Bauhaus-Universität Weimar

Analysis and Design of Blockcipher Based Cryptographic Algorithms

Inauguraldissertation
zur Erlangung des akademischen Grades
doctor rerum naturalium (Dr. rer. nat.)
der Bauhaus-Universität Weimar
an der Fakultät Medien

vorgelegt von
Ewan Fleischmann

Weimar, 2013

Gutachter:	Professor Dr. Stefan Lucks *Bauhaus-Universität Weimar*
	Dr. Martijn Stam, Lecturer *University of Bristol (Großbritannien)*
Mündliche Prüfung:	13. Mai 2013

For my loving wife Ursula
and my three children Alissa, Nils, and Jana

Acknowledgments

First of all I want to thank all the people who helped me to make this thesis successful. I would like to thank my PhD advisor Prof. Dr. Stefan Lucks. The mix of freedom and support Stefan gave me is unmatched and I'm very grateful for that. He always strived to keep organizational work for us students down to a minimum.

Another person that I'm indebted to is Martijn Stam. He not only agreed to serve as co-advisor for this thesis, but also gave me valuable and detailed feedback on some of my results.

I also want to thank my colleagues Christian Forler, Michael Gorski and Jakob Wenzel for good discussions, fun, friendship and proofreading.

Abstract

This thesis focuses on the analysis and design of hash functions and authenticated encryption schemes that are blockcipher based. We give an introduction into these fields of research – taking in a blockcipher based point of view – with special emphasis on the topics of double length, double call blockcipher based compression functions.

The first main topic (thesis parts I - III) is on analysis and design of hash functions. We start with a collision security analysis of some well known double length blockcipher based compression functions and hash functions: Abreast-DM, Tandem-DM and MDC-4. We also propose new double length compression functions that have elevated collision security guarantees. We complement the collision analysis with a preimage analysis by stating (near) optimal security results for Abreast-DM, Tandem-DM, and Hirose-DM. Also, some generalizations are discussed. These are the first preimage security results for blockcipher based double length hash functions that go *beyond* the birthday barrier.

We then raise the abstraction level and analyze the notion of 'hash

function indifferentiability from a random oracle'. So we not anymore focus on how to obtain a *good* compression function but, instead, on how to obtain a *good* hash function using (other) cryptographic primitives. In particular we give some examples when this strong notion of hash function security might give questionable advice for building a practical hash function.

In the second main topic (thesis part IV), which is on authenticated encryption schemes, we present an on-line authenticated encryption scheme, McOEx, that simultaneously achieves privacy and confidentiality *and* is secure against nonce-misuse. It is the first dedicated scheme that achieves high standards of security and – at the same time – is on-line computable.

Zusammenfassung

Die Schwerpunkte dieser Dissertation sind die Analyse und das Design von blockchiffrenbasierten Hashfunktionen (Abschnitte I-III) sowie die Entwicklung von robusten Verfahren zur authentifizierten Verschlüsselung (Abschnitt IV). Die Arbeit beginnt mit einer Einführung in diese Themengebiete, wobei – insbesondere bei den Hashfunktionen – eine blockchiffrenzentrierte Perspektive eingenommen wird.

Die Abschnitte I-III dieser Dissertation beschäftigen sich mit der Analyse und dem Design von Hashfunktionen. Zu Beginn werden die Kollisionssicherheit einiger wohlbekannter Kompressions- und Hashfunktionen mit zweifacher Blockchiffrenausgabelänge näher analysiert: Abreast-DM, Tandem-DM und MDC-4. Ebenso werden neue Designs vorgestellt, welche erhöhte Kollisionssicherheitsgarantien haben. Ergänzend zur Kollisionssicherheitsanalyse wird die Resistenz gegen Urbildangriffe von Kompressionsfunktionen doppelter Ausgabelänge untersucht. Dabei werden nahezu optimale Sicherheitsschranken für Abreast-DM, Tandem-DM und Hirose-DM abgeleitet. Einige Verallgemeinerungen sind ebenfalls Teil der Diskussion. Das sind

die ersten Sicherheitsresultate gegen Urbildangriffe auf blockchiffrenbasierte Kompressionsfunktionen doppelter Länge, die weit über die bis dahin bekannten Sicherheitsresultate hinausgehen.

Daran anschließend folgt eine Betrachtung, die auf einem erhöhten Abstraktionslevel durchgeführt wird und den Begriff der *Undifferenzierbarkeit einer Hashfunktion von einem Zufallsorakel* diskutiert. Hierbei liegt der Fokus nicht darauf, wie man eine *gute* Kompressionfunktion auf Basis anderer kryptographischer Funktionen erstellt, sondern auf dem Design einer Hashfunktionen auf Basis einer Kompressionsfunktion. Unter Einnahme eines eher praktischen Standpunktes wird anhand einiger Beispiele aufgezeigt, dass die relativ starke Eigenschaft der *Undifferenzierbarkeit einer Hashfunktion* zu widersprüchlichen Designempfehlungen für praktikable Hashfunktionen führen kann.

Im zweiten Schwerpunkt, in Abschnitt IV, werden Verfahren zur authentifizierten Verschlüsselung behandelt. Es wird ein neues Schema zur authentifizierten Verschlüsselung vorgestellt, McOEx. Es schützt gleichzeitig die Integrität und die Vertrauchlichkeit einer Nachricht. McOEx ist das erste konkrete Schema das sowohl robust gegen die Wiederverwendung von Nonces ist und gleichzeitig on-line berechnet werden kann.

Contents

Introduction

Modern Cryptography

Even in some contemporary publications, cryptology is described as the *art* of codifying messages, so that they become unreadable[1]. While being historically accurate, it does misrepresent the essence of modern cryptography: First, secret communication based on codes is only one of the current areas of activity, and, second, post-1980s cryptography distinguishes itself from the 'art' of classic cryptography by its emphasis on definitions, precise assumptions and rigorous proofs of security.

- **Definitions:** One of the most important contributions of modern cryptography is the development of formal definitions of security. These are crucial in the design of any cryptographic scheme: *If you do not precisely know what you want how should*

[1] A. Kahate, Cryptography and Network Security, McGraw-Hill, Second Edition 2008.

there be a measure if one has successfully achieved it?

- **Precise assumptions:** Essentially all known cryptographic schemes can *only* be proven secure when depending on some believed – yet unproven – assumptions. The modern cryptography approach states that these assumptions are completely defined.

- **Rigorous proofs of security:** The first and the second idea lay the groundwork for being able to use formal arguments showing that a cryptographic scheme is secure. Always with respect to clearly stated definitions and cryptographic assumptions. It is through this step that modern cryptography is nowadays more a science than an art.

One starts in modern cryptography by clearly *defining* the security target and stating the assumptions. An example of a common cryptographic assumptions in secret-key cryptography is *postulating the existence* of a *secure* so-called cryptographic *primitive* like, *e.g.*, a blockcipher or a compression function. Another example, taken from common public-key scenarios, is the postulated hardness, *i.e.*, the assumed practical unsolvability of a number theoretic problem like the factoring of a sufficiently large compound number. Also, the notion of *secure* is defined in more detail.

We then start from these atomic cryptographic primitives and transform them into schemes. Good atomic primitives are rare. Some of widely-used atomic block ciphers have never been convincingly broken, such as AES and DES[2]. But, on the contrary, most of the widely-used hash functions have been at least severely harmed or completely broken, such as, *e.g.*, MD4, MD5, SHA-1, and RIPEMD. It apparently seems that it is much more difficult to build good hash functions

[2] The best known practical attack on DES is still *complete key search* – although some attacks are known that are much better in theory. Due the to short key-size of 56-bit, DES is not recommended any more. But variants of DES, *e.g.*, Triple-DES, are still in use today: NIST considers Triple-DES with three independent keys to be appropriate through 2030 [131].

than it is to build good block ciphers. Consequently, an important effort in cryptography is to design new and efficient primitives and to analyze the old ones. However, this is *not* the part of cryptography we focus on in this thesis. One reason is that often the weak link in real-world cryptography seems to be between atomic primitives and schemes and not in the primitives themselves. It is in this transformation that the bulk of security flaws arise. And there is a science that can do something about it, namely, provable security.

Throughout this thesis, we view a cryptographer as someone who is responsible for turning atomic primitives into schemes. That is, we focus on scheme design under the assumption that good atomic primitives exist. Some examples of the kind of questions we are interested in are these: What is the best way to use a blockcipher to build a secure compression function that has a *sufficiently long* output? How secure are these methods? What can happen if some definitions get too strong for unambiguous '*what is best*' recommendations? How can we build robust ciphers that are on-line computable and guarantee privacy and authenticity at the same time by only relying on a blockcipher as a primitive.

A poorly designed scheme can be insecure even though the underlying atomic primitive is good. This is not the fault of the underlying atomic primitive, but that primitive was somehow misused. We would like to build on the strength of such primitives, especially block ciphers, in such a way that schemes can inherit this strength, not lose it. The work inside the proofs now essentially consists in providing a security reduction. A reduction shows that the only way to defeat the scheme is to break the underlying atomic primitive. A reduction is a proof that uses formal arguments to show that *if the atomic primitive does the job it is supposed to do, then the scheme does the job that it is supposed to do.* Believing this, it is no longer necessary to directly cryptanalyze the protocol: if you were to find a weakness in it, you would have found one in the underlying atomic primitive (or a flaw in the proof). So if one is going to do cryptanalysis, one might as well focus on the atomic primitive. And if we believe the latter is secure, then we know, without further cryptanalysis of the scheme, that the scheme is secure, too.

How to Read this Thesis

One way to read this thesis is, of course, sequentially from cover to cover. Apart from Chapter 1 where we introduce the concept of a hash function in greater detail and discuss some related work, we often provide theory and material on an as-needed basis. This helps keeping cross-dependency relations to a minimum. So, the proficient reader might well skip ahead and directly start with one of the four thesis parts. Parts I-III strongly rely on the contents introduced in Chapter 1 where the fourth part only lightly depends on it. However, we refer to the appropriate sections of Chapter 1 when we use the concepts for the first time.

Each of the four parts starts with a *results summary* (Chapters 2, 7, 9 and 11) aiming to provide an easily understandable overview of the results obtained by the theorems, proofs and discussions that follow.

Part I We give collision security results on some well-known blockcipher based hash functions as Abreast-DM, Tandem-DM, MDC-4 and some generalizations on them. Except for MDC-4, the analysis is for the compression function.

Part II (Near) Optimal preimage results are presented for Abreast-DM, Tandem-DM and Hirose-DM. Also, some generalizations are discussed and analyzed.

Part III In the third part, we make some observations on the well-known notion of 'indifferentiability from a random oracle' by taking in a hash function designer's point of view.

Part IV We present an new on-line authenticated encryption scheme, McOEx, that simultaneously achieves privacy and confidentiality and is secure against nonce-reuse.

Preliminary versions of the results obtained in this thesis have been published before in [3, 47, 51–54, 60–62, 99]. This material has been thoroughly revised and some deficiencies have been fixed. Other results in symmetric cryptography that are not considered in this thesis and have been published during my studies can be found in [39, 45, 46, 48–50, 55–59]. As they do not fit into the context of this thesis, we have not included them. A complete list of publications can be found at the back of this thesis on Pages 225-229.

1 Cryptographic Hash Functions

A cryptographic hash function is a function which maps an input of arbitrary length – usually called the message – to an output of fixed length. This output is often referred to as *hash value*, *message digest* or just *hash*. Hash functions are commonly used for public-key encryption, digital signatures, password protection, message authentication, key-derivation functions, pseudo-random number generators, etc. Recently, cryptographic hash functions have received a huge amount of attention due to the NIST SHA-3 contest [128]. This contest has been initiated since many of the prominent hash functions (*e.g.*, MD4 [142], MD5 [141], RIPEMD [71], SHA-0/1 [129]) have been successfully attacked in several ways, *e.g.*, [35, 70, 165, 166]. In 2012, the winner was announced (Keccak), now called SHA-3, as a successor of the currently still unbroken SHA-2 [130][1] hash standard.

In this thesis, we use the unkeyed notion of hash functions as proposed by Rogaway [147].

[1] Some even claim, that SHA-2 is so strong, that it might well serve as a backup for SHA-3. (D.J. Bernstein, Dagstuhl Seminar on Symmetric Cryptography, 2012).

Definition 1.1 (Hash Function). *A* hash function *is a function* $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^r$ *for some integer value* $r > 0$*.*

Practical designs of hash functions seem to come in two flavors: provably-secure constructions based on number-theoretic assumptions *or* highly-efficient constructions that are more heuristic in nature. Examples of the former types can be found in [4, 15, 29] and are based on problems from coding theory, number theory, or lattices. But only the heuristic hash functions are the ones used in practice. As it became apparent in the SHA-3 contest, and is especially true for most second-round and *any* final-round candidates, there is a new approach aiming to get the best of both worlds. In short, one tries to get as much provable security results for some macro structure as possible by assuming the security of one (small) atomic function working inside. And then thoroughly analyzing and arguing that this heuristically designed function is indeed at least as secure as claimed. We usually call this function a *cryptographic primitive*. The first part of the latter approach is also what this thesis is all about, since our proofs come to an end at the heuristic borderline.

1.1. Security Notions for Hash Functions

Most cryptographers expect a good hash function somehow to behave like a *random oracle* [12] which is restricted to a fixed output length. Such an oracle behaves as if it had chosen its outputs randomly from a set $\{0,1\}^r$, independently of the input, except that repeated queries are always treated consistently. A random oracle is a mathematical abstraction used in cryptographic proofs, hiding away virtually all real world and implementation details. They are typically used when no known implementable function provides the mathematical properties required for the proof – or when it gets too tedious to formalize these. A security proof of a cryptographic scheme using a random oracle as a component function is said to be in the *random oracle model.* From a theoretical point of view, it is clear that such a security proof is only a heuristic indication of the security of the scheme when instantiated with a particular hash function. In fact, many recent separation results [8, 27, 36, 68, 118, 127] illustrate that various cryptographic schemes are secure in the random oracle model but still completely insecure for *any* efficient instantiation of the random oracle. According to [96], all such counterexamples are 'artificial' and these results do not seem to attack any practically relevant scheme directly. Most cryptographers believe that a proof in the random oracle model at least shows that there are no structural flaws in the design of the scheme. In part III of this thesis, we discuss the issues one is faced with, when trying to design a practical hash function that is *as close as possible* to the merely abstract concept of a random oracle.

1.1.1. Defining Security

Apart from modeling a hash function as a random oracle, one can *define* some simpler (and strictly weaker) properties that it should provide. As for example stated in [120], a cryptographic hash function should at least be collision resistant, preimage resistant, and second-preimage resistant. These properties are always defined in terms of an algorithm that is called an *adversary.* We quantify the *insecurity* of a cryptographic function by the success probability of an optimal, resource-bounded adversary. Depending on the setting, different notions of success and different resource-bounds apply for the adversary. In this thesis, any analyzed cryptographic system is an algorithm that uses (at least one) other component function – the primitive – inside. As the adversary is assumed to have no knowledge on the inner workings of these primitives – in the past always formalized by assuming a secret key [147] – these are accessed by the adversary via an *oracle interface.* Such an oracle interface essentially formalizes the black-box mode of operation of an adversary towards the scheme or primitive being attacked. It provides a clearly defined set of exposed functions an adversary is able to send queries to and can expect to get an answer from. We always assume that such an adversary is computationally unbounded, but is given resource-bounded access to its oracles.

We now proceed with a short overview of the three most prominent, so called *standard-model*, security assumptions of hash functions: collision resistance, preimage resistance, and second-preimage resistance.

Collision Resistance This property formalizes the requirement that it should be difficult to find two different inputs that have the same hash value. We define the collision security of a hash function $\mathcal{H}$ by an experiment of an adversary $\mathcal{A}$ with a security parameter r.

Experiment 1.2 (Collision-Finding $\texttt{Exp-Coll}_{\mathcal{A},\mathcal{H}}(r)$).

1. *An adversary $\mathcal{A}$ is given oracle access to a hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^r$ and returns two values $x, x' \in \{0,1\}^*$.*
2. *The output of the experiment is defined to be 1 if and only if (iff) $x \neq x$ and $\mathcal{H}(x) = \mathcal{H}(x')$. In such a case we say that $\mathcal{A}$ has found a collision.*

The advantage of an adversary $\mathcal{A}$ finding such a collision of $\mathcal{H}$ is defined as

Definition 1.3. $\mathbf{Adv}_{\mathcal{H}}^{\mathrm{COLL}}(\mathcal{A}) = \Pr\left[\texttt{Exp-Coll}_{A,\mathcal{H}}(r) = 1\right].$

Since an adversary is only limited by the number of queries to its oracles, we can write

$$\mathbf{Adv}_{\mathcal{H}}^{\mathrm{COLL}}(q) := \max_{\mathcal{A}}\{\mathbf{Adv}_{\mathcal{H}}^{\mathrm{COLL}}(\mathcal{A})\},$$

where the maximum is taken over all adversaries that ask at most q oracle queries in total.

For an n-bit hash function the number of message pairs with q messages is $\binom{q}{2} = q(q-1)/2 \approx q^2/2$. An ideal r-bit hash function returns random r-bit strings. Since two of these are equal with probability 2^{-r}, one needs 2^r pairs before a collision can be expected. More precise, with $q = 2^{(r+1)/2}$ queries, the probability of a collision is $1 - 1/e \approx 0.63$. This generic attack works for any hash function [9] and is commonly known as the *birthday attack*.

Preimage Resistance Preimage resistance refers to the requirement that, given a hash value, it is difficult to find a message that hashes to this value. So finding a preimage for a hash function $\mathcal{H}$ refers to the ability of an adversary to find a value $x \in \{0,1\}^*$ for a specified value $y \in \{0,1\}^r$ such that $y = \mathcal{H}(x)$.[2] There are some notions known that formalize this case [149] by further clarifying how the value y is chosen. In this thesis, we adopt *everywhere preimage resistance* (epre) in the information-theoretic setting, which essentially lets the adversary pre-commit to the hash value y, it likes to be challenged on, *before* submitting any queries.

A method for finding preimages that works for any hash function is the brute force attack. For this, one hashes random messages until the hash value y is reached. Assuming that the output of the hash function is balanced, an adversary is expected to try 2^r messages in order to be successful.

Second-Preimage Resistance Finding a second-preimage describes the setting that an adversary is able to find a value $x' \in \{0,1\}^*$ for a given value $x \in \{0,1\}^*$ such that $\mathcal{H}(x) = \mathcal{H}(x')$. A brute force preimage attack can also be used to find a second-preimage. One simply ignores the message in the domain range that was used to

[2] We formally define a preimage experiment for compression functions on page 14 in this introductory chapter, but skip the experiment for hash functions since we do not need it in this thesis.

obtain the challenge hash value. By selecting messages at random and assuming that the domain is much larger than the co-domain, the probability of the second-preimage being identical to the first is negligible - therefore one can ignore this possibility. So finding a second-preimage seems to be never harder than finding a preimage, except for maliciously designed hash functions [120, Note 9.20].

1.2. Iterated Hash Functions

We defined hash functions as being able to take an input of arbitrary length and produce an output of fixed length. Since it seems to be very difficult to develop directly a function being able to handle efficiently variable length inputs, all known hash functions are based on a *compression function* that takes inputs of fixed size. Then, a domain-extending transform is used in order to get a full-fledged hash function. Lai and Massey [103] first called these types of hash functions *iterated*. These transforms obtain their popularity by being property-preserving, which means that certain properties of the compression function are transferred to the hash function. By far the most prominent transform technique has been named by its inventors, Merkle-Damgård (MD) [34, 121] which preserves *collision-resistance*. Since, in recent years, research has put a focus on designing constructions that preserve as many properties of the compression function as possible, many more constructions are known, *e.g.*, [1, 2, 11, 28, 44].

Due to its historic relevance, we discuss the Merkle-Damgård transform in detail in Section 1.2.2 after introducing some basics on compression functions.

1.2.1. Cryptographic Compression Functions

A cryptographic *compression function* transforms a fixed length input to a smaller, fixed length, output.

Definition 1.4 (Compression Function). *A* compression function *is a function $H : \{0,1\}^m \times \{0,1\}^r \to \{0,1\}^r$ for some integer values $m, r > 0$.*

Similar to hash functions, the most common security scenarios for cryptographic compression functions are collision resistance, preimage resistance and second-preimage resistance.

Collision Resistance We again define the security by an experiment with a security parameter r.

Experiment 1.5 (Collision-Finding $\texttt{Exp-Coll}_{\mathcal{A},H}(r)$).

1. *An adversary $\mathcal{A}$ is given oracle access to a compression function $H : \{0,1\}^m \times \{0,1\}^r \to \{0,1\}^r$ and returns two values $x, x' \in \{0,1\}^m \times \{0,1\}^r$.*
2. *The output of the experiment is defined to be 1 iff $x \neq x'$ and $H(x) = H(x')$. In such a case we say that $\mathcal{A}$ has found a collision.*

The advantage of an adversary $\mathcal{A}$ finding such a collision of H is defined as

Definition 1.6. $\mathbf{Adv}_H^{\mathrm{COLL}}(\mathcal{A}) = \Pr\left[\texttt{Exp-Coll}_{\mathcal{A},H}(r) = 1\right]$.

Since, as discussed, we limit an adversary only by the number of queries to its oracles, we write

$$\mathbf{Adv}_H^{\mathrm{COLL}}(q) := \max_{\mathcal{A}}\{\mathbf{Adv}_H^{\mathrm{COLL}}(\mathcal{A})\},$$

where the maximum is taken over all adversaries that ask at most q oracle queries in total.

Preimage Resistance We let $\mathbf{Adv}_H^{\mathrm{EPRE}}(\mathcal{A})$ be the predicate that is true iff the experiment $\texttt{Exp-Epre}_{\mathcal{A},H}(r)$ returns 1. The pre-commited preimage value y is an omitted parameter of $\mathbf{Adv}_H^{\mathrm{EPRE}}(\mathcal{A})$.

Experiment 1.7 (Preimage-Finding $\texttt{Exp-Epre}_{\mathcal{A},H}(r)$).

1. *An adversary $\mathcal{A}$ is given oracle access to a compression function $H : \{0,1\}^m \times \{0,1\}^r \to \{0,1\}^r$. $\mathcal{A}$ selects and announces a point $y \in \{0,1\}^r$ before making any oracle queries. It outputs a value $x \in \{0,1\}^m \times \{0,1\}^r$.*

2. *The output of the experiment is defined to be 1 iff $H(x) = y$. In such a case we say that $\mathcal{A}$ has found a preimage.*

The advantage of an adversary $\mathcal{A}$ finding such a preimage of H is defined as

Definition 1.8. $\mathbf{Adv}_H^{\mathrm{EPRE}}(\mathcal{A}) = \Pr\left[\texttt{Exp-Epre}_{\mathcal{A},H}(r) = 1\right]$.

Again, we define

$$\mathbf{Adv}_H^{\mathrm{EPRE}}(q) := \max_{\mathcal{A}}\{\mathbf{Adv}_H^{\mathrm{EPRE}}(\mathcal{A})\},$$

where the maximum is taken over all adversaries that ask at most q oracle queries in total.

Second-Preimage Resistance Finding a second-preimage describes the setting that an adversary is able to find a value $x' \in \{0,1\}^m \times \{0,1\}^r$ for a given value $x \in \{0,1\}^m \times \{0,1\}^r$ such that $H(x) = H(x')$. Since the related experiment and predicate is straightforward – and is not needed in this thesis – we omit it here.

1.2.2. Merkle-Damgård Iterated Hash Functions

At Crypto'89, Damgård [34] and Merkle [121] independently proposed an iterative structure to construct a collision resistant hash function accepting arbitrary length inputs while utilizing only a fixed-input length compression function. This iterated design has been called the *Merkle-Damgård* (MD) construction. It has influenced the design of virtually all popular hash functions such as MD4 [142], MD5 [141], SHA-0/1 [129], and the SHA-2 family [130].

Definition 1.9. *Let H be as in Definition 1.4. Fix an initial value $V_0 = IV \in \{0,1\}^r$. Let $M = (M_1, M_2, \ldots, M_\ell)$ be the message input where $M_i \in \{0,1\}^m$ for $1 \leq i \leq \ell$. The* iterated hash function *$\mathcal{H}$ is the hash function $\mathcal{H} : (\{0,1\}^m)^* \longrightarrow \{0,1\}^r$ returning the hash value V_ℓ which is computed by $V_{i+1} = H(M_{i+1}, V_i)$, for $0 \leq i < \ell$.*

Since the total number of bits of a message M, denoted by $|M|$, is not necessary a multiple of m, a *padding procedure* is usually specified. This padding procedure can also be applied if $|M|$ is a multiple of m since it can serve as a pre-processing function. This additional stage is sometimes called *message expansion*. The most common padding procedure is as follows:

> 10^***-Padding-Rule.** The message is padded on the right with a 1 followed, if necessary, by 0's until the last block becomes complete. If the last block is complete, a new block is added to the message, equal to a 1 followed by 0's.

There are numerous other padding rules known and the choice depends on the applications. More examples are discussed in [136, Chap. 2.4.1]. As we will see now, the length of the message might also be included into the padding as a security measure.

Damgård and Merkle independently provided theorems in their papers [34, 121] that essentially show Theorem 1.10.

Theorem 1.10 (Merkle-Damgård). *If the IV is fixed and if the padding procedure includes the length of the input into the padding bits, then $\mathcal{H}$ is collision resistant if H is collision resistant.*

Fixing the IV and adding a representation of the message length is called MD - *strengthening*.

A related work by Lai and Massey [103] considered the relation between the compression function and the iterated hash function with respect to second-preimage resistance.

Theorem 1.11 (Lai-Massey). *Let IV be fixed and let the padding procedure include the length of the input into the padding bits. Let the input M be at least two blocks, excluding padding. Then, finding a second-preimage for the iterated hash function $\mathcal{H}$ requires 2^r operations if and only if finding a preimage for the compression function H requires 2^r operations (with arbitrary chaining input).*

1.2.3. Attacks on Iterated Hash Functions

Although the Merkle-Damgård construction provably preserves the collision resistance of the compression function, some other properties are not inherited by $\mathcal{H}$ from H. There are currently some attacks known that exploit the iterated structure of the hash function, but *do not* utilize special properties of the compression function itself. Examples are multicollisions [84], the *herding* attack [87] or a second-preimage attack [88]. It seems that the MD transform does not have all the desired properties one demands from a general purpose hash function.

Apart from these structural attacks, one can always try to exploit the detailed inner workings of the compression functions. Often, but not always, compression function weaknesses harm the hash function built on it as well.

