + 1 \leq N(t,q) \leq 2^{\lceil \log_2 t \rceil \cdot \frac{q^2-q+2}{2}}$$

hold. Furthermore $N(t,2) = t^2 - t + 1$ for each integer $t \geq 1$.

Proof. The result

$$t(t-1)^{q-1} + (t-1)^{q-2} + \cdots + (t-1) + 1 \leq N(t,q) \leq 2^{\lceil \log_2 t \rceil \cdot \frac{q^2-q+2}{2}}$$

follows directly from Theorem 11 and Theorem 12. The proof of Theorem 1 shows that $N(t,2) \leq N(t^2-t+1,1) = t^2-t+1$ while Theorem 12 shows that $N(t,2) \geq t(t-1)^1 + 1 = t^2 - t + 1$. □

8 Conclusion

In this thesis, we have presented a new algorithm for creating a diamond structure and proven that time complexity of that algorithm is $O(2^{\frac{n+d}{2}})$ thus lowering the estimated complexity of all attacks that involve diamond structures. These include the herding attack and the second preimage attack against standard iterated hash functions and several of its variants like Hash Twice and Zipper Hash.

We have also formulated a new security property, labeled Trojan message resistance, required from all hash functions. We have studied the lower bound of hash function queries required to break Trojan message resistance, when the hash function is a random oracle. In addition we have created two new versions of Trojan message attacks that have lower time complexities than the previous ones. We have presented a new variant of Joux's multicollision attack that allows the attacker to lower the complexity of the attack, when there is enough memory available and the attacker is creating a large multicollision.

Finally, we studied the complexity of creating a Joux's type multicollision attack against generalized iterated $q-$bounded hash functions and lowered the upper bound of the complexity of creating such an attack. We have also stated a reason to consider combinatorics on words from a new viewpoint, where the number of times a symbol can appear in a word is restricted, and proven some results in this new research frame.

An obvious research topic in the future would be to search to generalize and sharpen the results of this thesis or implement them in practice. Especially it would be interesting to see, if it is possible to increase the lower bounds for many attacks presented here.

References

1. Andreeva E, Bouillaquet C, Dunkelman O & Kelsey J (2009) Herding, second preimage and Trojan message attacks beyond Merkle-Damgård. Proceedings of the 16th Annual International Workshop on Selected Areas in Cryptography – SAC 2009: 393–414 .
2. Andreeva E, Bouillaquet C, Fouque P-A, Hoch J, Kelsey J, Shamir A & Zimmer S (2008) Second preimage attacks on dithered hash functions. Proceedings of the 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2008: 270–288.
3. Bellare M & Rogaway P (2007) Random oracles are practical: a paradigm for designing efficient protocols. Proceedings of the 1st ACM Conference on Computer and Communications Security – CCS 1993: 62–73.
4. Biham E & Dunkelman O (2007) A framework for iterative hash functions — HAIFA. Cryptology ePrint Archive, Report 2007/278, URL:http://eprint.iacr.org. Cited 2014/29/10.
5. Blackburn S, Stinson D & Upadhyay J (2012) On the complexity of the herding attack and some related attacks on hash functions. Designs, Codes and Cryptography 64(1–2): 171–193.
6. Damgård I (1990) A design principle for hash functions. Proceedings of the 9th Annual International Cryptology Conference – CRYPTO 1989: 416-427.
7. Dean R (1999) Formal aspects of mobile code security. PhD thesis, Princeton University.
8. Diaconis P & Mosteller F (1989) Methods for studying coincidences. Journal of the American Statistical Association 84: 853–861.
9. Dilworth R (1950) A Decomposition theorem of partially ordered sets. Annals of Mathematics 51: 161–166.
10. Dobbertin H (1998) Cryptanalysis of MD4. Journal of Cryptology 11: 253–271.
11. Dunkelman O & Preneel B (2009) Generalizing the herding attack to concatenated hashing schemes. URL:http://events.iaik.tugraz.at/HashWorkshop07/program.html. Cited 2014/29/10.
12. Flajolet P & Odlyzko A (1990) Random mapping statistics. Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques – EUROCRYPT 1989: 329–354.
13. Halunen K, Kortelainen J & Kortelainen T (2010) Combinatorial multicollision attacks on generalized iterated hash functions. Proceedings of the 8th Australasian Information Security Conference – AISC 2010: 86–93.
14. Hoch J & Shamir A (2006) Breaking the ICE - finding multicollisions in iterated concatenated and expanded (ICE) hash functions. Proceedings of the 13th International Workshop on Fast Software Encryption – FSE 2006: 179–194.
15. Joux A (2004) Multicollisions in iterated hash functions. Application to cascaded construction., Proceedings of the 7th European Conference on Case-Based Reasoning – ECCBR 2004: 306–316.
16. Katz J & Lindell Y (2007) Introduction to modern cryptography. Boca Raton Chapman & Hall.
17. Kelsey J & Kohno T (2006) Herding hash functions and nostradamus attack. Proceedings of the 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2006: 183–200.
18. Kelsey J & Schneier B (2005) Second preimage on n-bit hash functions for much less than 2^n work. Proceedings of the 24th Annual International Conference on the Theory and