1.2.4. Ideal World Model for Iterated Hash Functions

Motivated by the practical need to *'say anything about structural flaws in the design of the hash function $\mathcal{H}$ itself'*, Coron et al. [30] presented a new notion of security for cryptographic hash functions which is called *indifferentiability.* It serves as a method to compare an iterated hash function with a random oracle. In short, if one models the compression function(s) as random oracles with fixed-size inputs, then the iterated hash function composed from these compression functions should be *indifferentiable from a random oracle* with variably-sized inputs. They propose this as a practically relevant criterion, *e.g.*, to separate practical hash functions with a good structure from those which might suffer from structural flaws, especially with regard to the SHA-3 [91, 132] contests. The neat thing about indifferentiability is that, if one proves that a hash function $\mathcal{H}$ *is* indifferential to a random oracle, one can replace the random oracle with this function and the whole scheme remains secure.

There are also three other *comparison methods* known and discussed in literature: preimage awareness, indifferentiability from a public-use random oracle and indistinguishability. A more detailed discussion of these techniques is given in Part III of this thesis, where we analyze some practically relevant issues of the notion *indifferentiability from a random oracle.*

1.3. Compression Functions Based on Block Ciphers

A *blockcipher* is a combined encryption/decryption cryptographic primitive. Most hash functions in use today are based on a blockcipher, but they are often not considered as such, since the blockcipher was designed for the hash function specifically, and was *never* intended to be used for encryption. We call these hash functions *dedicated hash functions*. This is in contrast to *blockcipher based hash functions* which only make use of an existing – general purpose – blockcipher. An advantage of dedicated hash function is that they are likely to be more efficient. Also, the use of block ciphers for purposes they were not designed for may reveal weaknesses which are not relevant in the case of encryption.

But despite these apparent disadvantages, there are several reasons for the recent resurgence of interest in blockcipher based hash function designs and their analysis. Presumably the most important one being the already discussed breakdown of virtually any widely-employed hash function which has stimulated researchers to look for alternatives. Generic blockcipher based constructions seem promising since they are very well known – they even predate the dedicated block-cipher based MD4-approach [115]. Also, many of the SHA-3 designs like Skein [43], SHAvite-3 [17], and SIMD [109] use blockcipher based instantiations. Another reason for the resurgence of interest in blockcipher based hash functions is due to the rise of resource restricted devices such as RFID tags or smart cards. A hardware designer only needs to implement a blockcipher in order to obtain an encryption function as well as a hash function.

1.3.1. Blockciphers

A (k, n) *blockcipher* is a keyed family of permutations consisting of two paired algorithms $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and $E^{-1} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ both accepting a key of size k bits and an input block of size n bits for some $k, n > 0$. For positive values k, n, $\textsc{Block}(k, n)$ is the set of all (k, n) block ciphers. For any $E \in \textsc{Block}(k, n)$ and any fixed key $K \in \{0, 1\}^k$, decryption $E_K^{-1} := E^{-1}(K, \cdot)$ is the inverse function of encryption $E_K := E(K, \cdot)$, so that $E_K^{-1}(E_K(X)) = X$ holds for any admissible input $X \in \{0, 1\}^n$.

Security Goals

Ideally, one models a blockcipher as a family of random permutations $\{E_K\}$, whereas the random permutations are chosen independently for each key K, *i.e.*, formally any E_K is selected at random from $\textsc{Block}(k, n)$. This setting is called the *ideal cipher model* [19, 42, 90]. Clearly, no practical blockcipher *is* an ideal cipher. Similar to the random oracle model, there are also – arguably maliciously designed – separation results known [18].

A 'weaker' variant to saying that a blockcipher is modeled as an ideal cipher is given by the common notion of a *pseudorandom permutation* (PRP). Intuitively, we say that a blockcipher is a PRP if no adversary running in polynomial time exists, which can distinguish between an 'ideal cipher' and the actual cipher E_K – assuming a randomly chosen secret key K that is not known to the adversary. So a PRP is a computational relaxation of the purely abstract concept of an ideal cipher. We note that we skip a formal definition of this classic PRP security notion, since we do not require it for our later discussion. A considerably stronger security notion than a PRP is a *RK-PRP* which is a PRP also secure against *related-key attacks*. In short, we say that a cipher is a RK-PRP, if the cipher is a PRP and remains secure if we give the adversary some control on key-differences used for encryption/decryption. The related key notions relevant in this thesis are given in Definition 1.12 using the concepts of Section 1.3.1.

Attack Settings

In order to assess the security of any given blockcipher, one usually considers an adversary having access to some *information* (*i.e.*, the power of the adversary) trying to achieve a specific *goal* (*e.g.*, showing that an efficient distinguisher between this cipher and a PRP exists). Common information settings of an adversary are (in order of severity):

- **Ciphertext-only** This is the most basic type and refers to the scenario where the adversary just observes ciphertexts.
- **Known Plaintext (KP)** Here, the adversary learns one or more pairs of plaintexts/ciphertexts encrypted under the same key.
- **Chosen Plaintext (CP)** In this scenario, the adversary has the ability to obtain the encryption of plaintexts of its choice.
- **Chosen Ciphertext (CC)** The final type is one where the adversary is given the capability to not only obtain encryptions of plaintexts, but also decryptions of ciphertexts of its choice.

The first two types are called *passive* adversaries, the last two types are called *active* adversaries.

Capabilities of an Active Adversary The adversary may make a *forward* query $(K, X)_{fwd}$ to discover the corresponding value $Y = E(K, X)$, or the adversary may make a *backward* query $(K, Y)_{bwd}$, so as to learn the corresponding value $X = E^{-1}(K, Y)$ such that $E(K, X) = Y$. Either way, the result of the query is stored in a triple $(K_i, X_i, Y_i) := (K, X, Y)$ and the *query history* $\mathcal{Q}$ is the tuple $(Q_1, \ldots, Q_q)$, where $Q_i = (K_i, X_i, Y_i)$ and q is the total number of queries made by the adversary. Without loss of generality, we always assume that $\mathcal{A}$ asks at most only once on a triplet of a key K_i, a plaintext X_i, and a ciphertext Y_i obtained by a query and the corresponding reply.

Definition 1.12. *Let $E \in Block(k, n)$ and denote by E^{-1} the corresponding inverse. Let $\varphi : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^k$. A fixed related key adversary $\mathcal{A}$ has access to an E oracle with two parameters such that it can query either $E_{\varphi(K,\cdot)}(\cdot)$ or its inverse. Let $\pi \in Block(n, n)$ and π^{-1} denote the corresponding reverse. By writing oracles that are given to an adversary as superscripts, we define the related-key (RK) advantage [112] of $\mathcal{A}$ in breaking E as*

$$\begin{aligned}\mathbf{Adv}_E^{\text{CPA-RKPRP}}(\mathcal{A}) &= \big|\Pr[K \xleftarrow{\$} \{0,1\}^k : \mathcal{A}^{E_{\varphi(K,\cdot)}(\cdot)} \Rightarrow 1] \\ &\quad - \Pr[\pi \xleftarrow{\$} \textsc{Perm}(n,n) : \mathcal{A}^{\pi(\cdot,\cdot)} \Rightarrow 1]\big|, \\ \mathbf{Adv}_E^{\text{CCA-RKPRP}}(\mathcal{A}) &= \big|\Pr[K \xleftarrow{\$} \{0,1\}^k : \mathcal{A}^{E_{\varphi(K,\cdot)}(\cdot), E^{-1}_{\varphi(K,\cdot)}(\cdot)} \Rightarrow 1] \\ &\quad - \Pr[\pi \xleftarrow{\$} \textsc{Perm}(n,n) : \mathcal{A}^{\pi(\cdot,\cdot),\pi^{-1}(\cdot,\cdot)} \Rightarrow 1]\big|.\end{aligned}$$

1.3.2. Single Length Compression Functions

Generally speaking, a *single length* (SL) blockcipher based compression function is a compression function $H : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ using a blockcipher with n-bit block size inside. The idea was first discussed in literature by Rabin [115]. Most SL functions use a blockcipher from $\textsc{Block}(n, n)$ and compress a $2n$-bit string to an n-bit string. Popular examples are the Davies-Meyer (DM) [167] mode

$$H(M, U) = E_M(U) \oplus U,$$

the Matyas-Meyer-Oseas (MMO) [117] mode

$$H(M, U) = E_U(M) \oplus M,$$

and the Miyaguchi-Preneel mode (MP) [123, 136] mode

$$H(M, U) = E_U(M) \oplus M \oplus U.$$

The $\oplus$ operation is usually called *feed-forward.* Any of the three examples have been proven to be collision resistant and/or one-way in the publications referenced above. But clearly, there are a lot more simple combinations possible, namely 64, by choosing for plaintext, ciphertext and the feed-forwarded value either the chaining value U, the message M, $U \oplus M$, or a constant. These possibilities were first systematically studied by Preneel et al. [138], which found attacks on any of them – except 12 combinations. Among these 12 apparently secure ones were DM, MMO and MP. A rigorous and proof-centric analysis of these 64 modes was given by Black et al. [19], who not only confirmed the analysis of the secure 12 but added another 8 constructions to the list to be *secure in the iteration*. This means that there are attacks known on the compression function itself, but if used *inside* of an iterated hash function these 8 might also be safe to use. The analysis in [19] is in the ideal cipher model just as any hash function security proof in this thesis.

Practical Relevance

Assume an iterated mode of operation with U being the chaining value and M the message. The DM mode has the advantage of knowing the key of the block ciphers used in advance which may lead to faster implementations since the key-scheduler can be run in parallel prior to having access to the chaining value result of the foregoing computation. This advantage can also be seen as a disadvantage since the adversary then has complete control over the key. Also, DM does not require U and M to be of equal length.

For example, DM is used in the SHA-2 hash function family [130], MMO is the basis of the MDC-2 and MDC-4 double length constructions discussed in Section 1.4 and MP is utilized in Whirlpool [139] and N-Hash [123].

Single Length (SL) Framework

Stam introduced a framework [160] that aims to provide more insight into SL compression functions. It works as follows. Given a message $M \in \{0,1\}^k$ and a chaining value $U \in \{0,1\}^n$ and two functions, for pre-processing and post-processing,

$$\begin{aligned} \mathrm{C}^{PRE} :&\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^k \times \{0,1\}^n \text{ and} \\ \mathrm{C}^{POST} :&\{0,1\}^k \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n, \end{aligned}$$

the new chaining value V is computed by:

1. Prepare key and plaintext: $(K, X) \leftarrow \mathrm{C}^{PRE}(M, U)$.
2. Set $Y \leftarrow E_K(X)$.
3. Output $V \leftarrow \mathrm{C}^{POST}(M, U, Y)$.

Since an adversary might also be asking for inverse queries, a modified post-processing function $\mathrm{C}^{AUX}(K, X, Y) = \mathrm{C}^{POST}(\mathrm{C}^{-PRE}(K, X), Y)$ is also defined for convenience. In the case of C^{PRE} being bijective, $\mathrm{C}^{-PRE}(\cdot)$ denotes the inverse function of $\mathrm{C}^{PRE}(\cdot)$. For example, choosing $|X| = |K| = |M| = |U| = |V| = |Y| = n$ and a blockcipher $E \in \textsc{Block}(n, n)$, the 12 secure SL schemes can be derived for appropriate choices of C^{PRE} and C^{POST}. In fact, Stam's analysis also covers more general choices by allowing values larger than the block size n. This framework, for a blockcipher $E \in \textsc{Block}(n, n)$, is shown in Figure 1.1. Given a query (K, X, Y), we call $V = \mathrm{C}^{POST}(\mathrm{C}^{-PRE}(K, X), Y)$ the post-output of that query.

The SL compression functions have been categorized [19, 160] into the following three groups:

(1) Type-I: The compression function is collision- and preimage resistant (12 out of 64).

(2) Type-II: The compression function is 'secure' only in the iteration (8 out of 64).

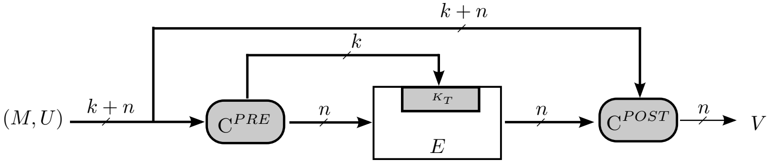


Figure 1.1.: Single Length Compression Function, $E \in \textsc{Block}(k,n)$. The gray notch inside the cipher rectangle indicates the input used as a key.

(3) Insecure: The compression function is not secure, *i.e.*, there are attacks known.

In order to get a Type-I compression function, Stam proved that it is sufficient to show that C^{PRE}, $\mathrm{C}^{POST}(M,U,\cdot)$ and $\mathrm{C}^{AUX}(K,\cdot,Y)$ are bijective.

1.4. Double Length Compression Functions

Due to the short output length of most practical block ciphers, it became evident that – for typical block ciphers and security expectations – the hash function needs to output a digest that is considerably larger than the blockcipher's block size. As a result, many proposals of double length (DL) hash functions have been published in the last 25 years. Such double length hash functions use a blockcipher with n-bit output as the building block by which they map possibly long strings to $2n$-bit ones.

From an efficiency point of view, double length compression functions that only require one blockcipher invocation for any hashed message block are desirable. We define the *rate* of a blockcipher based compression function simply as the number of message blocks that are processed by a compression function divided by the number of blockcipher invocations needed for this. This definition of a rate is only a rough estimate of efficiency since some important aspects – for example the number of key schedule runs or additional operations over $GF(2^n)$ – have been completely neglected. Also, this definition of a *rate* does not take the actual block size, the key size, or some relations among those into account.

Any hash function design based on blockciphers from $\textsc{Block}(n, n)$ with a rate of *one* has been broken [92], including Parallel Davies-Meyer [75], PBGV [137], and the LOKI DBH mode [24]. For single call constructions using a blockcipher with a longer key, *e.g.*, chosen from $\textsc{Block}(2n, n)$, some are known which are provably collision resistant. These do require some Galois field multiplication in their processing, *e.g.*, [114, 160] – a simple XOR-operation does not suffice.

There are four classical – somewhat unbroken and all with rate < 1 – double length blockcipher based hash function constructions known: MDC-2, MDC-4, Abreast-DM and Tandem-DM. We now give some details on the inner workings of these four schemes. Additionally, we also discuss a more current one, named by its inventor, Hirose-DM.

MDC-2 and MDC-4 MDC-2 and MDC-4 are acronyms for Modification Detection Code, with rate 1/2 and 1/4 respectively, and were both developed in the late eighties by Brachtl *et al.* [22]. They are also known as the Meyer-Schilling hash functions after the authors of the first paper describing them [122]. Both were originally specified for the blockcipher DES [125]. They were patented by IBM [22] and MDC-2 was standardized in ISO/IEC 10118-2 [77].

As [162], we consider a slightly generalized version of MDC-2 by ignoring an additional bit-fixing step that was used in the original specification as a security measure to avoid some DES specific key issues. Let $U, \widehat{U}$ be two n-bit chaining values, $E \in \textsc{Block}(n, n)$, and M an n-bit message block. The compression function MDC-2 (cf. Figure 1.2) is now computed as follows, where S^L and S^R denote the left and right halves of a string S.

1. $S = (S^L \| S^R) \leftarrow E_U(M) \oplus M$

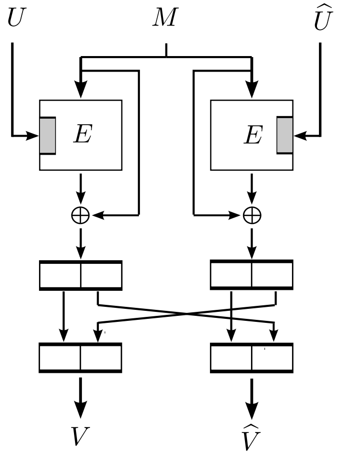


Figure 1.2.: Compression Function of MDC-2. The gray notch inside the cipher rectangle indicates the input to the blockcipher that is used as a key.

2. $T = (T^L \| T^R) \leftarrow E_{\widehat{U}}(M) \oplus M$
3. $V \leftarrow (S^L \| T^R)$
4. $\widehat{V} \leftarrow (T^L \| S^R)$

The compression function MDC-2 can only be secure in the iteration since $U = \widehat{U}$ essentially renders the double length compression function into a single length compression function. Steinberger showed [83, 162] that, in the ideal cipher model, the hash function MDC-2 is collision resistant up to at least $2^{3/5n}$ blockcipher calls. In [94] Knudsen *et al.* gave a collision attack requiring $2^{124.5}$ blockcipher calls (assuming $n = 128$) and a time-memory trade off preimage attack with $time \cdot memory = 2^{2n}$.

MDC-4 is an up-scaled variant of MDC-2 that uses twice as much blockcipher calls. MDC-4 was intended to offer a higher security mar-

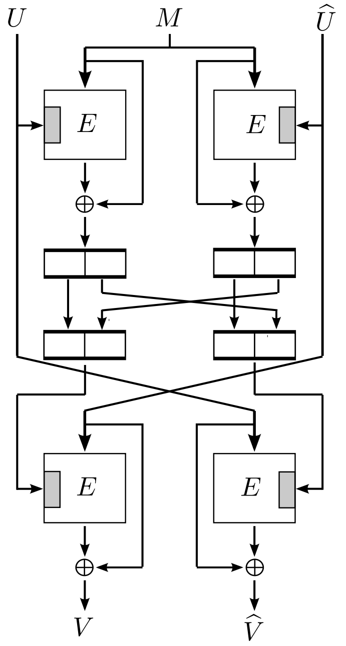


Figure 1.3.: Compression Function of MDC-4

gin than MDC-2 and has a rate of 1/4, *i.e.*, four blockcipher calls are needed for processing one message block. The MDC-4 hash function is part of the IBM CLiC cryptographic module [31]. Assuming again two n-bit chaining values $U, \widehat{U}$, a blockcipher $E \in \textsc{Block}(n, n)$, and an n-bit message block M, the compression function of MDC-4 (cf. Figure 1.3) works as follows.

1. $S = (S^L \| S^R) \leftarrow E_U(M) \oplus M$

2. $T = (T^L \| T^R) \leftarrow E_{\widehat{U}}(M) \oplus M$

3. $V \leftarrow E_{S^L \| T^R}(\widehat{U}) \oplus \widehat{U}$

4. $\widehat{V} \leftarrow E_{T^L \| S^R}(U) \oplus U$

In Chapter 6 we give the first collision security bound for MDC-4 showing that no adversary asking less than $2^{74.76}$ queries to the blockcipher E can find a collision with probability greater than 1/2 for $n = 128$.

Abreast-DM and Tandem-DM Proposed at EUROCRYPT '92 by Xuejia Lai and James L. Massey [103], both compression functions incorporate two Davies-Meyer (DM) single block length compression functions [120], which are used side-by-side. Each of them uses two calls to a blockcipher from $\textsc{Block}(2n, n)$ for hashing one message block meaning that they have a rate of 1/2. For Abreast-DM, the two blockcipher invocations can be computed in parallel whereas for Tandem-DM they have to be computed one after another. Assuming again two n-bit chaining values $U, \widehat{U}$ and an n-bit message block M, the algorithms output the new $2n$-bit chaining value $(V, \widehat{V})$ as follows (cf. Figures 1.4 and 1.5).

Abreast-DM H^{ADM}

1. $V \leftarrow E_{\widehat{U} \| M}(U) \oplus U$
2. $\widehat{V} \leftarrow E_{M \| U}(\widehat{U} \oplus 1^n) \oplus \widehat{U}$

Tandem-DM H^{TDM}

1. $V \leftarrow E_{\widehat{U} \| M}(U) \oplus U$
2. $\widehat{V} \leftarrow E_{M \| V \oplus U}(\widehat{U}) \oplus \widehat{U}$

We gave the first (flawed) collision security proof for Tandem-DM [60] which can be found in a corrected[3] version in Section 5.3.1. There we essentially state that, again in the case $n = 128$, no adversary asking less than $2^{119.6}$ queries can find a collision with probability greater than 1/2. The best collision security bound known for Tandem-DM today is given in [107] with $2^{120.87}$.

Hirose [73] gave the first collision security analysis for Abreast-DM, but the proof used the simplifying assumption of two independent block ciphers. We give in Chapter 4 the first collision security bound

[3] details on the corrections are also discussed in Section 5.3.1

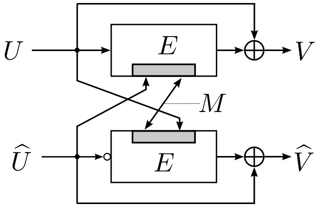


Figure 1.4.: Abreast-DM ('○' denotes a bit-by-bit complement)

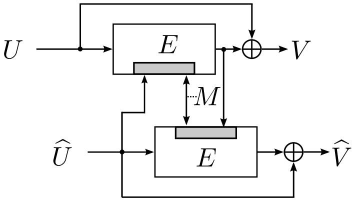


Figure 1.5.: Tandem-DM

for Abreast-DM that works in the case of one blockcipher. We generalize these techniques to be applicable to a more generic design, Cyclic-DL, which we introduce shortly. Using this design, we are able to give new double length compression functions that have higher collision security guarantees. Most of these results have been published in [61].

For Abreast-DM, a weak birthday-type *preimage* security bound is also known, $6q/(2^n - 6q)^2$ [105], which becomes void for $q \geq 2^{n-\log_2 6}$. The first preimage security bounds for Abreast-DM and Tandem-DM that go beyond the birthday barrier, *i.e.*, allowing the adversary to ask more than 2^n queries, are given in the second part of this thesis and were published in [3]. We show that no adversary asking less than $O(2^{2n-10})$ queries to the blockcipher can find a preimge with probability greater than 1/2.

Hirose-DM Hirose introduced a $3n$-bit to $2n$-bit compression function [74] making two calls to a blockcipher using a $2n$-bit key during message processing – more than ten years after Abreast-DM and Tandem-DM had been published. We call this construction Hirose-DM (cf. Figure 1.6). It is simpler than either Abreast-DM or Tandem-DM and uses only a single keying schedule for the 'top' and 'bottom' block ciphers.

Hirose's scheme

1. $V \leftarrow E_{\widehat{U}\|M}(U) \oplus U$

2. $\widehat{V} \leftarrow E_{\widehat{U}\|M}(U \oplus c) \oplus U \oplus c$

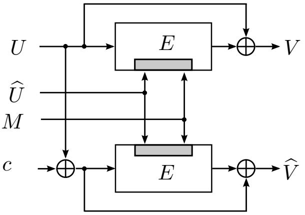


Figure 1.6.: Hirose-DM

As for Abreast-DM and Tandem-DM, we let $E \in \textsc{Block}(2n, n)$, $(U, \widehat{U})$ be a $2n$-bit chaining value, M be an n-bit message block, $(V, \widehat{V})$ the new $2n$-bit chaining value output, and c a non-zero n bit value.

Hirose himself already proved a birthday-type collision resistance bound of $2^{124.55}$ for his construction in the ideal cipher model (evaluated for $n = 128$), thereby pre-dating the collision resistance analyses for Abreast-DM and Tandem-DM. For preimage security, a birthday-type bound is also known, $2q/(2^n - 2q)^2$ [105] – which again becomes void once $q \geq 2^{n-1}$. Similar as for Abreast-DM and Tandem-DM, we also give for Hirose-DM the first preimage security bound in Section 8.2 that shows a near-perfect preimage security of $O(2^{2n-5})$. This result has also been published in [3].

More Related Work Merkle [121] presented three DBL hash functions composed of DES with rates of at most 0.276. They are optimally collision resistant in the ideal cipher model. Hirose [73] gave a class of DBL hash functions with rate 1/2 which are composed of two different and independent $(2n, n)$ block ciphers that have birthday-type collision resistance. In [104], Lee and Kwon independently provided a similar security bound for Abreast-DM. They also stated a generalized Theorem (without proof) and gave a compression function with a security guarantee of up to 2^{125} queries, which does not have a common key for both encryption functions. Knudsen *et al.* [93] discussed the insecurity of DBL hash functions with rate 1 composed of (n, n) block ciphers. Hohl *et al.* [75] analyzed the security of DBL compression functions with rate 1 and 1/2. Satoh *et al.* [157] and Hattoris *et al.* [72] discussed DBL hash functions with rate 1 composed of $(2n, n)$ block ciphers. Nandi *et al.* [124] proposed a construction with rate 2/3 but it is not optimally collision resistant. In [95], Knudsen and Muller presented some attacks against it. Gauravaram *et al.* [64] proposed a new approach based on iterated halving to design a hash function with a blockcipher. Steinberger [152, 153] proved some security bounds for fixed-key (n, n) blockcipher based hash functions, *i.e.*, permutation based hash functions, that all have small rates and low collision security bounds. Lee and Stam [106] gave a scheme similar to MDC-2, called MJH. It uses finite field multiplications to offer a collision security bound in the iteration of $O(2^{2n/3-\log n})$. Bos *et al.* [21] provided practical performance figures for some double length hash functions using the AES-NI instruction set.

1.4.1. Double Length Double Call Framework

Our starting point for analyzing double length double call compression functions is the recent framework for double length blockcipher based compression functions given by Özen and Stam [133], which is a natural extension of the SL framework [160]. We adapt this double length framework to be able to handle what we call *serial* double length constructions since their framework could not handle compression functions as complex as Tandem-DM. Examples of double length compression functions covered by our framework [47] are Tandem-DM, Abreast-DM, and Hirose-DM, but not MDC-2 or MDC-4 since we assume a blockcipher from $\text{Block}(k, n)$ with $k > n$. A compression function $H^{\text{DL}} : \{0,1\}^{k-n} \times \{0,1\}^{2n} \to \{0,1\}^{2n}$ is called *double length double call* (DL) when it requires exact two invocations of E for computing the $2n$-bit output; therefore compressing an $k - n + 2n = n + k$ bit string to a $2n$-bit string.

Formally, $(V, \widehat{V}) := H^{\text{DL}}(M, U, \widehat{U})$ for a given $(k-n)$-bit message M and a $2n$-bit chaining value $(U, \widehat{U})$ is computed using preprocessing and post-processing functions $\mathrm{C}_T^{PRE}, \mathrm{C}_B^{PRE} : \{0,1\}^{k+n} \to$

$\{0,1\}^{k+n}$ and $\mathrm{C}_T^{POST}, \mathrm{C}_B^{POST} : \{0,1\}^{k+2n} \rightarrow \{0,1\}^n$ as well as a linking function $\mathrm{C}^{LNK} : \{0,1\}^{k+2n} \rightarrow \{0,1\}^{k+n}$ by:

1. $(K_T, X_T) \leftarrow \mathrm{C}_T^{PRE}(M, U, \widehat{U})$
2. $Y_T \leftarrow E_{K_T}(X_T)$
3. $L \leftarrow \mathrm{C}^{LNK}(M, U, \widehat{U}, Y_T)$, $L \in \{0,1\}^{k+n}$
4. $(K_B, X_B) \leftarrow \mathrm{C}_B^{PRE}(L)$
5. $Y_B \leftarrow E_{K_B}(X_B)$
6. Output $(V, \widehat{V}) \leftarrow (\mathrm{C}_T^{POST}(M, U, \widehat{U}, Y_T), \mathrm{C}_B^{POST}(L, Y_B))$.

This framework is depicted in Figure 1.7. Using the notations given there, we say that a blockcipher query $Q = (K, X, Y)$, $Y = E_K(X)$, is used in the *top row* if its plaintext, ciphertext, and key are (K_T, X_T, Y_T), otherwise (*i.e.*, if the query is (K_B, X_B, Y_B)) we say that it is used in the *bottom row*.
In order to ensure uniqueness of presentation, the linking function $\mathrm{C}^{LNK} : \{0,1\}^{k+2n} \rightarrow \{0,1\}^{k+n}$ is assumed to discard exactly n bits and not performing any further operation. The output bits are filled up 'in ascending order' of the input bits.

1.4.2. Security Experiments for Double Length Compression Functions

Similar to the security notions for compression functions already given in Section 1.2.1, we now adapt the collision security and preimage security experiments to the case of double length double call blockcipher based compression functions. In the following we assume a double length compression function $H^{\mathrm{DL}} : \{0,1\}^{k-n} \times \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$ as given in the previous Section 1.4.1.

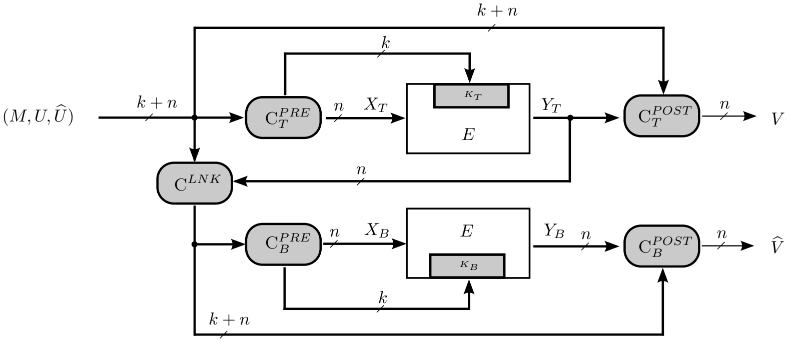


Figure 1.7.: Double Length Double Call Compression Function Framework H^{DL}; $E \in \textsc{Block}(k,n)$; the notch indicates the key input of the cipher

Experiment 1.13 (Collision-Finding $\texttt{Exp-Coll}_{\mathcal{A},H^{\mathrm{DL}}}(2n)$).

1. *An adversary $\mathcal{A}$ is given oracle access to a blockcipher $E \in \textsc{Block}(k,n)$ and returns values $(M, U, \widehat{U}), (M', U', \widehat{U}') \in \{0,1\}^{k-n} \times \{0,1\}^n \times \{0,1\}^n$.*

2. *The output of the experiment is defined to be 1 iff $(M, U, \widehat{U}) \neq (M', U', \widehat{U}')$ and $H^{\mathrm{DL}}(M, U, \widehat{U}) = H^{\mathrm{DL}}(M', U', \widehat{U}')$. In such a case we say that $\mathcal{A}$ has found a collision for H^{DL}.*

The advantage of an adversary $\mathcal{A}$ finding such a collision for H^{DL} is defined as

Definition 1.14. $\mathbf{Adv}^{\mathrm{COLL}}_{H^{\mathrm{DL}}}(\mathcal{A}) = \Pr\left[\texttt{Exp-Coll}_{\mathcal{A},H^{\mathrm{DL}}}(2n) = 1\right]$.

Due to the unlimited computing we give to the adversary, we write

$$\mathbf{Adv}^{\mathrm{COLL}}_{H^{\mathrm{DL}}}(q) := \max_{\mathcal{A}}\{\mathbf{Adv}^{\mathrm{COLL}}_{H^{\mathrm{DL}}}(\mathcal{A})\},$$

where the maximum is taken over all adversaries that ask at most q oracle queries in total.

Experiment 1.15 (Preimage-Finding $\texttt{Exp-Epre}_{\mathcal{A},H^{\mathrm{DL}}}(2n)$).

1. *An adversary $\mathcal{A}$ is given oracle access to a blockcipher $E \in \textsc{Block}(k,n)$. $\mathcal{A}$ selects and announces a value $(V,\widehat{V}) \in \{0,1\}^n \times \{0,1\}^n$ before making any oracle queries. It outputs a value $(M,U,\widehat{U}) \in \{0,1\}^{k-n} \times \{0,1\}^n \times \{0,1\}^n$.*
2. *The output of the experiment is defined to be 1 iff $H^{\mathrm{DL}}(M,U,\widehat{U}) = (V,\widehat{V})$. In such a case we say that $\mathcal{A}$ has found a preimage of H^{DL}.*

We let $\mathbf{Adv}^{\mathrm{EPRE}}_{H^{\mathrm{DL}}}(\mathcal{A})$ be the probability that the experiment $\texttt{Exp-Epre}_{\mathcal{A},H^{\mathrm{DL}}}(2n)$ returns 1. The pre-committed value $(V,\widehat{V})$ is an omitted parameter of $\mathbf{Adv}^{\mathrm{EPRE}}_{H^{\mathrm{DL}}}(\mathcal{A})$. Again, we define

$$\mathbf{Adv}^{\mathrm{EPRE}}_{H^{\mathrm{DL}}}(q) := \max_{\mathcal{A}}\{\mathbf{Adv}^{\mathrm{EPRE}}_{H^{\mathrm{DL}}}(\mathcal{A})\},$$

where the maximum is taken over all adversaries that ask at most q oracle queries in total.

1.4.3. Classification of Double Length Double Call Compression Functions

We now give a classification of DL compression functions that is later used for our analysis. It is a revised and unified variant of the frameworks used in the proofs published in [47, 61].

Generic-DL

The smallest common denominator any double length compression function shares in our analysis is that, given a top-row query $Q_T = (K_T, X_T, Y_T)$, one can compute the input to the bottom-row forward query (K_B, X_B). We do require this as a minimum-standard since otherwise the queries in both rows have some kind of 'unsolicited correlation'. Note that we do not require the converse, *i.e.*, we do *not* require that a bottom-row query uniquely determines the top-row query. We call DL compression functions satisfying this generic requirement Generic-DL. As an example set $K_B = K_T$ and $X_B = X_T \oplus Y_T$. It follows, that for any given top-row query $Q_T = (K_T, X_T, Y_T)$, the bottom-row input to a forward query is uniquely determined. Another example is Mix-Tandem-DM which is discussed in Section 5.3.2.

Serial-DL

Serial-DL is a special case of Generic-DL where one can compute (K_T, Y_T) uniquely given (K_B, X_B). Given $(M, U, \widehat{U})$, the result of the blockcipher query in the top row must be known since the query in the bottom row depends on the ciphertext output of the blockcipher in the top row. We define a helper function $\texttt{SER} : \{0,1\}^{k+n} \rightarrow \{0,1\}^{k+n}$ such that $(K_B, X_B) = \texttt{SER}(K_T, Y_T)$ and $(K_T, Y_T) = \texttt{SER}^{-1}(K_B, X_B)$.

We say that such a compression function is of type Serial-DL if the SL compression function in the 'top row' (C_T^{PRE}, C_T^{POST}) and the SL compression function in the bottom row (C_B^{PRE}, C_B^{POST}) are both of Type-I. A prominent example of a Serial-DL compression function is Tandem-DM [103]. So for a Serial-DL compression function, the following four statements hold:

1. The pre-processing functions C_T^{PRE} and C_B^{PRE} are both bijective with C_T^{-PRE} and C_B^{-PRE} being the inverse functions.

2. For any $(M, U, \widehat{U}) \in \{0,1\}^{k-n} \times \{0,1\}^n \times \{0,1\}^n$, the post-processing functions $\mathrm{C}_T^{POST}(M, U, \widehat{U}, \cdot)$ and $\mathrm{C}_B^{POST}(M, U, \widehat{U}, \cdot)$

are both bijective.

3. For any $A \in \{0,1\}^{k+n}, Y \in \{0,1\}^n$, the modified post-processing functions
$\mathrm{C}_T^{AUX}(A, \cdot, Y)$ and $\mathrm{C}_B^{AUX}(A, \cdot, Y)$ are both bijective.

4. `SER` exists and is bijective.

In order to give concrete security bounds, we have to make sure, that it is not 'too easy' to use one and the same query in the top row and the bottom row simultaneously. This might essentially render the double length compression function into a single length compression function. For an adversary this could imply that if it has found a collision in one row, there could automatically be a collision in the other row (depending on the pre-processing and post-processing functions).

Definition 1.16 (Independence of top row and bottom row). *Let $Q_{fwd} = (K, X, \mathcal{Y})$ be a forward query with $\mathcal{Y} = E_K(X)$. Now let $\zeta_1 \in \mathbb{R}$ be such that*

$$\max_{K,X} \left[\Pr[(K, X) = (\mathrm{C}_B^{PRE} \circ \mathrm{C}^{LNK})(\mathrm{C}_T^{-PRE}(K, X), \mathcal{Y})]\right] \leq \zeta_1.$$

Let $Q_{bwd} = (K, \mathcal{X}, Y)$ be a backward query with $\mathcal{X} = E_K^{-1}(Y)$. Let $\zeta_2 \in \mathbb{R}$ be such that

$$\max_{K,Y} \left[\Pr[(K, \mathcal{X}) = (\mathrm{C}_B^{PRE} \circ \mathrm{C}^{LNK})(\mathrm{C}_T^{-PRE}(K, \mathcal{X}), Y)]\right] \leq \zeta_2.$$

The independence *ζ of the top row and bottom row of a double-length compression function is now defined as $\zeta := \max(\zeta_1, \zeta_2)$.*

Cyclic-DL

This is also a special case of the Generic-DL construction where one can compute (K_T, X_T) uniquely given (K_B, X_B) and vice versa. Let $\sigma : \{0,1\}^{k+n} \rightarrow \{0,1\}^{k+n}$ be such that $(K_B, X_B) = \sigma(K_T, X_T)$. Clearly, σ is a bijection.

Definition 1.17. *Let $d > 0$ be an integer and* ID *be the identity mapping on $\{0,1\}^{k+n}$. The permutation σ^d is defined as $\sigma^d := \sigma \circ \sigma^{d-1}$ and $\sigma^0 :=$* ID.

(i) Fix some element $s \in \{0,1\}^{k+n}$. The order of s is defined to be $|s| = \min_{t>1}(\sigma^t(s) = s)$.

(ii) If there is an integer z such that $\forall\ s \in \{0,1\}^{k+n} : |s| = z$, we say that the order of the mapping σ, denoted by $|\sigma|$, is equal to z. If there is no such z, then $|\sigma| := 0$.

Now assume that $|\sigma| \geq 2$. We say that in such a case a compression function is of type Cyclic-DL, if the SL compression function in the top row (C_T^{PRE}, C_T^{POST}) and the SL compression function in the bottom row (C_B^{PRE}, C_B^{POST}) are both of Type-I. Examples of Cyclic-DL compression functions are Abreast-DM ($|\sigma| = 6$) and Hirose-DM ($|\sigma| = 2$). For a Cyclic-DL compression function, the following four hold:

1. The pre-processing functions C_T^{PRE} and C_B^{PRE} are both bijective with C_T^{-PRE} and C_B^{-PRE} being the inverse functions.

2. For any $(M, U, \widehat{U}) \in \{0,1\}^{k-n} \times \{0,1\}^n \times \{0,1\}^n$, the post-processing functions $\mathrm{C}_T^{POST}(M, U, \widehat{U}, \cdot)$ and $\mathrm{C}_B^{POST}(M, U, \widehat{U}, \cdot)$ are both bijective.

3. For any $A \in \{0,1\}^{k+n}, Y \in \{0,1\}^n$, the modified post-processing functions
$\mathrm{C}_T^{AUX}(A, \cdot, Y)$ and $\mathrm{C}_B^{AUX}(A, \cdot, Y)$ are both bijective.

4. $|\sigma| \geq 2$.

Part I.

Collision Security of Double Length Hash Functions

2 Results Summary

In the first part of this thesis we analyze the collision security of several blockcipher based double length compression functions and hash functions. An overview of our results presented in the following Chapters 3 - 6 is given in Table 2.1. The 'published' column lists related publications of the author. The numerical security bounds given in this table are evaluated for a blockcipher from $\text{Block}(256, 128)$ or $\text{Block}(128, 128)$ (the latter only in the case of MDC-4). The collision security bounds state that no adversary, asking less than the given amount of queries to the blockcipher, can find a collision with probability greater than 1/2.

Chapter 3 - Weimar-DM

This chapter starts with a simple motivational 'example analysis' of a double length compression function with rate 1/2, Weimar-DM, demonstrating the basic procedures. But this surprisingly simple proof should not hide the fact, that Weimar-DM is currently *the* double length compression function with the tightest collision security bound known: we show that no adversary asking less than $2^{126.73}$

Comp. Funct.	**Coll. sec. bound**	**Chap.**	**Published**
Weimar-DM	$2^{126.73}$ / $2^{n-1.27}$	3	[54]
Abreast-DM	$2^{124.42}$ / $2^{n-3.58}$	4	[61]
Cyclic-DL, Add/k-DM	$2^{127-\log_2 c}$ cycle length $c = 2^k$	4	[61]
Cube-DM	$2^{125.41}$	4	[61]
Serial-DL, Tandem-DM	$2^{119.6}$	5	[47, 60]
Generic-DL, Mix-Tandem-DM	$2^{83.2}$	5	[47]
MDC-4 (hash function)	$2^{74.76}$	6	[52]

Table 2.1.: Overview of our collision security results for double length compression and hash functions. Our results are in the ideal cipher model. The values in the 'Coll. sec. bound' column are evaluated for $n = 128$ and a success probability of 1/2. If available, an asymptotic bound for $n \to \infty$ is given.

queries can find a collision with probability greater than 1/2.

Chapter 4 - Abreast-DM, Cyclic-DL and Applications

We provide the collision security proof for Abreast-DM. In particular, we show that for Abreast-DM, when instantiated with a blockcipher from Block$(256, 128)$, an adversary that asks less than $2^{124.42}$ queries cannot find a collision with success probability greater than 1/2. We generalize our techniques used in the proof of Abreast-DM to Cyclic-DL (cf. Section 1.4.3). It is easy to see that Cyclic-DL covers any DL construction that enables one to compute both blockcipher calls *in parallel* as, *e.g.*, for Hirose-DM or for the modified Abreast-DM scheme given by Lee and Kwon [104]. Using our security results from Cyclic-DL, we are able to derive several DL constructions that lead to compression functions that have higher collision security guarantees and are more efficient than Abreast-DM: Add/k-DM and Cube-DM. At the time of publication [61], the newly presented Cube-DM had the highest collision security 'guarantee' of any known DL compression function. This crown has now been passed to Weimar-DM.

Chapter 5 - Serial-DL and Applications

In this chapter, we give a generic collision security bound for Serial-DL compression functions (cf. Section 1.4.3). From this, we can easily derive such a bound for Tandem-DM: Assuming a blockcipher from Block$(256, 128)$, we show that no adversary asking less than $2^{119.6}$ queries to the blockcipher can find a collision with probability greater than 1/2. We also give a collision security analysis for Generic-DL (cf. Section 1.4.3) and state, as an application, a collision security bound of $2^{83.2}$ for Mix-Tandem-DM, which is a simple example of a DL compression function that is neither in Serial-DL nor in Cyclic-DL, but in Generic-DL.

Chapter 6 - MDC-4

We show for the hash function MDC-4, assuming a blockcipher from Block$(128, 128)$, that no adversary asking less than $2^{74.76}$ queries can find a collision with probability greater than 1/2. This is the first result on the collision security of MDC-4. Although being somewhat similar to the MDC-2 collision security analysis of Steinberger [83, 162], his result could not be directly applied to MDC-4 due to the structural differences. Also, our proof for MDC-4 is much shorter than Steinberger's MDC-2 analysis and we claim that our presentation is also easier to grasp.

Revisions since Initial Publication

The results of [52] and Generic-DL [47] have found their way without major modifications into this thesis. This is also true for the results regarding Abreast-DM and Add/k-DM [61]. The analysis of Cyclic-DM [61] has been *further* generalized to Cyclic-DL– thus covering not only Davies-Meyer based double length compression functions but any reasonable combination of 'secure' Type-1 SL compression functions. This also fits nicely with our definition of double length compression functions given in Section 1.4. The numerical results have not been changed by this revision. The collision security results of Weimar-DM [54] has been further tightened (from $2^{n-1.77}$ to $2^{n-1.27}$).

Part of the results published in [60] and [47] (Serial-DL and Tandem-DM) unfortunately have been erroneous and are corrected using a technique given by Lee *et al.* [107]. We explicitly point out the modification we made in our proofs.

3 Weimar-DM

We start the discussion of collision security for double length compression functions with a simple example, Weimar-DM, and its collision security analysis.

Definition 3.1. *Let* $E \in \textsc{Block}(2n, n)$. *The compression function* $H^{\mathrm{WDM}} : \{0,1\}^n \times \{0,1\}^{2n} \to \{0,1\}^{2n}$ *is defined as (cf. Figure 3.1)*

$$H^{\mathrm{WDM}}(M, U, \widehat{U}) = \left(E_{M\|U}(\widehat{U}) \oplus \widehat{U}, E_{\overline{M\|U}}(\widehat{U}) \oplus \widehat{U}\right).$$

It is easy to see that H^{WDM} is of type Cyclic-DL with $|\sigma| = 2$. Since we are going to provide a proof for Cyclic-DL in Section 4.2, we could skip this proof here, but we do not: First, it is a good educational example on how such proofs work. Second, the proof is much simpler than the generic analysis for Cyclic-DL. And, third, the bound derived here is somewhat better than the bound we get by

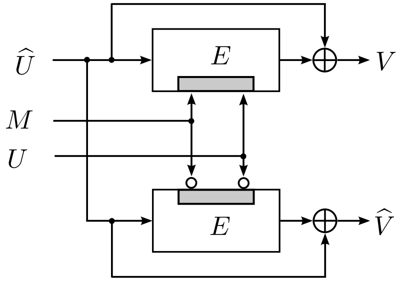


Figure 3.1.: Weimar-DM compression function H^{WDM}; the small circle '$\circ$' denotes a bit-by-bit complement

applying the Cyclic-DL result. We now proceed to show that the compression function H^{WDM} is indeed collision resistant.

Theorem 3.2. *Let $N = 2^n$. Then,* $\mathbf{Adv}^{\mathrm{Coll}}_{H^{\mathrm{WDM}}}(q) \leq \frac{q(q+1)}{(N-q)^2}$.

In numerical terms, *e.g.*, for $n = 128$ and $\mathbf{Adv}^{\mathrm{COLL}}_{H^{\mathrm{WDM}}}(q) = 1/2$, we have $q = 2^{126.73}$. Using simple calculus, it is easy to see that for $\alpha = N(\sqrt{2} - 1)= 2^{n-1.27}$ we have

$$\mathbf{Adv}^{\mathrm{COLL}}_{H^{\mathrm{WDM}}}(\alpha) = \frac{1}{2} + o(1),$$

where the term $o(1) \to 0$ for $n \to \infty$. Neglecting constant factors, our security bound reads as an asymptotically optimal bound of $O(2^n)$ for a compression function with $2n$-bit output.

Proof. We assume that the adversary has made any relevant query to E to come up with a collision – which is reasonable in the ideal cipher model. We start by considering an arbitrary q-query collision finding adversary $\mathcal{A}$. We then construct an adversary $\mathcal{A}'$ which simulates $\mathcal{A}$

but does sometimes ask an additional query to the E oracle under certain circumstances.

Since $\mathcal{A}'$ is more powerful than $\mathcal{A}$, it suffices to upper bound the success probability of A'. We now give a detailed description of $\mathcal{A}'$ by simultaneously upper bounding its chances of success.

Description of $\mathcal{A}'$. The adversary $\mathcal{A}'$ maintains an initially empty list $\mathcal{L}$ representing any possible input/output of the compression function H^{WDM} that can be computed by the adversary $\mathcal{A}$. An entry $L \in \mathcal{L}$ is a 4-tuple $(K, X, Y, Y') \in \{0,1\}^{5n}$ where $K \in \{0,1\}^{2n}$, $X \in \{0,1\}^n$ is the $3n$-bit input to the compression function such that $(M, U) = K$ and $\widehat{U} = X$. The n-bit values Y and Y' are given by $Y = E_K(X)$, $Y' = E_{\overline{K}}(X)$.

The list is now built as follows. Say that the adversary $\mathcal{A}$ mounts its i-th query to E or E^{-1}, $1 \leq i \leq q$. In the case of a forward query, the adversary gets hold of the tuple (K, X, Y) where $Y = E_K(X)$. In the case of a backward query, the adversary gets also hold of the tuple (K, X, Y), but in this case $X = E_K^{-1}(Y)$. In either case, the value $X \oplus Y$ is randomly determined by the output of the query.

Now, $\mathcal{A}'$ checks if an entry $L = (K, X, *, *)$ or $L' = (\overline{K}, X, *, *)$ is contained in $\mathcal{L}$ where $'*'$ denotes an arbitrary value. We now analyze the two possible cases $\mathcal{A}'$ might be confronted with and upper bound their success probability separately.

Case 1 Neither L nor L' are in $\mathcal{L}$. Then $\mathcal{A}'$ mounts an additional forward query $Y' = E_{\overline{K}}(X)$. Note that $Y' \oplus X$, the result of the 'bottom row' of the compression function, is always uniformly distributed since $K \neq \overline{K}$. Hence, the results of the first query asked by the adversary $\mathcal{A}$ and the second query asked additionally by the adversary $\mathcal{A}'$ are always independently distributed. Set $L_i := (K, X, Y, Y')$. We append L_i to the list $\mathcal{L}$.

We now define what we mean by a collision in the list. Fix two integers r and s such that $L_r = (K_r, X_r, Y_r, Y_r')$ represents the r-th entry in $\mathcal{L}$ and $L_s = (K_s, X_s, Y_s, Y_s')$ the s-th entry in $\mathcal{L}$ and both entries exist. We say that L_s and L_r collide if a collision of the compression functions occurs that can be computed using the query results given in L_r and L_s. This is the case if at least one of the following two conditions is met:

1. $Y_r \oplus X_r = Y_s \oplus X_s$ and $Y'_r \oplus X_r = Y'_s \oplus X_s$ or
2. $Y_r \oplus X_r = Y'_s \oplus X_s$ and $Y'_r \oplus X_r = Y_s \oplus X_s$.

So for the i-th query, there are at most $i-1$ entries in the list $\mathcal{L}$ that might collide with L_i. We can upper bound the probability of success of the i-th query by

$$\sum_{j=1}^{i-1} \frac{2}{(N-q)(N-q)} \leq \frac{2i}{(N-q)(N-q)}.$$

As the adversary can ask at most q queries, the list $\mathcal{L}$ cannot contain more than q entries since for any adversary query at most one additional entry is added to the list $\mathcal{L}$ of $\mathcal{A}'$. So the total chance of success for q queries is

$$\leq \sum_{i=1}^{q} \frac{2i}{(N-q)(N-q)} = \frac{q(q+1)}{(N-q)^2}.$$

In case of a collision in $\mathcal{L}$ we give the attack to the adversary.

Case 2 Now assume that at least one of the values L or L' is already in $\mathcal{L}$. Then $\mathcal{A}'$ ignores this query, since we already know that $\mathcal{A}$ has zero chance of winning since otherwise we would have given the attack to the adversary before. □

4 Abreast-DM, Cyclic-DL and Applications

4.1. Abreast-DM

4.1.1. Compression Function

We recall the already introduced Abreast-DM DL compression function. For better reference, we illustrate it again in Figure 4.1. Abreast-DM was formally defined in Section 1.4 on page 29.

4.1.2. Security Results

Our discussion results in a proof of the bound given in Theorem 4.1.

Theorem 4.1. *(Collision Resistance of* Abreast-DM*) Let n, q be natural numbers with $q < 2^{n-3.58}$ and $N = 2^n$. Then,*

$$\mathbf{Adv}_{H^{\mathrm{ADM}}}^{\mathrm{COLL}}(q) \leq 18 \left(\frac{q}{N/2}\right)^2 .$$

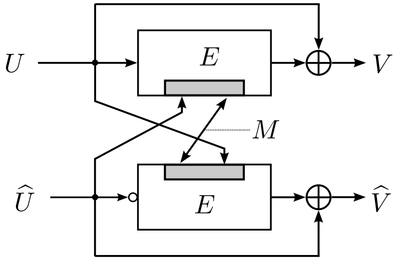


Figure 4.1.: Abreast-DM H^{ADM}; the small circle '$\circ$' denotes a bit-by-bit complement

For $n = 128$ we explicitly state, what this Theorem means numerically.

Corollary 4.2. *For the compression function* Abreast-DM*, 'instantiated' with an (ideal) blockcipher from* Block$(256, 128)$*, any adversary asking less than $q = 2^{124.42}$ (backward or forward) oracle queries cannot find a collision with probability greater than* $1/2$*.*

The proof of Theorem 4.1 is given in Section 4.1.3.

4.1.3. Collision Security of Abreast-DM

In this Section, we provide a proof of Theorem 4.1.

Analysis Overview. We analyze if the queries made by the adversary contain the means for constructing a collision of the compression function H^{ADM}. Two queries to the oracles E, E^{-1} in total are required to compute the output $(V, \widehat{V})$ of H^{ADM} for any given

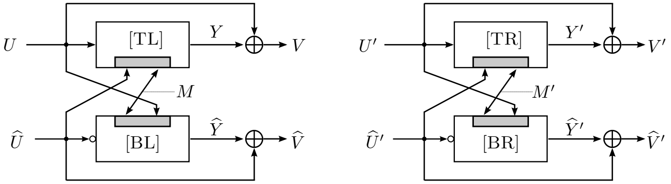


Figure 4.2.: Notations used for a collision of Abreast-DM; If $\text{Coll}^{ADM}(\mathcal{Q})$, then $V = V'$ and $\widehat{V} = \widehat{V}'$ hold.

input $(M, U, \widehat{U})$. It is easy to see that one oracle query and its result uniquely determines the other query. Effectively, we look to see whether there exist four (not necessarily distinct) queries that form a collision (cf. Figure 4.2).

To upper bound the probability of a fixed q-query asking adversary obtaining queries than can be used to construct a collision, we upper bound the probability of the adversary making one query that can be used as the final query to complete such a collision. Namely, for each i, $1 \leq i \leq q$, we upper bound the probability that the answer to the adversary's i-th forward query $(K_i, X_i, *)_{fwd}$ or backward query $(K_i, *, Y_i)_{bwd}$ allows it to use the i-th query to complete the collision. In the latter case, we say that the i-th query is *successful* and we give the attack to the adversary.