Applications of Cryptographic Techniques – EUROCRYPT 2005: 474–490.
19. Kortelainen J, Halunen K & Kortelainen T (2010) Multicollision attacks and generalized iterated hash functions. Journal of Mathematical Cryptology 4: 239–270.
20. Kortelainen J, Kortelainen T & Vesanen, A (2011) Unavoidable regularities in long words with bounded number of symbol occurences. Proceedings of the 17th Annual International Conference on Computing and Combinatorics – COCOON 2011: 519–530.
21. Kortelainen J, Kortelainen T & Vesanen A (2013) Unavoidable regularities in long words with bounded number of symbol occurrences. Journal of Combinatorial Optimization 26: 670–686.
22. Kortelainen T & Kortelainen J (2013) On diamond structures and Trojan message attacks. Proceedings of the 19th International Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 2013: 524–539.
23. Kortelainen T, Kortelainen J & Halunen K (2010) Variants of multicollision attacks on iterated hash functions. Proceedings of the 6th International Conference on Information Security and Cryptology – INSCRYPT 2010: 139–154.
24. Kortelainen T, Vesanen A & Kortelainen J (2012) Generalized iterated hash functions revisited: new complexity bounds for multicollision attacks. Proceedings of the 13th International Conference on Cryptology in India – INDOCRYPT 2012: 172–190.
25. Liskov M (2007) Constructing an ideal hash function from weak ideal compression functions. Proceedings of the 13th International Workshop on Selected Areas in Cryptography – SAC 2006: 358–375.
26. Lothaire M (2002) Algebraic combinatorics on words. Encyclopedia of Mathematics and its Applications 90. London Cambridge University Press.
27. Lucks S (2005) A failure-friendly design principle for hash functions. Proceedings of the 11th International Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 2005: 474–494.
28. Menezes A, van Oorschot P & Vanstone S (1996) Handbook of Applied Cryptography. Boca Raton CRC Press.
29. Merkle R (1990) One way hash functions and DES. Proceedings of the 9th Annual International Cryptology Conference – CRYPTO 1989: 428-446.
30. Nandi M & Stinson D (2007) Multicollision attacks on some generalized sequential hash functions. IEEE Transactions on Information Theory 53: 759–767.
31. Pin L, Wenling W, Chuankun W & Tian Q (2008) Analysis of zipper as a hash function. Proceedings of the 4th International Conference on Information Security Practice and Experience – ISPEC 2008: 392–403.
32. Suzuki K, Tonien D, Kurosawa K & Toyota K (2008) Birthday paradox for multicollisions. IEICE Transactions 91–A: 39–45.
33. Stevens M (2006) Fast Collision Attack on MD5. Cryptology ePrint Archive, Report 2006/104 URL:http://eprint.iacr.or/. Cited 2014/29/10
34. van Oorschot P & Wiener M (1999) Parallel collision search with cryptanalytic applications. Journal of Cryptology 12: 1–28.
35. Wang X, Yin Y & Yu H (2005) Finding collisions in the full SHA-1. Proceedings of the 25th Annual International Cryptology Conference – CRYPTO 2005: 17-36.
36. Wang X & Yu, H (2005) How to break MD5 and other hash functions. Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2005: 19–35.

37. Wendl M (2003) Collision probability between sets of random variables. Statistics & Probability Letters 64: 249–254.
38. Yu H & Wang, X (2007) Multi-collision attack on the compression function of MD4 and 3-pass HAVAL. Proceedings of the 10th International Conference on Information Security and Cryptology – ICISC 2007: 206–226.

ISBN 978-952-62-0642-4 (Paperback)
ISBN 978-952-62-0643-1 (PDF)
ISSN 0355-3191 (Print)
ISSN 1796-220X (Online)

UNIVERSITY of OULU
OULUN YLIOPISTO