Naturally, the computation of any single compression function depends on *two* blockcipher calls – assuming that the construction does not allow the use of one and the same query in the top row and the bottom row of the compression function. This is true for Abreast-DM. In order to upper bound the success probability of any single query mounted by the adversary, we have to upper bound the maximal number of compression functions the adversary can complete with this single query result. At a first glance, any single query can be used in the *top row* or in the *bottom row* of a compression function. Say, *e.g.*, the query is $(\widehat{U} \| M, U, Y)$ and the adversary intends to use it in the top row. As it is practically impossible to track whether the adversary has mounted the corresponding bottom-row query $(M\|U, \overline{\widehat{U}}, \widehat{Y})$ in the past, we have to assume that the adversary has access to this query. Formally, we give the adversary this query *for free*. These can be modeled as queries which the adversary is *forced* to query (under certain conditions), but for which the adversary is not charged, *i.e.*, they do not count towards the maximum of q queries which the adversary is allowed to ask. However, these queries become part of the adversary's query history, just like other queries. In particular, the adversary is not allowed, later, to remake these queries *on its own* (due to the previously discussed assumption that the adversary never makes a query which it already owns).

In general, just this *free* query that is intended for use in the bottom row, can be reused also in the top row by the adversary in order to start the computation of a new compression function. And, again, as we cannot say whether the adversary has access to the corresponding bottom row query, we have to assume that the adversary has access to it (*i.e.*, we give the query for free to the adversary). Pursuing this process seems to result in a virtually infinite action. But it does not for Abreast-DM.

On Abreast-DM's Cycle. Assume that the adversary mounts his i-th query denoted by $Q_{6i} = (\widehat{U}\|M, U, Y_1)$. For ease of presentation, we give the adversary's 'first' query the index zero and therefore $i \in \{0, 1, \ldots, q-1\}$ assuming that the adversary mounts q queries in total. Also we give the adversary's i-th query index $6i$ for reasons that will become clear later – in short, this is due to the 5 'free' queries the adversary is given for any single mounted query. First, assume that the query Q_{6i} is used in the top row. The adversary is given for free the corresponding query of the bottom row $Q_{6i+1} := (M\|U, \overline{\widehat{U}}, Y_2)$. Using Q_{6i} and Q_{6i+1}, the adversary is able to compute one result of a compression function

$$(V_1, \widehat{V}_1) := H^{\mathrm{ADM}}(M, U, \widehat{U}) = (E_{\widehat{U}\|M}(U) \oplus U, E_{M\|U}(\overline{\widehat{U}}) \oplus \widehat{U}).$$

As the adversary can reuse the free query Q_{6i+1} in the top row, we give the adversary for free the matching bottom-row query $Q_{6i+2} = (\overline{M}, U\|\overline{\widehat{U}}, Y_3)$. After this, the adversary can additionally compute

$$(V_2, \widehat{V}_2) := H^{\mathrm{ADM}}(\overline{\widehat{U}}, M, U) = (E_{M\|U}(\overline{\widehat{U}}) \oplus \overline{\widehat{U}}, E_{U\|\overline{\widehat{U}}}(\overline{M}) \oplus M),$$

using the queries Q_{6i+1} and Q_{6i+2}. The continuation of this process is summarized in Table 4.1.3. Likewise, the adversary is given for free the (forward) queries $Q_{6i+3}, Q_{6i+4}, Q_{6i+5}$. The query Q_{6i+6} is equal to the initial query of the adversary Q_{6i} and this process comes to an end.

Note that the query numbers in parentheses in Table 4.1.3 denote a reuse of a previous query, *e.g.*, $(6i+1)$ denotes the reuse of query Q_{6i+1} in another position (either top or bottom).
As indicated by Table 4.1.3, *any* single query is used in the bottom row as well as in the top row. Since any single query (used in top row or bottom row) uniquely determines the corresponding query in the bottom row or top row, it follows that all queries can only be used for the compression functions mentioned in Table 4.1.3. It is not possible to use them again for computing the output result of any other compression function for the adversary.

Analysis Details. Fix numbers n, q, and an adversary $\mathcal{A}$ asking in total q forward or backward queries to its oracle E. Let $\textsc{Coll}^{\textsc{adm}}(\mathcal{Q})$ be the event that the adversary can construct a collision of H^{ADM} using the queries in the query history $\mathcal{Q}$. The term 'last query' means the latest query made by the adversary. It is always the i-th query of the adversary and it is always denoted as Q_{6i}. As mentioned, this is due to the fact that we give the adversary, for any single query it asks, 5 additional free queries. We examine the adversary's mounted queries one at a time as they come in. Similarly, the free queries are also examined one at a time as they are given to the adversary.

We say a query $Q_m = (K_m^1\|K_m^2, X_m, Y_m)$ is successful if the output, Y_m for a forward query or X_m for a backward query, is such

$H^{\mathrm{UDM}}(\cdot)$	Query #	Plaintext	Key	Ciphertext	Chaining Value
$(M, U, \widehat{U})$	$6i$ (*)	U	$\widehat{U} \| M$	Y_1	$V_1 = Y_1 \oplus U$
	$6i+1$	$\overline{\widehat{U}}$	$M \| U$	Y_2	$\widehat{V}_1 = Y_2 \oplus \widehat{U}$
$(U, \overline{\widehat{U}}, M)$	$(6i+1)$	$\overline{\widehat{U}}$	$M \| U$	Y_2	$V_2 = Y_2 \oplus \overline{\widehat{U}}$
	$6i+2$	$\overline{M}$	$U \| \overline{\widehat{U}}$	Y_3	$\widehat{V}_2 = Y_3 \oplus M$
$(\overline{\widehat{U}}, \overline{M}, U)$	$(6i+2)$	$\overline{M}$	$U \| \overline{\widehat{U}}$	Y_3	$V_3 = Y_3 \oplus \overline{M}$
	$6i+3$	$\overline{U}$	$\overline{\widehat{U}} \| \overline{M}$	Y_4	$\widehat{V}_4 = Y_4 \oplus U$
$(\overline{M}, \overline{U}, \overline{\widehat{U}})$	$(6i+3)$	$\overline{U}$	$\overline{\widehat{U}} \| \overline{M}$	Y_4	$V_4 = Y_4 \oplus \overline{U}$
	$6i+4$	$\widehat{U}$	$\overline{M} \| \overline{U}$	Y_5	$\widehat{V}_4 = Y_5 \oplus \overline{\widehat{U}}$
$(\overline{U}, \widehat{U}, \overline{M})$	$(6i+4)$	$\widehat{U}$	$\overline{M} \| \overline{U}$	Y_5	$V_5 = Y_5 \oplus \widehat{U}$
	$6i+5$	M	$\overline{U} \| \widehat{U}$	Y_6	$\widehat{V}_5 = Y_6 \oplus \overline{M}$
$(\widehat{U}, M, \overline{U})$	$(6i+5)$	M	$\overline{U} \| \widehat{U}$	Y_6	$V_6 = Y_6 \oplus M$
	$(6i)$	U	$\widehat{U} \| M$	Y_1	$V_6 = Y_1 \oplus \overline{U}$

Table 4.1.: Starting with the i-th query of the adversary, Q_{6i}, either $(\widehat{U} \| M, U, *)_{fwd}$ or $(\widehat{U} \| M, *, Y)_{bwd}$, the adversary is given 5 additional forward queries, query #'s $6i+1, 6i+2, 6i+3, 6i+4, 6i+5$, for free. In total, it is able to compute six complete compression functions H^{ADM} by using these six queries. Notation: (*) is the only query the adversary has mounted.

that the adversary can use this very query Q_m to form a collision. More precise, there are three queries Q_j, Q_k, Q_l in $\mathcal{Q}$ such that the four (not necessarily pairwise different) queries Q_m, Q_j, Q_k, Q_l can be used for a collision (cf. Figure 4.2). The goal is thus to upper bound the adversary's chance of ever making a successful last query.

We now upper bound $\Pr[\textsc{Coll}^{\text{ADM}}(\mathcal{Q})]$ by exhibiting the predicates $\textsc{Win}_0(\mathcal{Q})$, …, $\textsc{Win}_{q-1}(\mathcal{Q})$ such that $\textsc{Coll}^{\text{ADM}} \implies \textsc{Win}_0(\mathcal{Q}) \vee \ldots \vee \textsc{Win}_{q-1}(\mathcal{Q})$. It follows that $\Pr[\textsc{Coll}^{\text{ADM}}(\mathcal{Q})] \leq \Pr[\textsc{Win}_0(\mathcal{Q})] + \ldots + \Pr[\textsc{Win}_{q-1}(\mathcal{Q})]$.

Since the adversary mounts q queries in total, we informally say that $\textsc{Win}_i(\mathcal{Q})$, $0 \leq i \leq q-1$, holds, if the adversary finds a collision after mounting its i-th query, *i.e.*, using at least one of the queries $Q_{6i}, \ldots, Q_{6i+5}$ conditioned on the fact that the adversary has not been successful before.

Notation: Let $\mathcal{Q}_k$ denote the first k queries made by the adversary or the adversary had been given for free: $\mathcal{Q}_k = \cup_{0 \leq j \leq k} \; Q_j$ and $|\mathcal{Q}_k| = k+1$.

To formally define the predicates $\textsc{Win}_i(\mathcal{Q})$ the following Definitions 4.3 and 4.4 are convenient.

Definition 4.3. *We say that a pair of queries (Q_a, Q_b) is successful in the query set $\mathcal{Q}$, if – for computation of the compression function output of H^{ADM} – the query Q_a is used in the top row, the query Q_b is used in the bottom row, and there exists a pair of queries $Q_j, Q_k \in \mathcal{Q}$ such that a collision for H^{ADM} can be computed:*

$$X_a \oplus Y_a = X_j \oplus Y_j \qquad \text{and} \qquad \overline{X_b} \oplus Y_b = \overline{X_k} \oplus Y_k.$$

Definition 4.4. *Let $d = 0, \ldots, 5$, $d' = d+1 \bmod 6$, $\widetilde{d} = \max(d, d')$. We say $\textsc{CollFit}_i^d(\mathcal{Q})$ if (i) the pair of queries $(6i+d, 6i+d')$ is successful in $\mathcal{Q}_{6i+\widetilde{d}}$ and (ii) the adversary had not been successful* *for* $0 \leq t \leq d-1$*:* $\neg\textsc{CollFit}_i^t(\mathcal{Q})$*. The probability is measured over the post-outputs of the newly received queries* $(6i+d)$ *and* $6i+d'$*.*

The predicates $\textsc{Win}_i(\mathcal{Q})$ are defined as follows:

Definition 4.5. *For* $0 \leq i \leq q-1$*,*

$$\begin{aligned}\textsc{Win}_i(\mathcal{Q}) =& \neg\left(\bigvee_{0 \leq j \leq i-1} \textsc{Win}_j(\mathcal{Q})\right)\\ &\wedge \left(\textsc{CollFit}_i^0(\mathcal{Q}) \vee \ldots \vee \textsc{CollFit}_i^5(\mathcal{Q})\right).\end{aligned}$$

We now show that our case analysis is complete.

Lemma 4.6. $\textsc{Coll}^{\text{ADM}}(\mathcal{Q}) \Longrightarrow \textsc{Win}_0(\mathcal{Q}) \vee \ldots \vee \textsc{Win}_{q-1}(\mathcal{Q})$*.*

Proof. Say $\textsc{Coll}^{\text{ADM}}(\mathcal{Q})$. Then a collision can be constructed from the queries in $\mathcal{Q}$. That is, our query history $\mathcal{Q}$ contains queries Q_i, Q_j, Q_k, Q_l (see Figure 4.2) that can be used in positions TL, TR, BL and BR, TL $\neq$ TR, such that $V = V'$ and $\widehat{V} = \widehat{V}'$. Note that the condition TL $\neq$ TR suffices to ensure that a collision from two different inputs has occurred. It is easy to see that no query mounted directly by the adversary can be successful since any such query can only serve for either a top- or bottom row position in the compression function H^{ADM}. Also, the corresponding query, which is necessary to compute the complete compression function, is given to the adversary for free *after* it has mounted a query. So the adversary can only be successful in the phase where it is given the free queries one after another. Say the adversary is successful during the phase where it gets the free queries following his i-th query. We can safely assume that this is the first time the adversary has found such a collision and therefore i is minimal. Then $\neg\textsc{Win}_j(\mathcal{Q})$, $0 \le j < i$, $\textsc{CollFit}_i^d(\mathcal{Q})$ such that $d \in \{0, 1, \ldots, 5\}$ is minimal and therefore $\textsc{Win}_i(\mathcal{Q})$. This proves our claim. □

Since $\Pr[\textsc{Coll}^{\text{ADM}}(\mathcal{Q})] \le \sum_{j=0}^{q-1} \textsc{Win}_j(\mathcal{Q})$ it follows that

$$\Pr[\textsc{Coll}^{\text{ADM}}(\mathcal{Q})] \le \sum_{i=0}^{q-1} \sum_{d=0}^{5} \textsc{CollFit}_i^d(\mathcal{Q}). \tag{4.1}$$

We now proceed to upper bound the probability of $\textsc{CollFit}_i^d(\mathcal{Q})$.

Lemma 4.7. *Let* $1 \le i \le q$ *and* $0 \le d \le 5$. *Then*

$$\Pr[\textsc{CollFit}_i^d(\mathcal{Q})] \le \frac{6i}{(N-6i)^2}.$$

Proof. Let $d' = d + 1 \ mod \ 6$. The $2n$-bit output of the compression function H^{ADM}, $(V, \widehat{V})$, is uniquely determined by the queries $Q_{6i+d} = (K_{6i+d}, X_{6i+d}, Y_{6i+d})$ and $Q_{6i+d'} = (K_{6i+d'}, X_{6i+d'}, Y_{6i+d'})$ by

$$V = Y_{6i+d} \oplus X_{6i+d} \qquad \text{and} \qquad \widehat{V} = Y_{6i+d'} \oplus \overline{X_{6i+d'}}.$$

Both, V and $\widehat{V}$, depend on the plaintext and the ciphertext of E. If Q_{6i+d} was received by a forward query, the key and the plaintext are fixed. The result of this query, *i.e.*, the ciphertext, is chosen uniformly at random from a set of at least $N - (6i + d)$ values. In the case of a backward query, the key and the ciphertext are fixed. The result of this query, the plaintext, is chosen uniformly at random from a set of at least $N - (6i + d)$ values. It follows that V is in either case randomly determined by the answer of the E oracle. Using the same arguments, it follows that $\widehat{V}$ is also randomly determined by the answer of the oracle. Note that the bit-by-bit inversion in the bottom row of $X_{6i+d'}$ does *not* change any of our arguments.

To form a collision, two queries Q_j, Q_k are needed that can be chosen from at most $6(i+1)$ queries in $\mathcal{Q}_{6(i+1)-1}$. The adversary can use them to compute the output of $< 6(i+1)$ compression functions H^{ADM}. Therefore,

$$\Pr[\textsc{CollFit}_i^d(\mathcal{Q})] \leq \frac{6(i+1)}{(N-6(i+1))^2}.$$

□

Using equation (4.1) we get the following upper bound for any $q < 2^{n-\log_2 6} = 2^{n-2.58}$

$$\begin{aligned}\Pr[\textsc{Coll}^{\text{ADM}}(\mathcal{Q})] &\leq \sum_{i=0}^{q-1}\sum_{d=0}^{5}\frac{6(i+1)}{(N-6(i+1))^2} \leq \sum_{i=1}^{q}\sum_{d=0}^{5}\frac{6i}{(N-6i)^2}\\ &\leq \sum_{i=1}^{q}\frac{36i}{(N-6i)^2} \leq \frac{36\cdot q^2\cdot\frac{1}{2}}{(N-6q)^2} \leq 18\left(\frac{q}{2^{n-1}}\right)^2.\end{aligned}$$

This completes our proof of Theorem 4.1. □

4.2. Cyclic-DL

We now generalize the techniques used in the last section for analyzing Abreast-DM and apply them to Cyclic-DL (cf. Section 1.4.3). Our discussion results in two security bounds for Cyclic-DL, one for the case $|\sigma| = 2$ and a proof for the case $|\sigma| > 2$.

Theorem 4.8. *(Collision Resistance for $c = 2$) Let $H^{2\text{CYC}}$ be a* Cyclic-DL *compression function with cycle length $c = 2$ and let $N = 2^n$. If $\mathrm{C}_T^{PRE} = \mathrm{C}_B^{PRE}$ and $\mathrm{C}_T^{POST} = \mathrm{C}_B^{POST}$, then $a = 1$, else $a = 2$. Then, for any $q > 1$ and $2q < N$,*

$$\mathbf{Adv}_{H^{2\text{CYC}}}^{\text{COLL}}(q) \leq \frac{2aq^2}{(N-2q)^2} + \frac{2q}{N-2q}.$$

Theorem 4.9. *(Collision Resistance for $c > 2$) Let H^{CYC} be a* Cyclic-DL *compression function with cycle length $c > 2$ and let $N = 2^n$. Then, for any $q > 1$ and $cq < N$,*

$$\mathbf{Adv}_{H^{\text{CYC}}}^{\text{COLL}}(q) \leq \frac{c^2}{2}\left(\frac{q}{N-cq}\right)^2.$$

Remarks. Naturally, the question arises what to conclude if only some form of 'generalization' of Definition 1.17 is applicable, *i.e.*, if not all elements s share the same order. In theory, one can imagine a case where some elements have an order of 2, some of 3 and some have an order of 4. Since for any $|\sigma| > 2$, Theorem 4.9 states a monotonically decreasing upper bound. We can easily assume the 'worst case' bound and apply this theorem for the maximum cycle length *if* all elements have a minimum order of 3. Should there be elements s that have an order of 2, one has to take into account the result given in Theorem 4.8. The final bound for the advantage can be obtained by taking the maximum of all bounds that result from the order of the elements. Simply, this is due to the fact that a query resulting in a 'query cycle' of length $\widetilde{c}$ cannot be used in any other query cycle since any query cycle is a closed set.

4.2.1. Proof of Theorem 4.8 – Collision Resistance of Cyclic-DL $(c = 2)$

Due to the special structure of the compression function in the case of $c = |\sigma| = 2$, the following definition, inspired by [74], is convenient for the proof.

Definition 4.10. *A pair of distinct inputs,* $(M, U, \widehat{U})$ *and* $(M', U', \widehat{U}')$*, to a compression function* $H^{2\mathrm{CYC}}$ *is called a* matching *pair if*

$$\sigma(\mathrm{C}_T^{PRE}(M, U, \widehat{U})) = \mathrm{C}_B^{PRE}(\mathrm{C}^{LNK}(M', U', \widehat{U}')).$$

Otherwise they are called a non-matching *pair.*

Fix numbers n, q, and an adversary $\mathcal{A}$ asking q backward and forward queries to its oracle E in total. All queries to the oracle as well as any queries received 'for free' are saved in a query history $\mathcal{Q}$. Let $\textsc{Coll}^{2\mathrm{CYC}}$ be the event that the adversary is able to construct a collision of $H^{2\mathrm{CYC}}$ from the queries in $\mathcal{Q}$. For this we examine the queries, one at a time, as they come in; the latest query made by the adversary, his i-th query, is always be given index $2i$, and is denoted by $Q_{2i} = (K_{2i}, X_{2i}, Y_{2i})$. Assume that the query Q_{2i} is a forward or backward query asked by the adversary and assume that Q_{2i} is used in the top row. Since two *different* queries are required for the computation of $H^{2\mathrm{CYC}}$ we give the adversary the bottom-row query for free. This query is uniquely determined by its plaintext X_{2i+1} and key K_{2i+1} component as follows:

$$(K_{2i+1}, X_{2i+1}) = \sigma(K_{2i}, X_{2i})$$

and the adversary is given the ciphertext $Y_{2i+1} = E_{K_{2i+1}}(X_{2i+1})$. If the adversary uses the query Q_{2i} in the bottom row, we give it the top-row query for free:

$$(K_{2i+1}, X_{2i+1}) = \sigma^{-1}(K_{2i}, X_{2i})$$

and the adversary is given the ciphertext $Y_{2i+1} = E_{K_{2i+1}}(X_{2i+1})$ in this case. Since $\sigma^2 = \mathrm{ID}$ it follows that $\sigma = \sigma^{-1}$. So in either case, the adversary is given the *same* free query, *i.e.*, the input to the other query is always uniquely determined using one and the same computation.

Now assume for simplicity of the following argument that the query Q_{2i} is used in the top row and Q_{2i+1} in the bottom row. In the case of a forward query Q_{2i} the output of the compression function in the top row is randomly determined via C_T^{POST}. Similarly, in the case of a backward query Q_{2i}, the output of the compression function in the top row is randomly determined via C_T^{AUX}. After getting the result of its query to Q_{2i}, the adversary gets the corresponding bottom-row forward query Q_{2i+1} for free. So the bottom-row output of the compression function is randomly determined via C_B^{POST}.

For any $2 \leq i \leq q$ let C_i be the event that a colliding pair of non-matching inputs is found for $H^{2\mathrm{CYC}}$ with the i-th pair of queries. Namely, it is the event that for some $i' < i$

$$\begin{aligned} H^{2\mathrm{CYC}}(\mathrm{C}_T^{-PRE}(K_{2i}, X_{2i})) \in \{&H^{2\mathrm{CYC}}(\mathrm{C}_T^{-PRE}(K_{2i'}, X_{2i'})), \\ &H^{2\mathrm{CYC}}((\mathrm{C}^{-LNK} \circ \mathrm{C}_B^{-PRE})(K_{2i'+1}, X_{2i'+1}))\} \end{aligned}$$

or

$$\begin{aligned} H^{2\mathrm{CYC}}((\mathrm{C}^{-LNK} \circ \mathrm{C}_B^{-PRE})(K_{2i+1}, X_{2i+1})) \in \\ \{H^{2\mathrm{CYC}}(\mathrm{C}_T^{-PRE}(K_{2i'}, X_{2i'})), \\ H^{2\mathrm{CYC}}((\mathrm{C}^{-LNK} \circ \mathrm{C}_B^{-PRE})(K_{2i'+1}, X_{2i'+1}))\}. \end{aligned}$$

Assuming a forward query Q_{2i}, this condition is equivalent to

$$\begin{aligned} &[\mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_{2i}, X_{2i}), Y_{2i}), \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_{2i+1}, X_{2i+1}), Y_{2i+1})] \\ &\quad = [\mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_{2i'}, X_{2i'}), Y_{2i'}), \\ &\qquad \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_{2i'+1}, X_{2i'+1}), Y_{2i'+1})] \quad \text{or} \qquad (4.2) \\ &[\mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_{2i}, X_{2i}), Y_{2i}), \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_{2i+1}, X_{2i+1}), Y_{2i+1})] \\ &\quad = [\mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_{2i'+1}, X_{2i'+1}), Y_{2i'+1}), \\ &\qquad \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_{2i'}, X_{2i'}), Y_{2i'})] \quad \text{or} \qquad (4.3) \\ &[\mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_{2i+1}, X_{2i+1}), Y_{2i+1}), \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_{2i}, X_{2i}), Y_{2i})] \\ &\quad = [\mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_{2i'}, X_{2i'}), Y_{2i'}), \\ &\qquad \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_{2i'+1}, X_{2i'+1}), Y_{2i'+1})] \quad \text{or} \qquad (4.4) \\ &[\mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_{2i+1}, X_{2i+1}), Y_{2i+1}), \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_{2i}, X_{2i}), Y_{2i})] \\ &\quad = [\mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_{2i'+1}, X_{2i'+1}), Y_{2i'+1}), \\ &\qquad \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_{2i'}, X_{2i'}), Y_{2i'})]. \qquad (4.5) \end{aligned}$$

Note that (4.2) is equal to (4.5) and (4.3) is equal to (4.4) if $\mathrm{C}_T^{POST} = \mathrm{C}_B^{POST}$. In this case, it follows that for $2q < N$

$$\Pr[\mathrm{C}_i] \le \frac{2(i-1)}{(N-(2i-2))(N-(2i-1))} \le \frac{2q}{(N-2q)^2}. \qquad (4.6)$$

Assuming $\mathrm{C}_T^{POST} \neq \mathrm{C}_B^{POST}$ we obtain

$$\Pr[\mathrm{C}_i] \le \frac{4(i-1)}{(N-(2i-2))(N-(2i-1))} \le \frac{4q}{(N-2q)^2}. \qquad (4.7)$$

For unifying the treatment of these two cases, we set $a = 1$ if $\mathrm{C}_T^{POST} = \mathrm{C}_B^{POST}$ and $a = 2$ otherwise. Let C be the event that a colliding pair of non-matching inputs is found for $H^{2\mathrm{CYC}}$ with q (pairs) of queries. Then,

$$\Pr[\mathrm{C}] \leq \sum_{i=2}^{q} \Pr[\mathrm{C}_j] \leq \sum_{i=2}^{q} \frac{2q \cdot a}{(N-2q)^2} \leq \frac{2aq^2}{(N-2q)^2}.$$

Now, assuming the i-th query, let $\hat{\mathrm{C}}_i$ be the event that a colliding pair of matching inputs has been found for $H^{2\mathrm{CYC}}$. It follows, that

$$\Pr[\hat{\mathrm{C}}_i] \leq \frac{2}{N-2i}.$$

Let $\hat{\mathrm{C}}$ be the event that a colliding pair of matching inputs is found for $H^{2\mathrm{CYC}}$ with q (pairs) of queries. Then,

$$\Pr[\hat{\mathrm{C}}] \leq \sum_{i=2}^{q} \Pr[\hat{\mathrm{C}}_i] \leq \frac{2q}{N-2q}.$$

Since $\mathbf{Adv}_{H^{2\mathrm{CYC}}}^{\mathrm{COLL}}(q) = \Pr[\mathrm{C} \vee \hat{\mathrm{C}}] \leq \Pr[\mathrm{C}] + \Pr[\hat{\mathrm{C}}]$, the claim follows. □

4.2.2. Proof of Theorem 4.9 – Collision Resistance of Cyclic-DL $(c > 2)$

Analysis Overview. In this section we omit some details that were already discussed in the proof of Abreast-DM in Section 4.1.3 since the basic arguments are the same. Again, we analyze if the queries made by the adversary contain the means for constructing a collision of the compression function H^{CYC} by upper bounding the probability of an adversary asking for a query that can be used as the *final* query to complete a collision.

The Cycle in Cyclic-DL. First assume that the adversary asks for a query $Q_{ci} = (K_{ci}, X_{ci}, Y_{ci})$. The query index $c \cdot i$ for the i-th query of the adversary is – similar as in the case of Abreast-DM – due to the $c-1$ additional free queries the adversary is given for any mounted query. Now, consider the case where the query is used in the top row. Let $P_1 = (K_{ci}, X_{ci})$ and $P_2 = (K_{ci+1}, X_{ci+1}) = \sigma(K_{ci}, X_{ci})$. After having received the answer to its query Q_{ci}, the adversary is given for free the corresponding forward query in the bottom row $Q_{ci+1} = (K_{ci+1}, X_{ci+1}, Y_{ci+1})$, $Y_{ci+1} = E_{K_{ci+1}}(X_{ci+1})$. With these two queries, the adversary is able to compute one output of the compression function $(V, \widehat{V}) = H^{\mathrm{CYC}}(\mathrm{C}_T^{-PRE}(P_1))$. The adversary can reuse the query Q_{ci+1} in the top row as a starting point to compute a new result of H^{CYC}. We now give the adversary for free the corresponding bottom row forward query, Q_{ci+2}, assuming that Q_{ci+1} is used in the top row. For the query Q_{ci+2}, we have $P_3 = (K_{ci+2}, X_{ci+2}) = \sigma(P_2)$. Our main observation is that $P_3 = \sigma(P_2) = \sigma^2(P_1)$. Let $c = |\sigma|$ denote the cycle length of H^{CYC}. This process can be continued. The adversary is given for free the forward queries $Q_{ci+3}, \ldots, Q_{ci+c-1}$ as shown in Table 4.2 in more detail. A cycle is formed since $P_c = \sigma(P_{c-1}) = \cdots = \sigma^c(P_1) = P_1$ and therefore $Q_{ci+c} = Q_{ci}$. The queries forming the cycle are shown in Figure 4.3.

Analysis Details. Fix an adversary $\mathcal{A}$ asking a total of q backward and forward queries to its oracle E. Let $\mathrm{Coll}^{\mathrm{CYC}}(\mathcal{Q})$ be the event that the adversary is able to construct a collision of H^{CYC} using the queries in $\mathcal{Q}$. The term 'last query' means the latest query made by the adversary and is always given index $c \cdot i$ and denoted by Q_{ci}. We examine the adversary's directly asked queries ($d = 0$) and the free queries ($d = 1, 2, \ldots, c-1$) one at a time as the adversary gets hold of them. A query $Q_m = (K_m, X_m, Y_m)$ is successful, if it can be used to form a collision using other queries contained in the query history $\mathcal{Q}_m$.

Similar to the proof of Abreast-DM, we now proceed to upper

$H^{\text{CYC}}(\cdot)$	Query #	Plaintext	Key	Ciphertext
$\mathrm{C}_T^{-PRE}(P_1)$	ci (*) $ci+1$	X_{ci} X_{ci+1}	K_{ci} K_{ci+1}	Y_{ci} Y_{ci+1}
$\mathrm{C}_T^{-PRE}(\sigma(P_1))$	$(ci+1)$ $ci+2$	X_{ci+1} X_{ci+2}	K_{ci+1} K_{ci+2}	Y_{ci+1} Y_{ci+2}
$\mathrm{C}_T^{-PRE}(\sigma^2(P_1))$	$(ci+2)$ $ci+3$	X_{ci+2} X_{ci+3}	K_{ci+2} K_{ci+3}	Y_{ci+2} Y_{ci+3}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathrm{C}_T^{-PRE}(\sigma^{c-2}(P_1))$	$(ci+c-2)$ $ci+c-1$	X_{ci+c-2} X_{ci+c-1}	K_{ci+c-2} K_{ci+c-1}	Y_{ci+c-2} Y_{ci+c-1}
$\mathrm{C}_T^{-PRE}(\sigma^{c-1}(P_1))$	$(ci+c-1)$ (ci)	X_{ci+c-1} X_{ci}	K_{ci+c-1} K_{ci}	Y_{ci+c-1} Y_{ci}

Table 4.2.: Starting with query Q_{ci}, $K_{ci},(X_{ci},*)_{fwd}$ or $(K_{ci},*,Y_{ci})_{bwd}$, the adversary is given $c-1$ forward queries $Q_{ci+1},Q_{ci+2},\ldots,Q_{ci+c-1}$ for free. In total, it is able to compute c results of H^{CYC} by using these c queries. The notations used in the table are given in the text.

bound $\Pr[\textsc{Coll}^{\text{CYC}}(\mathcal{Q})]$ by exhibiting some predicates $\textsc{Win}_0(\mathcal{Q}),\ldots$ $\textsc{Win}_{q-1}(\mathcal{Q})$ such that $\textsc{Coll}^{\text{CYC}} \implies \textsc{Win}_0(\mathcal{Q}) \vee \ldots \vee \textsc{Win}_{q-1}(\mathcal{Q})$. Since the adversary mounts q queries in total, we informally say that $\textsc{Win}_i(\mathcal{Q})$, $0 \le i \le q-1$, holds if the adversary finds a collision after mounting the i-th query, $0 \le i \le q-1$, using at least two of the following queries $Q_{ci},\ldots,Q_{ci+c-1}$ conditioned on the fact that the adversary has not been successful before. It follows, that $\Pr[\textsc{Coll}^{\text{CYC}}(\mathcal{Q})] \le \Pr[\textsc{Win}_0(\mathcal{Q})] + \ldots + \Pr[\textsc{Win}_{q-1}(\mathcal{Q})]$. For simplicity, we assume again that the free queries are always given in 'ascending' order as in in Table 4.2.

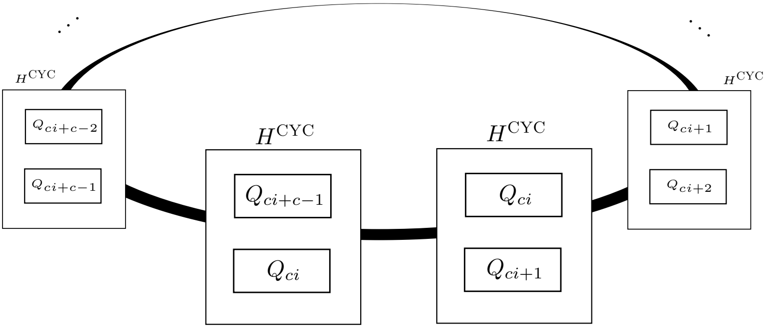


Figure 4.3.: Visualization of the query cycle of Cyclic-DL: An adversary uses the c queries to compute the complete output of c compression functions H^{CYC}.

Definition 4.11. *We say that a pair of queries (Q_a, Q_b) is successful in the query set $\mathcal{Q}$, if the query Q_a is used in the top row, the query Q_b is used the bottom-row of H^{CYC}, and there exists a pair of queries $Q_j, Q_k \in \mathcal{Q}$ that can also be used to compute an output of H^{CYC} such that a collision for H^{CYC} has been found:*

$$\mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_a, X_a), Y_a) = \mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_j, X_j), Y_j) \qquad \textit{and}$$
$$\mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_b, X_b), Y_b) = \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_k, X_k), Y_k).$$

Definition 4.12. *Let $d = 0, \ldots, c-1$, $d' = d+1 \mod c$, $\widetilde{d} = \max(d, d')$. We say $\textsc{CollFit}_i^d(\mathcal{Q})$ if (i) the pair of queries $(ci + d, ci + d')$ is successful in $\mathcal{Q}_{ci+\widetilde{d}}$ and (ii) the adversary had not been successful for $0 \leq t \leq d-1$: $\neg\textsc{CollFit}_i^t(\mathcal{Q})$. The probability is measured over the post-outputs of the newly received queries $(6i+d)$ and $6i+d'$.*

The predicates $\textsc{Win}_i(\mathcal{Q})$ are defined as follows:

Definition 4.13.

$$\begin{aligned}\textsc{Win}_i(\mathcal{Q}) =\neg &\left(\bigvee_{0\le j\le i-1} \textsc{Win}_j(\mathcal{Q})\right)\\ &\wedge \left(\textsc{CollFit}_i^1(\mathcal{Q}) \vee \ldots \vee \textsc{CollFit}_i^c(\mathcal{Q})\right).\end{aligned}$$

We now show that our case analysis is complete.

Lemma 4.14. $\textsc{Coll}^{\textsc{cyc}}(\mathcal{Q}) \Longrightarrow \textsc{Win}_0(\mathcal{Q}) \vee \ldots \vee \textsc{Win}_{q-1}(\mathcal{Q})$.

This proof is omitted as it is essentially the same as the proof of Lemma 4.6. The only difference is that d is not chosen from the set $\{0, 1, \ldots, 5\}$ but from the set $\{0, 1, \ldots, c-1\}$. □

Since $\Pr[\textsc{Coll}^{\textsc{cyc}}(\mathcal{Q})] \le \sum_{j=0}^{q-1} \textsc{Win}_j(\mathcal{Q})$ it follows that

$$\Pr[\textsc{Coll}^{\textsc{cyc}}(\mathcal{Q})] \le \sum_{i=0}^{q-1}\sum_{d=0}^{c-1} \textsc{CollFit}_i^d(\mathcal{Q}). \tag{4.8}$$

We now proceed to upper bound $\Pr[\textsc{CollFit}_i^d(\mathcal{Q})]$.

Lemma 4.15. *Let* $0 < i \le q-1$ *and* $0 \le d \le c-1$. *Then*

$$\Pr[\textsc{CollFit}_i^d(\mathcal{Q})] \le \frac{ci}{(N-ci)^2}.$$

Proof. Let $d' = d + 1 \ mod \ c$. The output of the compression function H^{CYC} is determined by the queries $Q_{ci+d} = (K_{ci+d}, X_{ci+d}, Y_{ci+d})$ and $Q_{6i+d'} = (K_{ci+d'}, X_{ci+d'}, Y_{ci+d'})$ and the bijective pre-processing and post-processing functions C_T^{PRE}, C_T^{POST}, C_B^{PRE}, C_B^{POST}. To form a collision, two queries, Q_j and Q_k, are needed that can be chosen from at most $c(i+1)$ queries in $\mathcal{Q}_{c(i+1)-1}$. The adversary can use them to compute the output of $< c(i+1)$ compression functions H^{CYC}. Therefore,

$$\Pr[\textsc{CollFit}_i^d(\mathcal{Q})] \leq \frac{c(i+1)}{(N - c(i+1))^2}.$$

□

Using (4.8) we get the following upper bound for any $q \geq 1$ and $N > cq$

$$\begin{aligned}\Pr[\textsc{Coll}^{\textsc{cyc}}(\mathcal{Q})] &\leq \sum_{i=0}^{q-1} \sum_{d=0}^{c-1} \frac{c(i+1)}{(N - c(i+1))^2} \leq \sum_{i=1}^{q} \sum_{d=0}^{c-1} \frac{ci}{(N - ci)^2} \\ &\leq \sum_{i=1}^{q} \frac{c^2 i}{(N - ci)^2} \leq \frac{c^2 \cdot q^2 \cdot \frac{1}{2}}{(N - cq)^2} \leq \frac{c^2}{2} \left(\frac{q}{N - cq} \right)^2.\end{aligned}$$

This completes our proof of Theorem 4.9. □

4.3. Applications

4.3.1. Add/k-DM (cycle length 2^k)

We give a very elegant and efficient method to instantiate a compression function with cycle length $c = 2^k$ for any $k \geq 1$. This construction is very similar to Hirose-DM. It is shown in Figure 4.4 and formally given in Definition 4.16.

Definition 4.16. *Let $H^{\textsc{Add/k}} : \{0,1\}^n \times \{0,1\}^{2n} \to \{0,1\}^{2n}$ be a compression function such that $(V, \widehat{V}) = H^{\textsc{Add/k}}(M, U, \widehat{U})$ where $M, U, \widehat{U} \in \{0,1\}^n$ and let $k \in \mathbb{N}$ such that $1 \leq k < n$. $H^{\textsc{Add/k}}$ uses a blockcipher $E \in \textsc{Block}(2n, n)$ as follows:*

$$\begin{aligned} V &= E_{\widehat{U}\|M}(U) \oplus U \\ \widehat{V} &= E_{\widehat{U}\|M}(U \boxplus 2^{n-k}) \oplus U \boxplus 2^{n-k}. \end{aligned}$$

The symbol $\boxplus$ denotes an addition modulo 2^n.

Lemma 4.17. *The compression function $H^{\textsc{Add/k}}$ is a* Cyclic-DL *compression function with a cycle length of $c = 2^k$.*

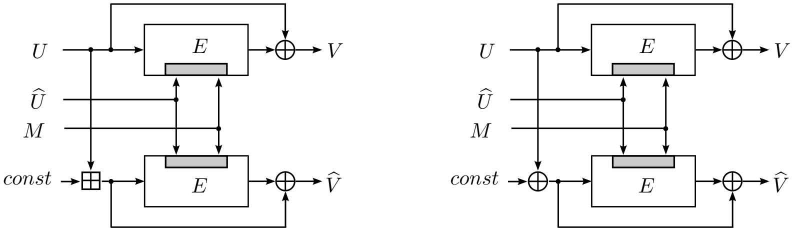
Proof. It is trivial to choose the pre-processing function and post-processing function as required. It follows that $\sigma(K, X) = (K, X \boxplus 2^{n-k})$. Using simple arithmetics,

$$\underbrace{(\sigma \circ \cdots \circ \sigma)}_{2^k \text{ times}}(K, X) = (K, X \boxplus 2^k \cdot 2^{n-k}) = (K, X).$$

gives our claim since $X \in \{0,1\}^n$. □

Therefore we can apply Theorem 4.8 for $k = 1$ or Theorem 4.9 if $k \geq 2$.

Corollary 4.18. *No adversary asking less than 2^{n-k-1} queries can have more than a chance of $1/2$ in finding a collision for the compression function $H^{\textsc{Add/k}}$ for any $1 < k < n$.*



Add/k-DM, cycle-length $c = 2^k$

Hirose-DM, cycle length $c = 2$, $const \neq 0$

Figure 4.4.: Left: Cyclic Compression Function with cycle length 2^k, $k > 1$. Right: (for comparison) Hirose-DM with a cycle length of 2.

Proof. As the cycle length c is equal to 2^k (Lemma 4.17), it follows, using Theorem 4.9, that

$$\mathbf{Adv}^{\mathrm{COLL}}_{H^{\textsc{Add/k}}}(q) = \frac{2^{2k}}{2}\left(\frac{q}{2^{n-1}}\right)^2.$$

By applying $\mathbf{Adv}^{\mathrm{COLL}}_{H^{\textsc{Add/k}}}(q) = 1/2$ and solving after q one obtains

$$q(k) = \sqrt{2^{2n-2k-2}} = 2^{n-k-1}.$$

□

For example, when considering $n = 128$, we can for derive without effort, that no adversary asking less than 2^{122} queries can have more than a chance of $1/2$ in finding a collision for the compression function $H^{\textsc{Add}/5}$. The compression function $H^{\textsc{Add}/5}$ has a cycle length of $2^5 = 32$.

4.3.2. Cube-DM (cycle length = 3)

The 'most optimal' result in terms of security – at least in the class Cyclic-DL as given by our theorems – can be achieved by using a compression function that has a cycle length of 3. The approach is slightly different compared to ADD/K-DM as neither additions modulo 2^n nor XOR can be used to create a permutation σ with $|\sigma| = 3$. The guiding idea is to use a message space Ω such that $|\Omega|$ is evenly divisible by three. This construction is shown in Figure 4.5 and is given in Definition 4.20.

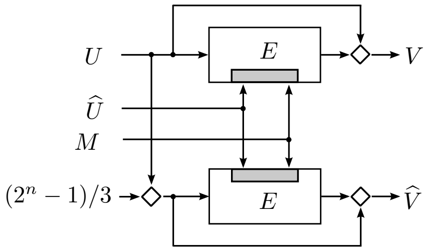


Figure 4.5.: CUBE-DM, a compression function with cycle length $|\sigma| = 3$, the symbol $\diamond$ denotes an addition modulo $2^n - 1$.

Definition 4.19. *Let* $E \in \text{BLOCK}(2n, n)$ *and let* $\Omega = \{0,1\}^n - \{1^n\}$, i.e., $|\Omega| = 2^n - 1$. *The blockcipher* $E' : (\Omega \times \{0,1\}^n) \times \Omega \to \Omega$, *where* $\Omega \times \{0,1\}^n$ *is the key space and is defined as*

$$E'_K(X) = \begin{cases} E_K(X), & \text{if } E_K(X) \neq 1^n \\ E_K(E_K(X)), & \text{else.} \end{cases}$$

This definition of the blockcipher E' ensures that $E'_K(X) \in \Omega$ for any value of $X \in \Omega$: since E is a permutation, it follows that E' is a permutation. It is easy to see that, for n even, $|\Omega|$ is divisible by three as

$$|\Omega| \; mod\, 3 \equiv 2^n - 1 \; mod\; 3 \equiv (2 \cdot 2)^{n'} - 1 \; mod\, 3 \equiv 0 \; mod\, 3. \qquad (4.9)$$

Definition 4.20. *Let* $\Omega = \{0,1\}^n \backslash \{1^n\}$ *and* $N = |\Omega| = 2^n - 1$. *Let* $H^{\text{Cube}} : \Omega^2 \times \{0,1\}^n \to \Omega^2$ *be a compression function such that* $(V, \widehat{V}) = H^{\text{Cube}}(M, U, \widehat{U})$ *where* $U, \widehat{U}, V, \widehat{V} \in \Omega$ *and* $M_i \in \{0,1\}^n$. *Furthermore, let const* $= (2^n - 1)/3$ *and* $\diamond$ *be the addition modulo* $2^n - 1$. *Now* H^{Cube} *is built upon a blockcipher* E' *as in Definition 4.19:*

$$\begin{aligned} V &= E'_{\widehat{U}\|M}(U) \diamond U \\ \widehat{V} &= E'_{\widehat{U}\|M}(U \diamond const) \diamond U \diamond const. \end{aligned}$$

Actually, we would theoretically need a slightly generalized variant of Cyclic-DL (operating not only over binary sets, but being also able to work over arbitrary sets as Ω). But it is easy to see that this extension is very simple to do, so we skip the details here. Note that in the original publication [61] the proof was given for the generic set Ω.

The compression function H^{Cube} has a cycle length of 3 using $\sigma(K, X) = (K, X \diamond (2^n - 1)/3)$ and since

$$(\sigma \circ \sigma \circ \sigma)(K, X) = (K, X \diamond 3 \cdot \frac{2^n - 1}{3} \; mod\,(2^n - 1)) = (K, X).$$

The operation $\diamond$ is trivially efficient since a simple 'if' and an 'addition' suffice for an implementation. Also, the implementation of E' is not assumed to cost any practically relevant performance.

5 Serial-DL, Tandem-DM and Applications

5.1. Serial-DL

5.1.1. Collision Security Results

Theorem 5.1. *Let H^{SER} be a* Serial-DL *compression function as given in Section 1.4.3. Let α, β, n be such that $\alpha > e$ and let $N = 2^n$, $N'' = N - 2q$ and $\tau = N''\alpha/2q$. Then*

$$\mathbf{Adv}^{\textsc{Coll}}_{H^{\mathrm{SER}}}(q) \leq 2q(2\alpha + 1)/N'' + 2q\zeta + L, \tag{5.1}$$

where

$$L = 4 \cdot \left(q2^n e^{\tau 2q(1-\ln\tau)/N''} + q^2/(\beta N'')\right)$$

and ζ is from Definition 1.16.

Since the proof of this theorem is rather brief, and based on several propositions that are discussed shortly, we postpone it to Page 82. Numerical results, *e.g.*, for TANDEM-DM, are discussed in Section 5.3.1 and 5.3.2.

We also show that Theorem 5.1 implies the following asymptotic bound.

Theorem 5.2. *Let* $q = 2^{0.93n-\epsilon}$ *where* $\epsilon \geq 0$ *and* H^{SER} *be a* SERIAL-DL *compression function with* $\zeta \leq 1/N''$. *Then* $\mathbf{Adv}^{\mathrm{COLL}}_{H^{\mathrm{SER}}}(q) \to 0$ *as* $n \to \infty$.

Corrections since Initial Publication The collision security proof of SERIAL-DL in [47] had a flaw in the analysis of (what now is) Case 1 in Proposition 5.5. We 'repaired' it using a technique first given in [107]. As a consequence of this technique, we had to give the adversary a second query for each query asked.

5.1.2. Proof Model

We analyze whether the queries to the oracle E made by the adversary *can* be used for constructing a collision of the compression function H^{SER}. We look to see whether there exist four not necessarily distinct queries that form a collision (cf. Figure 5.1).

We start our discussion from an arbitrary collision-finding adversary $\mathcal{A}$ which asks q queries. We then construct an adversary $\mathcal{A}'$ that simulates $\mathcal{A}$ and asks additional queries as follows: First, assume that $\mathcal{A}$ has launched a forward query $E_{K_B}(X_B)$. Then $\mathcal{A}'$ computes $(K_T, Y_T) = \mathtt{SER}^{-1}(K_B, X_B)$ and launches an additional backward query $E^{-1}_{K_T}(Y_T)$. In the case of a backward query $E^{-1}_{K_T}(X_T)$, $\mathcal{A}'$ computes $(K_B, X_B) = \mathtt{SER}(K_T, Y_T)$ and launches an additional forward query $E_{K_B}(X_B)$. If $\mathcal{A}$ ever makes a query to which $\mathcal{A}'$ already knows the answer from its query history, $\mathcal{A}'$ ignores this query and returns

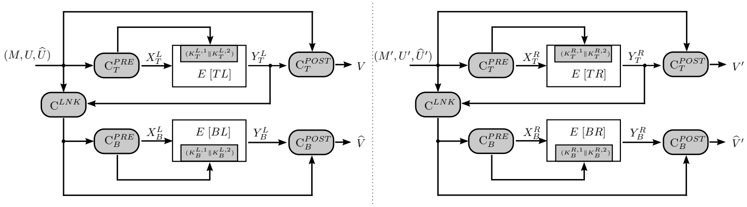


Figure 5.1.: The double length compression function H^{SER} using a blockcipher $E \in \textsc{Block}(k,n)$. The four possible positions a query can be used in are denoted by TL, BL, TR, BR which are abbreviations for *top-left*, *bottom-left*, *top-right*, and *bottom-right*.

the 'old' result that has been stored in its query history. So $\mathcal{A}'$ never makes a query to which it knows the answer. Let $\mathcal{Q}'$ be the query history of $\mathcal{A}'$ and $\mathcal{Q}$ be the query history of $\mathcal{A}$. Then, $\mathcal{Q} \subset \mathcal{Q}'$ and $|\mathcal{Q}'| = q_* \leq 2q$. Our analysis now focuses on the adversary $\mathcal{A}'$ with its query history $\mathcal{Q}'$.

To upper bound the probability of the adversary obtaining queries than can be used for a collision, we upper bound the probability of the adversary $\mathcal{A}'$ making a query that can be used as the final query to complete such a collision. Let $\mathcal{Q}'_i$ denote the set of the first i queries made by the adversary $\mathcal{A}$', $i \leq 2q$. So, from the point of view of $\mathcal{A}'$, it responds to two events: either $\mathcal{A}$ asks for a query that is already in $\mathcal{Q}'$ by just returning it or $\mathcal{A}$ asks for a query that is *not* in $\mathcal{Q}'$. Since the success probability in the first case is zero we analyze in the following the latter case.

In our analysis, we examine the queries of the adversary one at a time (either forward or backward) as they are placed. We denote by the term *last query* the latest query made by the adversary. This query is always given the index i. Therefore, for each i, $1 \leq i \leq 2q$, we upper bound the probability that the answer to the adversary's i-th query, $(K_i, X_i)_{fwd}$ or $(K_i, Y_i)_{bwd}$, allows the adversary to use this query to complete the collision. In the latter case, the last query is called *successful* and the attack is given to the adversary. As the probability depends on the first $i-1$ queries, we need to make sure that the adversary has not already been too lucky with these. Being lucky is explicitly defined in (5.2), *e.g.*, it means – among others – that there exists a large subset of the first $i-1$ queries that have equal post-output. Our upper bound thus breaks down into two pieces: an upper bound for the probability of the adversary getting lucky and the probability of the adversary ever making a successful i-th query, conditioned on the fact that the adversary has not yet become lucky by its $(i-1)$-th query.

We say $\textsc{Coll}(\mathcal{Q}')$ if the adversary $\mathcal{A}'$ wins in either finding a collision *or* gets lucky in one of the predefined ways specified shortly. (So 'winning' does not necessarily imply that a collision has been found.) We upper bound $\Pr[\textsc{Coll}(\mathcal{Q}')]$ by exhibiting predicates $\textsc{Lucky}(\mathcal{Q}')$, $\textsc{Win}_{\mathrm{TL}}(\mathcal{Q}')$, $\textsc{Win}_{\mathrm{BL}}(\mathcal{Q}')$, $\textsc{Win}_{TLBL}(\mathcal{Q}')$ and $\textsc{Win}_{TLBR}(\mathcal{Q}')$ such that

$$\begin{aligned}\textsc{Coll}(\mathcal{Q}') \Rightarrow \textsc{Lucky}(\mathcal{Q}') \vee \textsc{Win}_{\mathrm{TL}}(\mathcal{Q}') \vee \textsc{Win}_{\mathrm{BL}}(\mathcal{Q}')\\ \vee \textsc{Win}_{TLBL}(\mathcal{Q}') \vee \textsc{Win}_{TLBR}(\mathcal{Q}').\end{aligned}$$

We then proceed to derive separately upper bounds for the probabilities $\textsc{Lucky}(\mathcal{Q}')$, $\textsc{Win}_{\mathrm{TL}}(\mathcal{Q}')$, $\textsc{Win}_{\mathrm{BL}}(\mathcal{Q}')$, $\textsc{Win}_{TLBL}(\mathcal{Q}')$ and $\textsc{Win}_{TLBR}(\mathcal{Q}')$. The union bound finally gives

$$\begin{aligned}\Pr[\textsc{Coll}(\mathcal{Q})] \leq & \Pr[\textsc{Lucky}(\mathcal{Q}')] + \Pr[\textsc{Win}_{\mathrm{TL}}(\mathcal{Q}')] + \\ & \Pr[\textsc{Win}_{\mathrm{BL}}(\mathcal{Q}')] + \Pr[\textsc{Win}_{TLBL}(\mathcal{Q}')] + \\ & \Pr[\textsc{Win}_{TLBR}(\mathcal{Q}')].\end{aligned}$$

Clearly, it suffices for our purposes to upper bound $\Pr[\textsc{Coll}(\mathcal{Q}')]$ since

$$\Pr[\textsc{Coll}(\mathcal{Q})] \leq \Pr[\textsc{Coll}(\mathcal{Q}')].$$

To formally state these predicates, some additional definitions are convenient. Let $\texttt{NumEqual}_{\mathrm{T}}(\mathcal{Q}')$, $\texttt{NumEqual}_{\mathrm{B}}(\mathcal{Q}')$ be functions defined on query sequences $\mathcal{Q}'$ of length q_* as follows:

$$\texttt{NumEqual}_{\mathrm{T}}(\mathcal{Q}') = \max_{Z\in\{0,1\}^n} |\{i : \mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_i, X_i), Y_i) = Z\}|,$$

$$\texttt{NumEqual}_{\mathrm{B}}(\mathcal{Q}') = \max_{Z\in\{0,1\}^n} |\{i : \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_i, X_i), Y_i) = Z\}|.$$

They give the maximum size of a set of queries in $\mathcal{Q}'$ whose post-outputs are all the same (for the top row and bottom row, respectively). Let $\texttt{NumColl}_{\mathrm{T}}(\mathcal{Q}')$, $\texttt{NumColl}_{\mathrm{B}}(\mathcal{Q}')$ be also defined on query sequences $\mathcal{Q}'$ of length q_* as

$$\begin{aligned}\texttt{NumColl}_{\mathrm{T}}(\mathcal{Q}') = |\{(i,j) \in \{1,\dots,q_*\}^2, i \neq j :&\\ \mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_i, X_i), Y_i) &=\\ \mathrm{C}_T^{POST}(\mathrm{C}_T^{-PRE}(K_j, X_j), Y_j)\}|&,\\ \texttt{NumColl}_{\mathrm{B}}(\mathcal{Q}') = |\{(i,j) \in \{1,\dots,q_*\}^2, i \neq j :&\\ \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_i, X_i), Y_i) &=\\ \mathrm{C}_B^{POST}(\mathrm{C}_B^{-PRE}(K_j, X_j), Y_j)\}|&.\end{aligned}$$

They give the number of ordered pairs of distinct queries in $\mathcal{Q}'$ which have the same post-outputs, again for the top row and bottom row, respectively.

We now define the event $\textsc{Lucky}(\mathcal{Q}')$ as

$$\begin{aligned}\textsc{Lucky}(\mathcal{Q}') =&(\texttt{NumEqual}_{\mathrm{T}}(\mathcal{Q}') > \alpha) \vee (\texttt{NumEqual}_{\mathrm{B}}(\mathcal{Q}') > \alpha) \vee\\ &(\texttt{NumColl}_{\mathrm{T}}(\mathcal{Q}') > \beta) \vee (\texttt{NumColl}_{\mathrm{B}}(\mathcal{Q}') > \beta)\ , \quad (5.2)\end{aligned}$$

where α and β are the constants from Theorem 5.1. These constants are chosen depending on n and q_* by a numerical optimization process. If α and β are chosen larger, $\Pr[\textsc{Lucky}(\mathcal{Q}')]$ diminishes. The other events consider mutually exclusive cases on how to find a collision for the compression function. These cases are formalized by the following four predicates.

Fit$_{TL}(\mathcal{Q}')$**:** The last query is used only once: in position TL. This is equivalent to the case where the last query is used in position TR.

Fit$_{BL}(\mathcal{Q}')$**:** The last query is used only once: in position BL or (equivalent) BR.

Fit$_{TLBL}(\mathcal{Q}')$**:** The last query is used twice in a collision: either TL and BL or (equivalent) TR and BR.

Fit$_{TLBR}(\mathcal{Q}')$**:** The last query is used twice in a collision: either TL and BR or (equivalent) TR and BL.

In Proposition 5.3, we prove that these four predicates do cover all possible cases of a collision. For practical purposes we now define some additional predicates as follows:

$$\begin{aligned}
\textsc{Win}_{\mathrm{TL}}(\mathcal{Q}') &= \neg\textsc{Lucky}(\mathcal{Q}') \wedge \textsc{Fit}_{\mathrm{TL}}(\mathcal{Q}'),\\
\textsc{Win}_{\mathrm{BL}}(\mathcal{Q}') &= \neg(\textsc{Lucky}(\mathcal{Q}') \vee \textsc{Fit}_{\mathrm{TL}}(\mathcal{Q}')) \wedge \textsc{Fit}_{\mathrm{BL}}(\mathcal{Q}'),\\
\textsc{Win}_{TLBL}(\mathcal{Q}') &= \neg(\textsc{Lucky}(\mathcal{Q}') \vee \textsc{Fit}_{\mathrm{TL}}(\mathcal{Q}') \vee \textsc{Fit}_{\mathrm{BL}}(\mathcal{Q}'))\\
&\quad \wedge \textsc{Fit}_{TLBL}(\mathcal{Q}'),\\
\textsc{Win}_{TLBR}(\mathcal{Q}') &= \neg(\textsc{Lucky}(\mathcal{Q}') \vee \textsc{Fit}_{\mathrm{TL}}(\mathcal{Q}') \vee \textsc{Fit}_{\mathrm{BL}}(\mathcal{Q}')\\
&\quad \vee \textsc{Fit}_{TLBL}(\mathcal{Q}')) \wedge \textsc{Fit}_{TLBR}(\mathcal{Q}').
\end{aligned}$$

Proposition 5.3.

$$\begin{aligned}
\textsc{Coll}(\mathcal{Q}') \Rightarrow &\textsc{Win}_{\mathrm{TL}}(\mathcal{Q}') \vee \textsc{Win}_{\mathrm{BL}}(\mathcal{Q}') \vee \textsc{Win}_{TLBL}(\mathcal{Q}')\\
&\vee \textsc{Win}_{TLBR}(\mathcal{Q}').
\end{aligned}$$

Proof. We can assume that the adversary has not been lucky, *i.e.*, $\neg\textsc{Lucky}(\mathcal{Q}')$. Then it is easy to see that

$$\begin{aligned}
&\textsc{Fit}_{\mathrm{TL}}(\mathcal{Q}') \vee \textsc{Fit}_{\mathrm{BL}}(\mathcal{Q}') \vee \textsc{Fit}_{TLBL}(\mathcal{Q}') \vee \textsc{Fit}_{TLBR}(\mathcal{Q}') \Longrightarrow\\
&\textsc{Win}_{\mathrm{TL}}(\mathcal{Q}') \vee \textsc{Win}_{\mathrm{BL}}(\mathcal{Q}') \vee \textsc{Win}_{TLBL}(\mathcal{Q}') \vee \textsc{Win}_{TLBR}(\mathcal{Q}').
\end{aligned}$$

It is sufficient to show that $\textsc{Coll}(\mathcal{Q}') \Rightarrow \textsc{Fit}_{\mathrm{TL}}(\mathcal{Q}') \vee \textsc{Fit}_{\mathrm{BL}}(\mathcal{Q}') \vee \textsc{Fit}_{TLBL}(\mathcal{Q}') \vee \textsc{Fit}_{TLBR}(\mathcal{Q}')$. Now, assume $\textsc{Coll}(\mathcal{Q}')$ in the adversary's i-th query. So the adversary has not been able to find a successful query prior to the i-th query but is successful now. Hence, it has found queries Q_i, Q_j, Q_k, Q_l, including queries from the query history $\mathcal{Q}'$, for some integers $1 \leq j, k, l \leq i$, such that a collision can be constructed.

For this, first assume that the last query, *i.e.*, the i-th query, is used *once* in the collision. If it is used in TL (or TR), then $\textsc{Fit}_{\mathrm{TL}}(\mathcal{Q}')$. If it is used in BL (or BR), then $\textsc{Fit}_{\mathrm{BL}}(\mathcal{Q}')$. Now assume that the query is used *twice* in the collision. If it is used in TL and BL (or TR and BR), then $\textsc{Fit}_{TLBL}(\mathcal{Q}')$. If it is used in TL and BR (or BL and TR), then $\textsc{Fit}_{TLBR}(\mathcal{Q}')$. We note that a query cannot be used twice in one row (top row or bottom row) since then we would not have found a collision. This is due to the fact that the top-row query uniquely determines the bottom-row query and vice versa. The same argument also holds in the case that the last query is used more than twice. This concludes our analysis as no cases are left. □

The next step is to upper bound separately the probability of occurrence of any of the predicates: $\Pr[\textsc{Lucky}(\mathcal{Q}')]$, $\Pr[\textsc{Win}_{\mathrm{TL}}(\mathcal{Q}')]$, $\Pr[\textsc{Win}_{\mathrm{BL}}(\mathcal{Q}')]$, $\Pr[\textsc{Win}_{TLBL}(\mathcal{Q}')]$, and $\Pr[\textsc{Win}_{TLBR}(\mathcal{Q}')]$.

Proposition 5.4. *Let α and β as in Theorem 5.1 and let $N^* = 2^n - q_*, \alpha > e$ and $\tau = N^*\alpha/q_*$. Then*

$$\Pr[\textsc{Lucky}(\mathcal{Q}')] \leq 2 \cdot \left(q_* 2^n e^{\tau q_* (1-\ln\tau)/N^*} + q_*^2/(\beta N^*)\right).$$

Proof. The proof is somewhat technical and is postponed to Section 5.4. Also, the methods are well known in literature. □

We note that the multiplication factor '2' can be omitted if the post processing functions of the top and bottom row are equal, *i.e.*, $\mathrm{C}_T^{POST} = \mathrm{C}_B^{POST}$.

5.1.3. Proof Details

Proposition 5.5. $\Pr[\textsc{Win}_{\mathrm{TL}}(\mathcal{Q}')] \leq q_* \cdot \alpha/N^*$.

Proof. Let $Q_i = (K_i, X_i, Y_i)$ be the last query.

Case 1: Q_i is a forward query. As in [107], we call a value R *good* if there exists a query $(K', X', \cdot)$ in $\mathcal{Q}'$ that was obtained as a backward query such that $(X', K') = \texttt{SER}(K_i, R)$. By definition, it follows that there are at most α good values R. By construction of $\mathcal{A}'$, a successful query that is used in the bottom row cannot have been obtained as a forward query, since $\mathcal{A}'$ would have immediately launched the corresponding backward query in the top row. In such a case, also by the design of $\mathcal{A}'$, we would have ignored the last query for analysis. Thus, a successful query in the bottom row must have been obtained as a backward query, which leaves us with a chance of success of $\leq \alpha/N^*$ and for q_* queries we have $\leq q_* \cdot \alpha/N^*$.

Case 2: Q_i is a backward query. In order to be successful, there must be a matching query in $\mathrm{BL} = \texttt{SER}(K_i, Y_i) \in \mathcal{Q}_{i-1}$. We now assume that this query is in fact in the query history and denote it by $Q^* = (K_i^*, X_i^*, Y_i^*)$. The post-output of Q^* is uniquely determined and we denote it by V_i^*. Now, there are at most α queries that can be used in BR (*i.e.*, that have an equal post-output as BL). And for any such matching query in BR, there is at most one query in TR (computed via $\texttt{SER}^{-1}$). In total, there are no more than α queries that can possibly be used to find a collision in the top row. The last query has a chance of succeeding of $\leq \alpha/N^*$. In total, for q_* queries, the chance of ever making a successful query of this type can be upper bounded by $q_* \cdot \alpha/N^*$.

□

Proposition 5.6. $\Pr[\textsc{Win}_{\mathrm{BL}}(\mathcal{Q}')] \leq q_* \cdot \alpha / N^*$.

Proof. Apart from swapping the roles of *top row* and *bottom row*, *forward queries* and *backward queries*, and `SER` and $\mathtt{SER}^{-1}$, the proof is the same as for Proposition 5.5 – so we omit it here. □

Proposition 5.7. $\Pr[\textsc{Win}_{TLBL}(\mathcal{Q}')] \leq q_* \cdot \zeta$.

Proof. The analysis is the same for forward queries and backward queries. The probability that the last query can be used concurrently in the top row and bottom row (*i.e.*, TL and BL) can be upper bounded by ζ (cf. Definition 1.16). For a total of q queries, this bound is $q_* \cdot \zeta$. □

Proposition 5.8. $\Pr[\textsc{Win}_{TLBR}(\mathcal{Q}')] \leq q_* / N^*$.

Proof. Assume that the last query is a forward query and we use it in TL and BR (TL = BR). It follows that the query in TR is uniquely determined by the input of the last query. So the chance of success in the top row for q_* queries is upper bounded by q_*/N^*. Now assume that the last query is a backward query used again in TL and BR. Since the query in BL is uniquely determined by the input of the last query used in TL, the total chance of success for q_* queries in the bottom row is again upper bounded by q_*/N^*. In both cases, the success probability is bounded by $\leq q_*/N^*$. □

Proof of Theorem 5.1 The proof follows with Proposition 5.3 and the bounds given in Propositions 5.4, 5.5, 5.6, 5.7, and 5.8. We also let $2q = q_*$ since the new adversary $\mathcal{A}'$ never makes more than twice the queries of the original adversary $\mathcal{A}$. □

5.2. Generic-DL

Theorem 5.9. *Let H^{GEN} be a* GENERIC-DL *compression function as given in Section 1.4.3. Let α, β, n, and γ be constants such that $\alpha > e$ and $N = 2^n$, $N' = N - q$, and $\tau = N'\alpha/q$. Let $\Gamma(q, \gamma)$ be as in equation* (5.3)*. Then*

$$\mathbf{Adv}^{\text{COLL}}_{H^{\text{GEN}}}(q) \leq q\beta(\gamma + 1)/N' + q^2\zeta/N' + q\gamma/N' + L,$$

where

$$L = 2q2^n e^{\tau q(1-\ln\tau)/N'} + 2q^2/(\beta N') + \Gamma(q, \gamma).$$

Proof. The proof of Theorem 5.9 closely follows the proof of Theorem 5.1 generalized by a further parameter γ, but without the 'additional' adversary $\mathcal{A}'$. So the queries of the adversary $\mathcal{A}$ and its query history $\mathcal{Q}$ are analyzed directly. This parameter gives an upper bound on how many queries can be used in the top row for a given bottom-row query. Informally, the function $\Gamma(q, \gamma)$ is defined as the probability that a given parameter γ for a specific value of q is in fact such an upper bound. For sake of convenience, we only discuss the relevant differences in the proof. In particular, we do not restate the basic notions, definitions and the case analysis as given in Section 5.1.2. We note that the bound for GENERIC-DL is considerably worse than for SERIAL-DL or CYCLIC-DL. The main reason for this is that the arguments used in the proofs of Propositions 5.5 and 5.6 do not work here anymore.

5.2.1. Proof Details

First, we give some additional notations. Let γ be an upper bound for the number of queries that can be used in the top row for any given query in the bottom row. Let $\texttt{MaxTopCount}(\mathcal{Q})$ be a function defined on query sequences $\mathcal{Q}$ of length q as follows:

$$\texttt{MaxTopCount}(\mathcal{Q}) = \max_{1 \le j \le q} |\{i : (\mathrm{C}_B^{PRE} \circ \mathrm{C}^{LNK})(\mathrm{C}_T^{-PRE}(K_i, X_i), Y_i) = (K_j, X_j)\}|.$$

Using the notions of Section 5.1.2 we state the event $\textsc{Lucky}^{\textsc{Gen}}(\mathcal{Q})$ as

$$\textsc{Lucky}^{\textsc{Gen}}(\mathcal{Q}) = \textsc{Lucky}(\mathcal{Q}) \ \vee \ \texttt{MaxTopCount}(\mathcal{Q}) > \gamma.$$

The function $\Gamma(q, \gamma)$ with $|\mathcal{Q}| = q$ is defined as

$$\Gamma(q, \gamma) = \Pr[\texttt{MaxTopCount}(\mathcal{Q}) > \gamma]. \tag{5.3}$$

We define the four predicates $\textsc{Win}_{\mathrm{TL}}(\mathcal{Q})$, $\textsc{Win}_{\mathrm{BL}}(\mathcal{Q})$, $\textsc{Win}_{TLBL}(\mathcal{Q})$ and $\textsc{Win}_{TLBR}(\mathcal{Q})$ as on page 78, only $\textsc{Lucky}(\mathcal{Q}')$ is substituted by $\textsc{Lucky}^{\textsc{Gen}}(\mathcal{Q})$. Now, the proof is essentially the same as for Theorem 5.1, it only differs in the arguments and some subtle modifications in Propositions 5.5, 5.6, 5.7, and 5.8.

Argument MTC. We label a row (either top row or bottom row) as *query row* (QR) if this is the row the last query shall be used in. Similarly, the term *other row* (OR) is used in the discussion if the last query is *not* used in this row. Let γ' be a parameter giving the maximum number of queries that can be used in the *query row* given a query in the *other row*. We assume *wlog.* that the last query is used in position QL. We upper bound the number of queries possible for QR as follows. There are at most β pairs of queries that can be used in the other row to form a collision (*i.e.*, the post-outputs of OL and OR are equal). For any query in OR, there are at most γ' queries that can be used in QR. So the success probability can be upper bounded by $\beta\gamma'/N'$. For q queries, this bound is $q\beta\gamma'/N'$.

Argument MTC is now used to derive easily a collision resistance bound in Propositions 5.10 and 5.11.

Proposition 5.10. $\Pr[\textsc{Win}_{\mathrm{TL}}(\mathcal{Q})] \leq q\beta\gamma/N'$.

Proof. The *query row* QR is the top row and the *other row* OR is the bottom row. The analysis is the same for a forward query and backward query since in both cases Argument MTC with parameter $\gamma' = \gamma$ holds. □

Proposition 5.11. $\Pr[\textsc{Win}_{\mathrm{BL}}(\mathcal{Q})] \leq q\beta/N'$.

Proof. The *query row* QR is the bottom row and the *other row* OR is the top row. The analysis is the same for a forward row and backward query since in both cases Argument MTC with parameter $\gamma' = 1$ holds. □

Proposition 5.12. $\Pr[\textsc{Win}_{TLBL}(\mathcal{Q})] \leq q^2 \cdot \zeta/N'$.

Proof. The proof is the same as for Proposition 5.7. We omit it here. □

Proposition 5.13. $\Pr[\textsc{Win}_{TLBR}(\mathcal{Q})] \leq q\gamma/N'$.

Proof. The following argument is the same for a forward query and a backward query. Assume that the last query is used in TL and BR. Then there are at most γ queries that can be used in TR. So the probability of a collision of the post-outputs in the top row is upper bounded (for q queries) by $q\gamma/N'$ and our claim follows. □

Proof of Theorem 5.9 The proof follows from our discussion together with Proposition 5.3 and the bounds given in Propositions 5.4, 5.10, 5.11, 5.12, and 5.13. □

5.3. Applications

5.3.1. Collision Resistance of Tandem-DM

We now apply Theorem 5.1 to Tandem-DM (cf. also Figure 5.2(a)). We again let $N = 2^n$ and $N'' = (N - 2q)$. We have to derive an upper bound for the value ζ, *i.e.*, the probability that a query can be used at the same time in the top row and bottom row. It follows that this is the case iff $U = \widehat{U}$, $M = \widehat{U}$ and $E_{\widehat{U}\|M}(U) = \widehat{U}$ which can be upper bounded by $1/(N-2q)$, *i.e.*, $\zeta = 1/(N-2q)$. So we evaluate

$$\begin{aligned}\mathbf{Adv}^{\text{Coll}}_{H^{\text{TDM}}}(q) \leq & 2q(2\alpha+1)/N'' + 2q/(N-2q) \\ & + 4 \cdot \left(qNe^{\tau 2q(1-\ln\tau)/N''} + q^2/(\beta N'')\right).\end{aligned}$$

Using a numerical optimization process, we find for $n = 128$ that no adversary asking less than $q = 2^{119.6}$ queries can find a collision with probability greater than 1/2. The parameters in this case are $\alpha = 24$ and $\beta = q/4$. More details are given in Table 5.1.

5.3.2. Collision Resistance of Mix-Tandem-DM

We now apply the generic construction Generic-DL in order to get a collision security bound for a construction that we call Mix-Tandem-DM depicted in Figure 5.2 (b) and given in Definition 5.14.

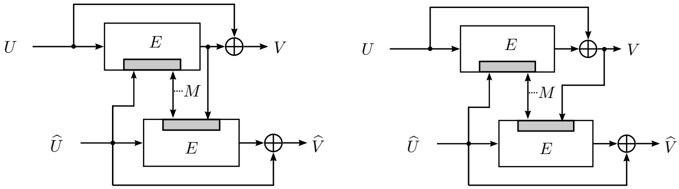
Definition 5.14. *Let* $H^{\text{MTDM}} : \{0,1\}^{2n} \times \{0,1\}^n \to \{0,1\}^{2n}$ *be a compression function such that* $(V, \widehat{V}) = H^{\text{MTDM}}(M, U, \widehat{U})$ *where* $U, \widehat{U}, V, \widehat{V}, M \in \{0,1\}^n$. H^{MTDM} *consists of a blockcipher* $E \in$

q	$\mathbf{Adv}^{\text{COLL}}_{H^{\text{TDM}}}(q) \leq$	α	β
$2^{111.16}$	10^{-4}	14	$q/4$
$2^{114.6}$	$1/100$	19	$q/4$
$2^{117.8}$	$1/10$	23	$q/4$
$2^{119.2}$	$1/4$	24	$q/4$
$2^{119.6}$	$1/2$	24	$q/4$

Table 5.1.: Upper bounds on $\mathbf{Adv}^{\text{COLL}}_{H^{\text{TDM}}}(q)$ as given by Theorem 5.1 evaluated for $n = 128$; we always let $\beta = q/4$

BLOCK$(2n, n)$ *as follows:*

$$V = E_{\widehat{U}\|M}(U) \oplus U,$$
$$\widehat{V} = E_{M\|V}(\widehat{U}) \oplus \widehat{U}.$$



(a) TANDEM-DM (b) MIX-TANDEM-DM

Figure 5.2.: Comparison of TANDEM-DM and MIX-TANDEM-DM

We show that in this case $\zeta = 1/N'$ (Step 1), $\Gamma(q, \gamma) = 0$ (Step 2), and $\gamma = \alpha$ (Step 3). Using again a numerical optimization process,

q	$\mathbf{Adv}_{H^{\text{MTDM}}}^{\text{Coll}}(q) \leq$	α	β
$2^{77.4}$	10^{-4}	3	2^{38}
$2^{79.6}$	1/100	3	2^{39}
$2^{81.8}$	1/10	3	2^{40}
$2^{82.6}$	1/4	3	2^{40}
$2^{83.3}$	1/2	3	2^{41}

Table 5.2.: Upper bounds on $\mathbf{Adv}_{H^{\text{MTDM}}}^{\text{Coll}}(q)$ as given by Theorem 5.9 evaluated for $n = 128$

it is easy to compute that for $n = 128$ no adversary asking less than $q = 2^{83.3}$ queries can find a collision with probability greater than 1/2. In this case, the parameters are $\alpha = 3$ and $\beta = 2^{41}$. More details can be found in Table 5.2.

(Step 1) Determining ζ. First, we upper bound the probability that a query can be used in the top row and bottom row simultaneously. This is the case iff $U = \widehat{U}$, $M = \widehat{U}$, and $E_{\widehat{U}\|M}(U) \oplus U = \widehat{U}$. Independent of the query type – either forward or backward – this is upper bounded by $1/N'$.

(Step 2) Deriving a bound for $\Pr[\texttt{MaxTopCount}(\mathcal{Q}) > \gamma]$**.** We have to upper bound the probability that for a given bottom-row query, the number of matching top-row queries that can be used in a compression function exceeds a specific value, here γ. For Mix-Tandem-DM, this is simply the number of queries that all share the same XOR-output. Since we already have an upper bound for this, we can set $\gamma = \alpha$ and 'reuse' the bound we already got in the analysis of $\text{Lucky}^{\text{Gen}}(\mathcal{Q})$.

(Step 3) Success probability of an adversary. Following closely Theorem 5.9, we can state the bound:

$$\mathbf{Adv}_{H^{\mathrm{MTDM}}}^{\mathrm{COLL}}(q) \leq q\beta(\alpha+1)/N' + q^2/(N')^2 + q\alpha/N' + L,$$

where

$$L = qNe^{\tau q(1-\ln\tau)/N'} + q^2/(\beta N').$$

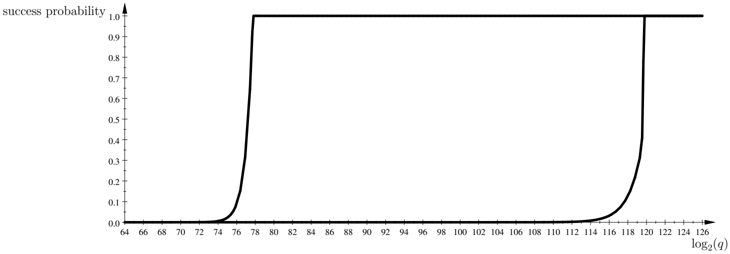


Figure 5.3.: Collision Security Bound for TANDEM-DM (right) and MIX-TANDEM-DM (left).

5.4. Combinatorial Proofs

We now give a proof for Proposition 5.4. The proof essentially follows by adding up the individual results of Lemmas 5.15 and 5.16. The analysis is only given for the top-row bounds, *i.e.*, for $\mathtt{NumEqual}_{\mathrm{T}}(\mathcal{Q}')$, $\mathtt{NumColl}_{\mathrm{T}}(\mathcal{Q}')$, and $\mathtt{NumTriangleColl}_{\mathrm{T}}(\mathcal{Q}')$. The bottom-row equivalents $\mathtt{NumEqual}_{\mathrm{B}}(\mathcal{Q}')$, $\mathtt{NumColl}_{\mathrm{B}}(\mathcal{Q}')$, and $\mathtt{NumTriangleColl}_{\mathrm{B}}(\mathcal{Q}')$ can be omitted for the bound $\Pr[\textsc{Lucky}(\mathcal{Q}')]$ iff $\mathrm{C}_T^{POST} = \mathrm{C}_B^{POST}$. In Proposition 5.4, we have stated the more generic bound that also permits different post-processing functions in the top row and bottom row, *i.e.*,

$$\begin{aligned}\Pr[\textsc{Lucky}(\mathcal{Q}')] \leq & 2 \cdot (\Pr[\mathtt{NumEqual}_{\mathrm{T}}(\mathcal{Q}') > \alpha] \\ & + \Pr[\mathtt{NumColl}_{\mathrm{T}}(\mathcal{Q}') > \beta]).\end{aligned}$$

Note that the constants in the following Lemmas are the same as before, including $|\mathcal{Q}'| = q_*$ and $N^* = 2^n - q_*$.

Lemma 5.15. $\Pr[\texttt{NumEqual}_{\mathrm{T}}(\mathcal{Q}') > \alpha] \leq q_* 2^n e^{\tau q_*(1-\ln\tau)/N^*}$.

Lemma 5.16. $\Pr[\texttt{NumColl}_{\mathrm{T}}(\mathcal{Q}') > \beta] \leq q_*^2/(\beta N^*)$.

The proofs are essentially due to Steinberger [161].

Proof of Lemma 5.15.

We now upper bound the number of queries in the query history $\mathcal{Q}'$ that have equal post-outputs. For a forward query with key K and plaintext X, the post-output V is computed as

1. $(M, U, \widehat{U}) = \mathrm{C}^{-PRE}(K, X)$,
2. $Y = E_K(X)$,
3. $V = \mathrm{C}^{POST}(M, U, \widehat{U}, Y)$.

For a backward query using key K and ciphertext Y, the post-output V is computed as

1. $X = E_K^{-1}(Y)$,
2. $V = \mathrm{C}^{AUX}(K, X, Y)$.

Since $\mathrm{C}^{POST}(M, U, \widehat{U}, \cdot)$ and $\mathrm{C}^{AUX}(K, \cdot, Y)$ are both bijective and since for a forward query Y (for a backward query X) is chosen independently and evenly distributed, the post-outputs V are also chosen independently and evenly distributed (in both cases).

So the queries $Q_i = (K_i, X_i, Y_i)$, with post-output V_i, and $Q_j = (K_j, X_j, Y_j)$, with post-output V_j, have equal post-outputs iff $V_i = V_j$. Since V_i and V_j are chosen independently from a set of at least $2^n - q_*$ values, we can rephrase this question as a balls-in-bins question. Let $N = 2^n$ be the number of bins and q_* be the number of balls to be thrown. The i-th ball falls into the j-th bin, if the post-output of the i-th query is equal to the post output of the j-th query, *i.e.*, $V_i = V_j$. In the following we upper bound the probability that some bin contains more than α balls. As the balls are thrown independent of each other, the i-th ball always has probability $\leq p = 1/N^*$ of falling in the j-th bin. Let $B(k)$ be the probability of having exactly k balls in a particular bin, say bin 1. Then we can upper bound

$$B(k) \leq p^k \binom{q_*}{k}.$$

Let $\nu = q_* p$, where ν is an upper bound for the expected number of balls in any bin. By Stirlings approximation [41] (and e^x being the exponential function)

$$n! \leq \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \cdot e^{1/(12n)},$$

we can upper bound $B(k)$ as follows:

$$\begin{aligned}
B(k) &\leq p^k \frac{q_*!}{k!(q-k)!} \\
&\leq \frac{p^k}{\sqrt{2\pi}} \sqrt{\frac{q_*}{k(q-k)}} \cdot \frac{q_*^q}{k^k (q-k)^{1-k}} \cdot \frac{e^k \cdot e^{q_* - k}}{e^q} \cdot e^{\frac{1}{12}(q-k-(q-k))} \\
&\leq k^{-k} \nu^k \left(\frac{q_*}{q_* - k}\right)^{q_* - k} \\
&\leq \nu^k \cdot k^{-k} \cdot e^k.
\end{aligned}$$

And, since $\alpha = \tau\nu$, we get

$$B(\alpha) \leq \frac{\nu^{\tau\nu} e^{\tau\nu}}{(\tau\nu)^{\tau\nu}} = \frac{e^{\tau\nu}}{\tau^{\tau\nu}} = e^{\tau\nu(1-\ln\tau)}.$$

It follows that $B(\alpha)$ is a decreasing function of α if $\tau > e$. Then we can upper bound

$$\begin{aligned}\Pr[\texttt{NumEqual}_{\mathrm{T}}(\mathcal{Q}') > \alpha] &\leq 2^n \cdot \sum_{j=\alpha}^{q} B(j) \\ &\leq q_* 2^n B(\alpha) \quad \leq \quad q_* 2^n e^{\tau\nu(1-\ln\tau)}.\end{aligned}$$

This proves our claim. □

Proof of Lemma 5.16.

Let $C_{i,j}$ be the event that the queries i and j, $i \neq j$, have the same post-output, *i.e.*, $\Pr[C_{i,j}] \leq 1/N^*$ (cf. the discussion in den previous section). As a result, $E[\texttt{NumColl}_{\mathrm{T}}(\mathcal{Q}')] = \sum_{i \neq j} E[C_{i,j}] \leq q_*^2/N^*$. Using Markov's inequality, we get

$$\Pr[\texttt{NumColl}_{\mathrm{T}}(\mathcal{Q}') > \beta] \leq \frac{q_*^2}{\beta N^*},$$

□

since Q_i uniquely determines the query Q_j.

6 MDC-4

6.1. MDC-4 Hash Function

6.1.1. The compression function of MDC-4

We recall the definition of the MDC-4 compression function given on Page 25 (cf. Figure 6.1). Assuming again two n-bit chaining values U and $\widehat{U}$, a blockcipher $E \in \textsc{Block}(n,n)$, and an n-bit message block M. The compression function of MDC-4 (cf. Figure 6.1) works as follows:

1. $S = (S^L \| S^R) \leftarrow E_U(M) \oplus M$
2. $T = (T^L \| T^R) \leftarrow E_{\widehat{U}}(M) \oplus M$
3. $V \leftarrow E_{S^L \| T^R}(\widehat{U}) \oplus \widehat{U}$
4. $\widehat{V} \leftarrow E_{T^L \| S^R}(U) \oplus U$

The superscript L denotes the left $n/2$ bits of an expression, and the superscript R denotes the right $n/2$ bits of an expression. The original MDC-4 specification [122] swaps the right halves of V and $\widehat{V}$. But, since we are in the ideal cipher model, this operation does neither change the distribution of the output nor our collision security analysis. So, for ease of presentation, we omit this additional step. Apart from notational complication of matters, the right/left swap would also not change any of our arguments of the following proof(s).

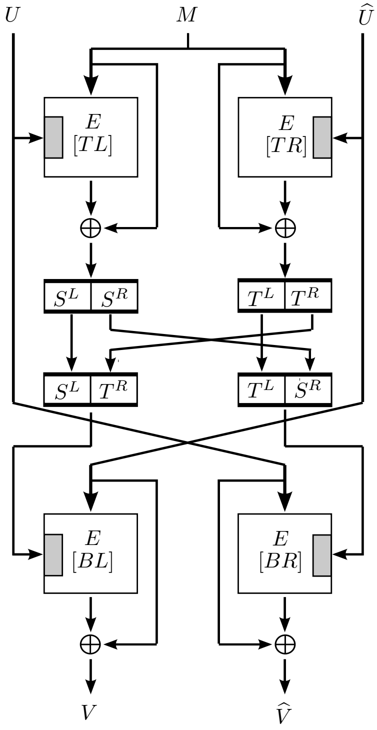


Figure 6.1.: The double-length quad-call compression function $H^{\text{MDC-4}}$ where $E \in \text{Block}(n, n)$; the notch indicates the key input of the cipher.

Our analysis is for the MDC-4 hash function $\mathcal{H}^{\text{MDC-4}}$ which is obtained by a simple iteration of the MDC-4 compression function $H^{\text{MDC-4}}$ in the Merkle-Damgård manner: Given some $n \cdot \ell$-bit message $\mathcal{M} = (M_1, \ldots, M_\ell)$, $M_j \in \{0,1\}^n$ for $j = 1, \ldots, \ell$ and an initial value $(U_0, \widehat{U}_0) \in \{0,1\}^{2n}$ it works by computing

$$(U_i, \widehat{U}_i) = H^{\text{MDC-4}}(M_i, U_{i-1}, \widehat{U}_{i-1})$$

for $i = 1, \ldots, \ell$. The hash value $\mathcal{H}^{\text{MDC-4}}(\mathcal{M})$ is $(U_\ell, \widehat{U}_\ell)$.

Collision Security of the MDC-4 compression function. There is a very simple attack on the compression function which only requires about $2^{n/2}$ invocations of the E oracle: Let the adversary find values $K, K', M, M' \in \{0,1\}^n$ such that $E_K(M) = E_{K'}(M')$. Then, by

$$H^{\text{MDC-4}}(M, K, K) = H^{\text{MDC-4}}(M', K', K'),$$

a collision for the MDC-4 compression function has been found.

Another interesting observation is that a collision for the MDC-2 compression function might also lead to a collision for the MDC-4 compression function as follows: Assume that an adversary has found values M, M', K, K' such that

$$H^{\text{MDC-2}}(M, K, K') = H^{\text{MDC-2}}(M', K, K')$$

then this automatically leads to a collision for the MDC-4 compression function since

$$H^{\text{MDC-4}}(M, K, K') = H^{\text{MDC-4}}(M', K, K').$$

Though, the relevance of this observation is quite limited, since such a collision for MDC-2 is much harder to find than our final bound will state (which is for the full MDC-4 hash function and not only for the MDC-4 compression function).

6.2. Collision Security

6.2.1. Proof Model

Our analysis is for the MDC-4 hash function $\mathcal{H}^{\text{MDC-4}}$ assuming that the initial chaining values are different, *i.e.*, $U_0 \neq \widehat{U}_0$. The goal of the adversary is to output two messages $\mathcal{M}_1 \in \{0,1\}^{n\cdot\ell}$ and $\mathcal{M}_2 \in \{0,1\}^{n\cdot\ell'}$ such that $\mathcal{H}(\mathcal{M}_1) = \mathcal{H}(\mathcal{M}_2)$ for some positive integers ℓ and ℓ'.

In our analysis, we dispense the adversary from returning these two messages. Instead, we upper bound its success probability by giving the attack to it if

(i) an 'internal' collision has been found, *i.e.*, $(M, U, \widehat{U})$ such that

$$(V, V) = H^{\text{MDC-4}}(M, U, \widehat{U})$$

for some $V \in \{0,1\}^n$ or

(ii) case (i) is not `true` but a collision in the compression function $H^{\text{MDC-4}}$ has been found, *i.e.*, $(M, U, \widehat{U})$ and $(M', U', \widehat{U}')$, such that

$$H^{\text{MDC-4}}(M, U, \widehat{U}) = H^{\text{MDC-4}}(M', U', \widehat{U}'),$$

and $U \neq \widehat{U}, U' \neq \widehat{U}'$ or

(iii) cases (i), (ii) are not `true` but values $(M, U, \widehat{U})$ have been found such that

$$(U_0, \widehat{U}_0) = H^{\text{MDC-4}}(M, U, \widehat{U}).$$

Note that this requirement essentially models the preimage resistance of the MDC-4 *compression function.*

The proof is well-known and straightforward: Assume a collision for $\mathcal{H}^{\text{MDC-4}}$ has been found using two not necessarily equal-length messages $\mathcal{M}$ and $\mathcal{M}'$, *i.e.*, $\mathcal{H}^{\text{MDC-4}}(\mathcal{M}) = \mathcal{H}^{\text{MDC-4}}(\mathcal{M}')$. Also assume that the collision is the earliest possible, *i.e.*, that neither message cannot be shortened without loosing the collision. Then, the adversary has either found (i) or (ii) or has 'hit' the initial value while hashing the second to the last block. The latter is covered by (iii).

For our analysis, we again impose the reasonable condition that the adversary must have made all queries necessary to compute the results. We determine whether the adversary has been successful or not by examining the query history $\mathcal{Q}$. Formally, we say that $\textsc{Coll}^{\text{MDC-4}}(\mathcal{Q})$ holds if there is such a collision and $\mathcal{Q}$ contains all the queries necessary to compute it.

We now define what we formally mean by a collision of the MDC-4 compression function.

Definition 6.1 (Collision resistance of the MDC-4 CF). *Let $H^{\text{MDC-4}}$ be a* MDC-4 *compression function. Fix an adversary $\mathcal{A}$. Then the advantage of $\mathcal{A}$ in finding collisions for $H^{\text{MDC-4}}$ is the real number*

$$\begin{aligned}\mathbf{Adv}^{\textsc{Coll}}_{H^{\text{MDC-4}}}(\mathcal{A}) = \Pr[E &\overset{\$}{\leftarrow} \textsc{Block}(n,n);\\ &((M,U,\widehat{U}),(M',U',\widehat{U}')) \overset{\$}{\leftarrow} \mathcal{A}^{E,E^{-1}} :\\ &((M,U,\widehat{U}) \neq (M',U',\widehat{U}'))\\ &\wedge H^{\text{MDC-4}}(M,U,\widehat{U}) = H^{\text{MDC-4}}(M',U',\widehat{U}')\\ &\wedge U \neq \widehat{U} \ \wedge \ U' \neq \widehat{U}'].\end{aligned}$$

For $q \geq 1$ we write

$$\mathbf{Adv}^{\textsc{Coll}}_{H^{\text{MDC-4}}}(q) = \max_{\mathcal{A}}\{\mathbf{Adv}^{\textsc{Coll}}_{H^{\text{MDC-4}}}(\mathcal{A})\},$$

where the maximum is taken over all adversaries that ask at most q oracle queries, *i.e.*, forward and backward queries to E.

Since our analysis in the next sections is for $\mathcal{H}^{\text{MDC-4}}$, we informally say that the probability of a collision for $\mathcal{H}^{\text{MDC-4}}$ is upper bounded by using a union bound for the cases (i), (ii) and (iii). This is part of the formalization in Theorem 6.2.

6.2.2. Our Results

We now give our main result. Although, having a substantial complexity on the first sight in its general form, we can easily evaluate it to numerical terms (cf. Corollary 6.3).

Theorem 6.2. *Fix some initial values $S_0, T_0 \in \{0,1\}^n$ with $S_0 \neq T_0$ and let $\mathcal{H}^{\text{MDC-4}}$ be the* MDC-4 *hash function as given in Section 6.1.1. Let $N = 2^n$ and α, β, γ be constants such that $eq2^{n/2}/(N - q) \leq \alpha$, $eq/(N-q) \leq \beta$, and let* $\Pr[\textsc{Lucky}(\mathcal{Q})]$ *as in Proposition 6.15. Then,*

$$\begin{aligned}\mathbf{Adv}^{\textsc{Coll}}_{\mathcal{H}^{\text{MDC-4}}}(q) \leq & q\left(\frac{\alpha^2+\gamma}{N-q} + \frac{2\alpha}{N-\alpha 2^{n/2+1}+\alpha^2}\right) \\ & + q\left(\frac{\beta^2+4}{N-q} + \frac{\beta}{N-q}\right) \\ & + 2q\left(\frac{\alpha^4+\alpha^2+3\alpha\gamma+2\gamma}{N-q} + 6\frac{\alpha^2+1}{N^{1/2}-\alpha}\right) \\ & + q\left(\frac{\gamma\alpha^2+\gamma^2}{N-q} + \frac{2\alpha}{(2^{n/2}-\alpha)^2}\right) + \Pr[\textsc{Lucky}(\mathcal{Q})].\end{aligned} \tag{6.1}$$

The proof of Theorem 6.2 is developed throughout the following discussion and explicitly stated in Section 6.2.5. As mentioned before, our bound is rather non-transparent, so we discuss it for $n = 128$. We evaluate the equation above such that the adversary's advantage is upper bounded by 1/2 – thereby maximizing the value of q by numerically optimizing the values of α, β, and γ. Our result is the following corollary.

q	$\mathbf{Adv}^{\textsc{Coll}}_{\mathcal{H}^{\text{MDC-4}}}(q) \leq$	α	β	γ
2^{64}	$7.18 \cdot 10^{-7}$	42	4.0	$2 \cdot 10^{6}$
$2^{68.26}$	10^{-4}	126	4.0	$6 \cdot 10^{6}$
$2^{72.19}$	1/100	900	4.0	$1.3 \cdot 10^{7}$
$2^{73.84}$	1/10	2600	4.0	$1.4 \cdot 10^{7}$
$2^{74.40}$	1/4	3780	4.0	$1.5 \cdot 10^{7}$
$2^{74.76}$	1/2	4900	4.0	$1.5 \cdot 10^{7}$

Table 6.1.: Upper bounds on $\mathbf{Adv}^{\textsc{Coll}}_{\mathcal{H}^{\text{MDC-4}}}(q)$ as given by Theorem 6.2

Corollary 6.3. *No adversary asking less than* $2^{74.76}$ *queries can find a collision for the* MDC-4 *hash function with probability greater than* 1/2.

An overview of the behavior of our upper bound is given in Table 6.1 and is depicted in Figure 6.2. Note that for other values of (α, β, γ) the bound stays *correct* but worsens numerically (as long as the conditions given in Theorem 6.2 hold).

6.2.3. Proof Preliminaries

Overview. Our discussion starts with case (ii). We analyze whether the list of oracle queries to E made by the adversary can be used for a collision of the MDC-4 compression function $H^{\text{MDC-4}}$. For a collision, there are eight – not necessarily distinct – blockcipher queries necessary (cf. Figure 6.3).

To upper bound the probability of the adversary obtaining queries that can be used for a collision, we upper bound the probability

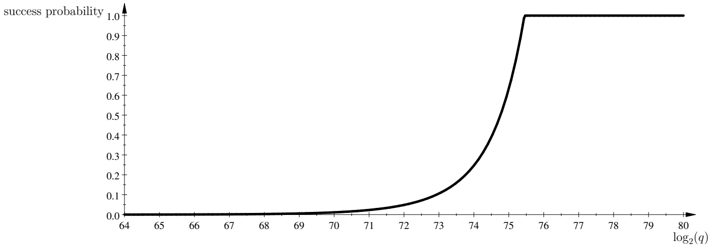


Figure 6.2.: Collision Security Bound for MDC-4; evaluated for $n = 128$.

of the adversary making a final query that can be used as the last query to complete such a collision. Let $\mathcal{Q}_i$ denote the set of the first i queries $(K_1, X_1, Y_1), ..., (K_i, X_i, Y_i)$ (either forward or backward) made by the adversary. Furthermore, we denote by the term *last query* the latest query made by the adversary. This query has always index i. Therefore, for each i with $1 \leq i \leq q$, we upper bound the success probability of an adversary to use the i-th query to complete a collision.

As the probability depends on the first $i-1$ queries, we have to put some restrictions on these and also upper bound the probability that these restrictions are not met by an adversary. One example of such a restriction is to assume that, *e.g.*, the adversary has not found too many *collisions* for the underlying component function $E_K(X) \oplus X$.

Thus, our upper bound breaks down into two parts: first, an upper bound for the probability of an adversary not meeting our restrictions and, second, the probability of an adversary ever making a successful i-th query, conditioned on the fact that the adversary does meet our restrictions and has not been successful by its $(i-1)$-th query. We use some notations that are given in Figure 6.3, *e.g.*, the statement 1BL $\neq$ 2BL means that the query used in the bottom left of the 'left' side is not the same as the query used in the bottom left of the 'right' side.

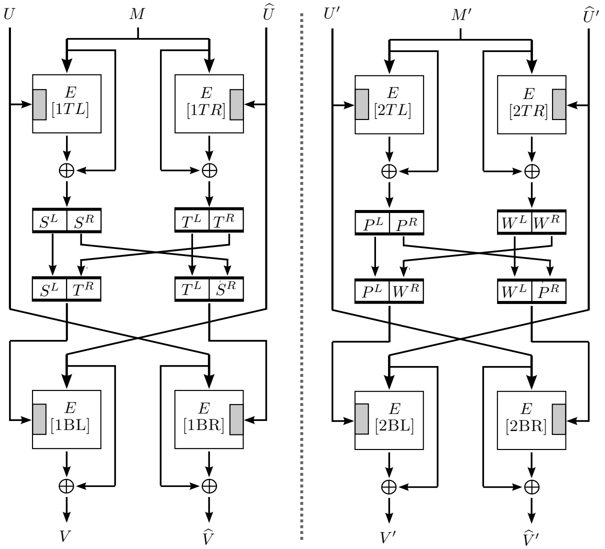


Figure 6.3.: The double-length MDC-4 compression function $H^{\text{MDC-4}}$, where $E \in \textsc{Block}(n,n)$. If $(M,U,\widehat{U}) \neq (M',U',\widehat{U}')$ but $(V,\widehat{V}) = (V',\widehat{V}')$ then the adversary has found a collision for $H^{\text{MDC-4}}$. The notch inside the cipher indicates the key input. For later reference, the different positions a query can be used in are denoted by $1TL, 1TR, \ldots, 2\text{BR}$.

6.2.4. Details

We say $\textsc{Coll}^{\text{MDC-4}}(\mathcal{Q})$ if the adversary *wins*. We note that *winning* does not necessarily imply that the adversary has found a collision. It might also be that the adversary got lucky and does not meet our restrictions any more. But in the case of a collision $\textsc{Coll}^{\text{MDC-4}}(\mathcal{Q})$ always holds.

Proposition 6.4.

$$\begin{gathered}\textsc{Coll}^{\text{MDC-4}}(\mathcal{Q}) \implies \\ \textsc{Lucky}(\mathcal{Q}) \vee \textsc{InternalColl}(\mathcal{Q}) \vee \textsc{CollTopRows}(\mathcal{Q}) \vee \\ \textsc{CollLeftColumns}(\mathcal{Q}) \vee \textsc{CollRightColumns}(\mathcal{Q}) \vee \\ \textsc{CollBothColumns}(\mathcal{Q}) \vee \textsc{Preimage}(\mathcal{Q}).\end{gathered}$$

We now define the involved predicates of Proposition 6.4 and then give a proof. The predicates on the 'right' side of the expression are made mutually exclusive, meaning that if the left side is true it follows that exactly one of the predicates on the right side is true. By upper bounding separately the probabilities of these predicates on the right side it is easy to see that the union bound can be used to upper bound the probability of $\textsc{Coll}^{\text{MDC-4}}(\mathcal{Q})$ as follows:

$$\begin{aligned}\Pr[\textsc{Coll}^{\text{MDC-4}}(\mathcal{Q})] \leq & \Pr[\textsc{Lucky}(\mathcal{Q})] + \Pr[\textsc{InternalColl}(\mathcal{Q})] \\ & + \Pr[\textsc{CollTopRows}(\mathcal{Q})] \\ & + \Pr[\textsc{CollLeftColumns}(\mathcal{Q})] \\ & + \Pr[\textsc{CollRightColumns}(\mathcal{Q})] \\ & + \Pr[\textsc{CollBothColumns}(\mathcal{Q})] \\ & + \Pr[\textsc{Preimage}(\mathcal{Q})].\end{aligned}$$

To state the predicate $\textsc{Lucky}(\mathcal{Q})$, we give some helper definitions that are also used as restrictions for the other predicates. We now let $\texttt{NumEqual}(\mathcal{Q})$ be a function defined on the query set $\mathcal{Q}$, $|\mathcal{Q}| = q$, as follows:

$$\texttt{NumEqual}(\mathcal{Q}) = \max_{Z \in \{0,1\}^n} |\{i : E_{K_i}(X_i) \oplus X_i = Z\}|.$$

It is the maximum size of a set of queries in $\mathcal{Q}$ whose XOR-outputs are all the same. Similarly, we define $\texttt{NumEqualHalf}(\mathcal{Q})$ as the maximum size of a set of queries whose XOR-outputs either share the same left half or the same right half. Let

$$\texttt{NEH-L}(\mathcal{Q}) = \max_{Z \in \{0,1\}^{n/2}} |\{i : (E_{K_i}(X_i) \oplus X_i)^L = Z\}|,$$
$$\texttt{NEH-R}(\mathcal{Q}) = \max_{Z \in \{0,1\}^{n/2}} |\{i : (E_{K_i}(X_i) \oplus X_i)^R = Z\}|,$$

then $\texttt{NumEqualHalf}(\mathcal{Q}) = \max(\texttt{NEH-L}(\mathcal{Q}), \texttt{NEH-R}(\mathcal{Q}))$. We now let $\texttt{NumColl}(\mathcal{Q})$ be also defined on a query set $\mathcal{Q}$, $|\mathcal{Q}| = q$, as

$$\begin{aligned}\texttt{NumColl}(\mathcal{Q}) = |\{(i,j) \in \{1,\ldots,q\}^2 : i \neq j,\\ E_{K_i}(X_i) \oplus X_i = E_{K_j}(X_j) \oplus X_j\}|.\end{aligned}$$

It outputs the number of ordered pairs of distinct queries in $\mathcal{Q}$ which have the same XOR-outputs.
We now define the event $\textsc{Lucky}(\mathcal{Q})$ as

$$\begin{aligned}\textsc{Lucky}(\mathcal{Q}) =&(\texttt{NumEqualHalf}(\mathcal{Q}) > \alpha) \ \vee \ (\texttt{NumEqual}(\mathcal{Q}) > \beta) \\ &\vee \ (\texttt{NumColl}(\mathcal{Q}) > \gamma),\end{aligned} \tag{6.2}$$

where α, β, and γ are the constants from Theorem 6.2. These constants, depending on n and q, are chosen by a simple numerical optimization process such that the upper bound of the advantage of an adversary is minimized for given values of n and q. We now give the definitions of the other predicates.

FitInternalColl($\mathcal{Q}$). The adversary has found four – not necessarily distinct – queries such that $H^{\text{MDC-4}}(M, U, \widehat{U})$ can be computed and $H^{\text{MDC-4}}(M, U, \widehat{U}) = (V, V)$ holds for some arbitrary V with $U \neq \widehat{U}$.

FitCollLeftColumns($\mathcal{Q}$). The adversary has found eight – not necessarily distinct – queries such that $(V, \widehat{V}) = H^{\text{MDC-4}}(M, U, \widehat{U})$ and $(V', \widehat{V}') = H^{\text{MDC-4}}(M', U', \widehat{U}')$ can be computed with $V = V'$, 1BL $\neq$ 2BL and 1BR = 2BR.

FitCollRightColumns($\mathcal{Q}$)**.** The adversary has found eight – not necessarily distinct – queries such that $(V, \widehat{V}) = H^{\text{MDC-4}}(M, U, \widehat{U})$ and $(V', \widehat{V}') = H^{\text{MDC-4}}(M', U', \widehat{U}')$ can be computed with $\widehat{V} = \widehat{V}'$, 1BR $\neq$ 2BR and 1BL = 2BL.

FitCollTopRows($\mathcal{Q}$)**.** The adversary has found four – not necessarily distinct – queries such that

$$(E_U(M) \oplus M, E_{\widehat{U}}(M) \oplus M) = (E_{U'}(M') \oplus M', E_{\widehat{U}'}(M') \oplus M')$$

for $U \neq \widehat{U}$, $U' \neq \widehat{U}'$, 1BL = 2BL, and 1BR = 2BR. Clearly, in this case, $U = U'$ and $\widehat{U} = \widehat{U}'$ and therefore $M \neq M'$.

FitCollBothColumns($\mathcal{Q}$)**.** In this case we assume that both $\neg$FitCollLeftColumns($\mathcal{Q}$), $\neg$FitCollRightColumns($\mathcal{Q}$) are 'true'. The adversary has found eight – not necessarily distinct – queries such that

$$(V, \widehat{V}) = H^{\text{MDC-4}}(M, U, \widehat{U}) \text{ and } (V', \widehat{V}') = H^{\text{MDC-4}}(M', U', \widehat{U}')$$

can be computed with $V = V'$, $\widehat{V} = \widehat{V}'$, 1BL $\neq$ 2BL, and 1BR $\neq$ 2BR.

FitPreimage($\mathcal{Q}$)**.** This formalizes case (iii). The adversary has found four – not necessarily distinct – queries used in $H^{\text{MDC-4}}$ in positions 1TL, 1TR, 1BL, 1BR such that the output of $H^{\text{MDC-4}}$ is equal to $(U_0, \widehat{U_0})$, *i.e.*, the initial chaining values of the MDC-4 hash function.

For practical purposes we derive our predicates as follows.

$$\begin{aligned}
\text{InternalColl}(\mathcal{Q}) &= \neg\text{Lucky}(\mathcal{Q}) \wedge \text{FitInternalColl}(\mathcal{Q}),\\
\text{CollLeftColumns}(\mathcal{Q}) &= \neg(\text{Lucky}(\mathcal{Q}) \vee \text{FitInternalColl}(\mathcal{Q}))\\
&\quad \wedge \text{FitCollLeftColumns}(\mathcal{Q}),
\end{aligned}$$

$$\begin{aligned}
\textsc{CollRightColumns}(\mathcal{Q}) &= \neg(\textsc{Lucky}(\mathcal{Q}) \vee \textsc{FitInternalColl}(\mathcal{Q}) \\
&\quad \vee \textsc{FitCollLeftColumns}(\mathcal{Q})) \\
&\quad \wedge \textsc{FitCollRightColumns}(\mathcal{Q}), \\
\textsc{CollTopRows}(\mathcal{Q}) &= \neg(\textsc{Lucky}(\mathcal{Q}) \vee \textsc{FitInternalColl}(\mathcal{Q}) \\
&\quad \vee \textsc{FitCollLeftColumns}(\mathcal{Q}) \\
&\quad\quad \vee \textsc{FitCollRightColumns}(\mathcal{Q})) \\
&\quad \wedge \textsc{FitCollTopRows}(\mathcal{Q}), \\
\textsc{CollBothColumns}(\mathcal{Q}) &= \neg(\textsc{Lucky}(\mathcal{Q}) \vee \textsc{FitInternalColl}(\mathcal{Q}) \\
&\quad \vee \textsc{FitCollLeftColumns}(\mathcal{Q}) \\
&\quad\quad \vee \textsc{FitCollRightColumns}(\mathcal{Q}) \\
&\quad \vee \textsc{FitCollTopRows}(\mathcal{Q})) \\
&\quad \wedge \textsc{FitCollBothColumns}(\mathcal{Q}), \\
\textsc{Preimage}(\mathcal{Q}) &= \neg(\textsc{Lucky}(\mathcal{Q}) \vee \textsc{FitInternalColl}(\mathcal{Q}) \\
&\quad \vee \textsc{FitCollLeftColumns}(\mathcal{Q}) \\
&\quad\quad \vee \textsc{FitCollRightColumns}(\mathcal{Q}) \\
&\quad \vee \textsc{FitCollTopRows}(\mathcal{Q}) \\
&\quad \vee \textsc{FitCollBothColumns}(\mathcal{Q})) \\
&\quad \wedge \textsc{FitPreimage}(\mathcal{Q}).
\end{aligned}$$

Proof of Proposition 6.4. Assume that the adversary is not lucky, *i.e.*, $\neg\textsc{Lucky}(\mathcal{Q})$. Then it is easy to see that

$$\begin{gathered}
\textsc{FitInternalColl}(\mathcal{Q}) \vee \textsc{FitCollLeftColumns}(\mathcal{Q}) \vee \\
\textsc{FitCollRightColumns}(\mathcal{Q}) \vee \textsc{FitCollTopRows}(\mathcal{Q}) \vee \\
\textsc{FitCollBothColumns}(\mathcal{Q}) \vee \textsc{FitPreimage}(\mathcal{Q}) \\
\Longrightarrow \\
\textsc{InternalColl}(\mathcal{Q}) \vee \textsc{CollLeftColumns}(\mathcal{Q}) \vee \\
\textsc{CollRightColumns}(\mathcal{Q}) \vee \textsc{CollTopRows}(\mathcal{Q}) \vee \\
\textsc{CollBothColumns}(\mathcal{Q}) \vee \textsc{Preimage}(\mathcal{Q})
\end{gathered}$$

holds. Therefore, it is sufficient to show that

$$\begin{aligned}\textsc{Coll}^{\text{MDC-4}}(\mathcal{Q}) \quad &\Longrightarrow \textsc{FitInternalColl}(\mathcal{Q})\\ &\quad\vee \textsc{FitCollLeftColumns}(\mathcal{Q})\\ &\quad\vee \textsc{FitCollRightColumns}(\mathcal{Q})\\ &\quad\vee \textsc{FitCollTopRows}(\mathcal{Q})\\ &\quad\vee \textsc{FitCollBothColumns}(\mathcal{Q})\\ &\quad\vee \textsc{FitPreimage}(\mathcal{Q}).\end{aligned}$$

To ensure that the chaining values are always different, we give the attack to the adversary if these values collide, *i.e.*, $V = \widehat{V}$ or $V' = \widehat{V}'$. Note that this is usually not a *real* collision, but we can exclude this case in our analysis. We call this $\textsc{InternalColl}(\mathcal{Q})$. This corresponds to case (i) described in Section 6.2.1.

For the case (ii), we assume that a collision for the MDC-4 compression function $H^{\text{MDC-4}}$ can be constructed from the queries in $\mathcal{Q}$. Then, there are inputs $\mathcal{M}, \mathcal{M}' \in (\{0,1\}^n)^+$, $\mathcal{M} \neq \mathcal{M}'$, such that $\mathcal{H}(\mathcal{M}) = \mathcal{H}(\mathcal{M}')$. In particular, there are message blocks $M, M' \in \{0,1\}^n$ and chaining values $(U, \widehat{U}), (U', \widehat{U}') \in \{0,1\}^{2n}$, $(M, U, \widehat{U}) \neq (M', U', \widehat{U}')$, such that $H^{\text{MDC-4}}(M, U, \widehat{U}) = H^{\text{MDC-4}}(M', U', \widehat{U}')$.

For the following analysis we have $\neg\textsc{InternalColl}(\mathcal{Q})$, *i.e.*, $U \neq \widehat{U}$ and $U' \neq \widehat{U}'$. Our case differentiation is based on the disposal of queries in the bottom row. First, assume that 1BL = 2BL and 1BR = 2BR. Then $\textsc{CollTopRows}(\mathcal{Q})$. Now, assume that 1BL = 2BL and 1BR ≠ 2BR. Then $\textsc{CollRightColumns}(\mathcal{Q})$. Conversely, if 1BL ≠ 2BL and 1BR = 2BR, we say $\textsc{CollLeftColumns}(\mathcal{Q})$. The only missing case, 1BL ≠ 2BL and 1BR ≠ 2BR, is denoted by $\textsc{CollBothColumns}(\mathcal{Q})$. $\textsc{Preimage}(\mathcal{Q})$ formalizes case (iii) of Section 6.2.1 and corresponds to $\textsc{FitPreimage}(\mathcal{Q})$. □

General Remarks. The strategy for the other predicates is to upper bound the probability of the last query being successful conditioned on the fact that the adversary has not yet been successful in its previous queries. We say that the last query is successful if the output is such that $\texttt{NumEqualHalf}(\mathcal{Q}) < \alpha$, $\texttt{NumEqual}(\mathcal{Q}) < \beta$, $\texttt{NumColl}(\mathcal{Q}) < \gamma$, and that one of the predicates is *true*.

Proposition 6.5 (InternalColl($\mathcal{Q}$)).

$$\Pr[\textsc{InternalColl}(\mathcal{Q})] \leq q\left(\frac{\alpha^2+\gamma}{N-q} + \frac{\alpha\beta}{(N-q)(2^{n/2}-\alpha)}\right) + \frac{q\alpha}{N-2^{n/2}\alpha}.$$

Proof. The adversary can use the last query Q_i either once or twice. When Q_i is used three times or more then it must occur twice either in the top- or bottom row. But this would imply $U = \widehat{U}$.

In the case that the query is used once it can either be used in the top row or bottom row. Due to the symmetric structure of MDC-4, we can assume *wlog.* that the last query Q_i is either used in position TL or BL (in this case we only consider the 'left' side of Figure 6.3 and denote 1TL by TL, 1TR by TR, 1BL by BL, and 1BR by BR.). The success probability is analyzed in Lemma 6.6.

In the case that Q_i is used twice, it must be used once in the top row and once in the bottom row. We again assume that the last query is *wlog.* used in TL and BL or in TL and BR. The success probability is analyzed in Lemma 6.7. A union bound, *i.e.*, adding up the individual results, gives our claim. □

Lemma 6.6. *Let $U \neq \widehat{U}$ and assume that Q_i is used once in the* MDC-4 *compression function $H^{\text{MDC-4}}$. Then*

$$\Pr[(V,V) = H^{\text{MDC-4}}(M,U,\widehat{U})] \leq q\left(\frac{\alpha^2+\gamma}{N-q}\right).$$

Proof. Depending on the place of the intended use of the query Q_i, which is either BL or TL, our analysis is as follows.

Case 1: Assume first that $Q_i = (K_i^L || K_i^R, X_i, Y_i)$ is used in position BL. It follows that K_i^L must be equal to the XOR-output Z_{TL}^L of the query in TL. It follows that there are at most α different candidates for the query in TL in the query history $\mathcal{Q}_{i-1}$. Similarly, because K_i^R must be equal to the right half of the XOR-output of TR, Z_{TR}^R, there are at most α candidates for that which can be used in TR. For the query in BR, there are at most α^2 possible key inputs. The ciphertext input of BR is determined by the query used in TL. So the probability that there is a query in $\mathcal{Q}_i$ such that $V = \widehat{V}$ is upper bounded by $\alpha^2/(N-q)$. For q queries, the total chance of success is $\leq q\alpha^2/(N-q)$.

Case 2: Now assume that Q_i is used in position TL. Since $U \neq \widehat{U}$ it follows that BL $\neq$ BR. So, there are at most γ ordered pairs of queries that can be used in BL and BR such that their XOR-output collide. Fixing one of these, it fully determines the XOR-output TL. So, for q queries, Q_i has at most a chance of $q\gamma/(N-q)$. □

Lemma 6.7. *Let $U \neq \widehat{U}$ and assume that Q_i is used twice in the computation of $H^{\text{MDC-4}}$. Then*

$$\Pr[(V,V) = H^{\text{MDC-4}}(M,U,\widehat{U})] \leq q\frac{2\alpha}{(N - \alpha 2^{n/2+1} + \alpha^2)}.$$

Proof. By symmetry arguments, we assume *wlog.*, that the last query Q_i is used in position TL. Since $U \neq \widehat{U}$, the last query can only appear a second time in position BL, or BR but not in TR.

Case 1: Assume Q_i is used in positions TL and BL. This query can be used in these positions if the randomly determined left-side XOR-output Z_i^L is equal to the left-side of the key K_i^L. This event is called P_K and its probability of success can be upper bounded for Q_i by $\Pr[P_K] \leq \frac{1}{2^{n/2}-\alpha}$.

It follows that the left side of BL and therefore the left side of BR, too, get also fixed. So there are at most α queries left that can be used in BR. Assuming that P_K has been successful, the probability that Z_i^R 'matches' the right half of one of the α key inputs in BR is upper bounded by $\alpha/(2^{n/2} - \alpha)$.

So for q queries the total probability of success is upper bounded by $q\alpha/(2^{n/2} - \alpha)^2$.

Case 2: Assume Q_i is used in positions TL and BR. Then, $K_i = X_i$. The query Q_i can be used in these two positions at the same time if the randomly determined right-half XOR-output Z_i^R is equal to the right-half of the key, $K_i^R = X_i^R$. This event is called O_K and its probability of success can be upper bounded for Q_i by $\Pr[O_K] \leq \frac{1}{2^{n/2}-\alpha}$.

We now upper bound the number of queries that can be used in TR conditioned on the fact the O_K is successful. There are at most α queries that can be used in TR such that $Z_{TR}^L = K_i^L$ holds. Hence, there are at most α queries that can be used in BL. We denote the chance that $Z_{BL}^L = Z_i^L$ for the i-the query as $\Pr[Z_i^L]$. This event can thus be upper bounded by $\frac{\alpha}{2^{n/2}-\alpha} \cdot \Pr[O_K] \leq \frac{\alpha}{(2^{n/2}-\alpha)^2}$. For q queries we can upper bound this case by $\frac{q\alpha}{(2^{n/2}-\alpha)^2}$. □

Proposition 6.8 (CollTopRows($\mathcal{Q}$)).

$$\Pr[\textsc{CollTopRows}(\mathcal{Q})] \leq \frac{q\beta}{N-q}.$$

Proof. In this case we consider a collision in the top row, with $1\text{BL} = 2\text{BL}$ and $1\text{BR} = 2\text{BR}$. This implies $U = U'$ and $\widehat{U} = \widehat{U}'$. Furthermore, it implies $M \neq M'$, because otherwise we would have $1\text{TL} = 2\text{TL}$ and $1\text{TR} = 2\text{TR}$. Regarding to this constraints we have to upper bound the probability that the i-th query can be used such that

$$(E_U(M) \oplus M, E_{\widehat{U}}(M) \oplus M) = (E_{U'}(M') \oplus M', E_{\widehat{U}'}(M') \oplus M').$$

Note that no internal collision has happened before, or, more formal, $\neg\textsc{InternalColl}(\mathcal{Q})$, which implies that the chaining values are *always* different. First, assume that the last query is used twice or more. In order to find a collision in the top row, the last query must be used in the top row or otherwise the success probability is zero. Since $M \neq M'$, it follows that $1\text{TL} \neq 2\text{TL}$ and $1\text{TR} \neq 2\text{TR}$. Additionally, $1\text{TL} \neq 2\text{TR}$ must also hold or else $U = \widehat{U}' = U' = \widehat{U}$ would follow. These arguments rule out the possibility of the last query being used twice.

Now, assume that Q_i is used once, *wlog.* in 1TL. Then, there are at most β pairs of queries for $(1\text{TR}, 2\text{TR})$ that form a collision. So there are at most β queries that can be used in 2TL that may form a collision with the XOR-output of the last query used in 1TL. The success probability for q queries can therefore be upper bounded by $q\beta/(N-q)$. □

Proposition 6.9 (CollLeftColumns($\mathcal{Q}$)).

$$\Pr[\textsc{CollLeftColumns}(\mathcal{Q})] \leq$$
$$q\left(\frac{\alpha^4 + \alpha^2 + 3\alpha\gamma + 2\gamma}{N-q} + 6\frac{\alpha^2+1}{N^{1/2}-\alpha}\right).$$

Proof. Since $1\text{TL} \neq 1\text{TR}$ and $2\text{TL} \neq 2\text{TR}$ always, a query can never be used more than twice in the top row. First assume that a query is used twice in the top row. Then, either 1TL = 2TL or 1TR = 2TR. If 1TR = 2TR and – by precondition – 1BR = 2BR, it follows that 1TL = 2TL, *i.e.* the success probability of this case is zero since this would not lead to a collision (or we would have given the collision to the adversary before). The case 1TL = 2TL is upper bounded by Lemma 6.10. The remaining case, *i.e.* all queries used in the top row are pairwise different, is upper bounded by Lemma 6.11. Since no cases are left, a union bound gives our claim.

□

Lemma 6.10. *Let* $U \neq \widehat{U}$, $U' \neq \widehat{U}'$, 1BL $\neq$ 2BL *and* 1BR = 2BR. *Assume that* 1TL = 2TL. *Then,*

$$\begin{aligned}\Pr[H^{\text{MDC-4}}(M, U, \widehat{U}) = & H^{\text{MDC-4}}(M', U', \widehat{U}')] \leq \\ & q \cdot \left(\frac{\gamma\alpha + \alpha^2}{N - q} + 2\frac{\alpha^2 + 1}{N^{1/2} - \alpha} \right)\end{aligned}$$

Proof. Since 1TL = 2TL, it follows that 1TR $\neq$ 2TR. It suffices to analyze the disposition of the last query in in 1TR, 1BL, 2TR, and 2BL since a usage of the last query in 1TL and 2TL reverts to this case. The same is true for the usage of the last query in 1BR and 2BR.

Case 1: The last query is used exactly once in either 1TR, 1BL, 2TR, or 2BL; we *wlog.* assume that it is used in 1TR or 1BL.

Subcase 1.1 The last query is used in position 1TR. There are at most γ queries that can be used in 1BL, 2BL that form a collision. So there are at most $\gamma\alpha$ queries for the pair (1BL, 2TR). The output of the last query is completely determined by that pair so the last query has a chance of success $\leq \gamma\alpha/N'$ and for q queries $\leq q\gamma\alpha/(N - q)$.

Subcase 1.2 The last query is used in position 1BL. The key input of the last query admits at most α possible queries for 1TR. Since the left half of the XOR output of 1TR is equal to the left half of the XOR output of 2TR, there are at most α^2 queries that can be used for 2TR. Since the last query together with 2TR uniquely determines the number of possible queries in 2BL, the probability of success of the last query is upper bounded, for q queries, by $\leq q\alpha^2/(N-q)$.

Case 2: The last query is used twice or more. By symmetry we can assume that the last query is used exactly twice, either in positions 1TR and 1BL or in positions 1TR, 2BL.

Subcase 2.1 The last query is used in positions 1TR and 1BL. The left output of the query has a chance of $\leq 1/(N^{1/2}-\alpha)$ of succeeding since it must match the left half of the key. Conditioned on the success, there are at most α possible queries in 2BL that share the same left XOR output. Now, for any query in 2BL, there are at most α queries that can be used in 2TR. Since the left half of the XOR output of 2TR must match the left half of the XOR output of the last query in 1TR, the left half as a chance $\leq \alpha^2/(N^{1/2}-\alpha)$ of succeeding. The total success probability for q queries is $\leq (\alpha^2+1)/(N^{1/2}-\alpha)$.

Subcase 2.2 The last query is used in position 1TR and 2BL. There are at most α possible choices for 2TR given the key input of the last query. Since the left halves of the XOR outputs of 1TR and 2TR must be equal, the right half of the last query has a chance of success of $\leq q/(N^{1/2}-\alpha)$. If the left half is successful, there are at most α possible queries for 1BL (with that right half XOR output), so the left half has a chance of success of $\leq \alpha/(N^{1/2}-\alpha)$. For q queries, the total probability of success is $\leq q(\alpha^2+1)/(N^{1/2}-\alpha)$.

Adding up Case 1 and Case 2, the overall chance of success is

$$\leq q \cdot \Big(\gamma\alpha/(N-q) + \alpha^2/(N-q) + 2(\alpha^2+1)/(N^{1/2}-\alpha)\Big)$$

□

Lemma 6.11. *Let* $U \neq \widehat{U}$, $U' \neq \widehat{U}'$, 1BL $\neq$ 2BL *and* 1BR $=$ 2BR. *Assume that* 1TL $\neq$ 2TL *and* 1TR $\neq$ 2TR. *Then,*

$$\Pr[H^{\text{MDC-4}}(M, U, \widehat{U}) = H^{\text{MDC-4}}(M', U', \widehat{U}')] \leq \\ q \cdot \left(\frac{\alpha^4 + 2\alpha\gamma + 2\gamma}{N-q} + 4\frac{\alpha^2+1}{N^{1/2}-q}\right).$$

Proof. **Case 1:** The last query is only used once, *wlog.* in either 1BL, 1TL, or 1TR.

Subcase 1.1 The last query is used in position 1BL. There are at most α possible choices for query in 1TL, and at most α possible queries for 1TR. Then, since 1BR $=$ 2BR, there are at most α^2 possible queries for 2TL and at most α^2 possible queries for 2TR. So there are at most α^4 possible choices for 2BL. So the probability of success for q queries is $\leq q\alpha^4/(N-q)$.

Subcase 1.2 The last query is used in position 1TL. There are at most γ possible pairs for queries for 1BL and 2BL. For any query in 2BL, there are at most α queries in 2TL. Since the output of the last query is completely determined by the queries $(1\text{BL}, 2\text{TL})$ the probability of success for q queries is $\leq q\alpha\gamma/(N-q)$.

Subcase 1.3 The last query is used in position 1TR. The analysis is the same as for 1TL giving the same bound of $\leq q\alpha\gamma/(N-q)$.

Case 2: The last query is used twice or more. Then it is used at least once in the top-row, *wlog.* at least in 1TL or 1TR. Our analysis assumes that the last query is used in 1TL, since the case where it is used in 1TR is essentially the same (we adjust for this second case by doubling our probabilities of success for the 1TL case).

Subcase 2.1 The last query is also used in 1BL. The left half of the XOR output of the last query has a chance $\leq 1/(N^{1/2} - \alpha)$ of being successful. If the left output half is successful, there are at most α different queries 2BL with that left XOR output and for each of them there are at most α possible choices for 2TL. So the right output half has a chance of success of $\leq \alpha^2/(N^{1/2} - q)$. For q queries, the chance of success is $\leq (\alpha^2 + 1)/(N^{1/2} - q)$.

Subcase 2.2 The last query is also used in 2BL. Given the key input of the last query in 2BL, the right half of the XOR output has a chance of success of $\leq 1/(N^{1/2} - \alpha)$. If the right half is successful, then there are at most α queries for 1BL so the left half of the XOR output has a chance of success of $\leq \alpha/(N^{1/2} - \alpha)$ of being successful. The total chance for q queries is therefore $\leq (\alpha^2 + 1)/(N^{1/2} - q)$.

Subcase 2.3 The last query is also used in 2TR. It does not appear in 1BL or 2BL (or our analysis of the prior subcases would hold). There are at most γ pairs for 1BL and 2BL, so the last query has a chance $\leq \gamma/(N - q)$ of success and for q queries $\leq q\gamma/(N - q)$.

□

Proposition 6.12 (CollRightColumns($\mathcal{Q}$)).

$$\Pr[\textsc{CollRightColumns}(\mathcal{Q})] \leq$$
$$q\left(\frac{\alpha^4 + \alpha^2 + 3\alpha\gamma + 2\gamma}{N - q} + 6\frac{\alpha^2 + 1}{N^{1/2} - \alpha}\right).$$

Proof. Due to the symmetric structure of MDC-4 this proof is essentially the same as for Proposition 6.9. □

Proposition 6.13 (CollBothColumns($\mathcal{Q}$)).

$$\Pr[\textsc{CollBothColumns}(\mathcal{Q})] \leq q\left(\frac{\gamma\alpha^2+\gamma^2}{N-q}+\frac{2\alpha}{(2^{n/2}-\alpha)^2}\right).$$

Proof. In Case 1, we discuss the implication if the last query is only used once, the Cases 2-4 give bounds if the last query is used at least twice.

Case 1: The last query is used exactly once. We can *wlog.* assume the it is either used in 1TL or 1BL.

Subcase 1.1: The last query is used in position 1BL. Since 1BR = 2BR, there are at most γ pairs of queries in the query history that can be used for positions 1BR and 2BR. Now, for any query 2BR, there are at most α matching queries in position 2TL and at most α matching queries in 2TR. Since the queries in 2TL and 2TR uniquely determine the query 2BL, there are at most $\gamma\alpha^2$ queries that can be used for 2BL. Therefore, the last query has a chance of being successful $\leq \gamma\alpha^2/(N-q)$. For q queries, the total chance of success in this case is $\leq q\gamma\alpha^2/(N-q)$.

Subcase 1.2: The last query is used in position 1TL. There are at most γ possible pairs of queries that can be used for 1BL and 2BL and there are at most γ possible queries that can be used for 1BR and 2BR. We now upper bound the probability that the last query can be used in 1TL assuming a collision. There are at most γ^2 pairs of queries that can be used for 1BL and 1BR. Therefore, the success probability of the last query can be upper bounded by $\leq \gamma^2/(N-q)$ and for q queries by $q\gamma^2/(N-q)$.

Case 2: The last query is only used in the bottom row. Then it is used exactly twice, *wlog.* in positions 1BL and 2BR. This would imply $V = \widehat{V}'$ which then – in the case of success – implies InternalColl($\mathcal{Q}$).

Case 3: The last query is only used in the top row. We can *wlog.* assume it is used in 1TL. We can use the same reasoning as in Subcase 1.2 and therefore extend Subcase 1.2 to also handle this slightly more general situation here.

Case 4: The last query is used at least once in the bottom row and at least once in the top row. We can *wlog.* assume that it is used in position 1TL. Using the same argument as for Case 2, the last query must then appear exactly once in the bottom row. The following four subcases discuss the implications of the last query being also used in 1BL, 1BR, 2BL and 2BR. Note that the adversary may use it also a second time in the top row – apart from 1TL– but this does not change our bounds.

Subcase 4.1: The last query is also used in 1BL. The left half of the XOR-output of 1TL has a chance of being equal to its key input (*i.e.*, the key input of 1BL) of $\leq 1/(2^{n/2}-\alpha)$. The following analysis is now based on the fact that the left half of the XOR-output has matched the left half of the key input. Since we now also know the left half of the XOR-output of 2BL, there are at most α queries that can be used in 2BL. The chance that the right half of the XOR-output of 2BL matches the right half of the XOR-output of 1BL is therefore $\leq \alpha/(2^{n/2}-\alpha)$. So, for q queries, the total chance of success is $\leq q\alpha/(2^{n/2}-\alpha)^2$.

Subcase 4.2: The last query is also used in 1BR. The same arguing as for Subcase 4.1 can be used (apart from exchanging 'left' and 'right') and the bound for q queries is again $\leq q\alpha/(2^{n/2} - \alpha)^2$.

Subcase 4.3 The last query is also used in position 2BL. There are at most γ possible pairs of queries in the query history that can be used for the pair 1BR, 2BR that form a collision. The probability that the right half of the XOR-output of 1TL matches the right half of its key input (*i.e.*, for the last query being also used in 1BR) is $\leq 1/(2^{n/2} - \alpha)$. Conditioned on the fact that the right half of the XOR-output is now fixed there are at most α queries that can be used in 1BL such that the XOR-outputs of 1BL and 2BL collide. The probability that the left half of the XOR-output of 1TL is equal to the left half of the key of 1BL is therefore $\leq \alpha/(2^{n/2} - \alpha)$ and the total chance of success for q queries is $\leq q\alpha/(2^{n/2} - \alpha)^2$.

Subcase 4.4 The last query is also used in 2BR. The same arguing as for Subcase 4.3 can be used (apart from exchanging 'left' and 'right') and the bound for q queries is again $\leq q\alpha/(2^{n/2} - \alpha)^2$. □

Proposition 6.14 (Preimage($\mathcal{Q}$)).

$$\Pr[\textsc{Preimage}(\mathcal{Q})] \leq \frac{q(4+\beta^2)}{N-q}.$$

Proof. The adversary can use the last query either once or twice. If it is used twice, it is used at least once in the bottom row.

Case 1: Assume first, that the last query is used once and that it is used in the top row. Assume *wlog.* that it is used in 1TL. Since there are at most β queries that can be used in 1BL and also at most β queries for 1BR, the success probability for q queries is upper bounded by $q\beta^2/(N-q)$.

Now, assume that the last query is used once and that it is used in the bottom row. Whether it is used in 1BL or 1BR, the success probability in each case for one query is $\leq 1/(N-q)$.

So, the total success probability for q queries for this case is upper bounded by $q(2+\beta^2)/(N-q)$.

Case 2: Now, assume that the last query is used twice. So it is used exactly once in the bottom row and the analysis of Case 1 (bottom row) gives an upper bound of $2q/(N-q)$.

□

Proposition 6.15. *Let $n, q \in \mathbb{N}$, $n \geq q$. Let α, β, and γ be as in Theorem 6.2 with $eq2^{n/2}/(N-q) \leq \alpha$ and $eq/(N-q) \leq \beta$. Set $\tau = \frac{\alpha(N-q)}{q2^{n/2}}$ and $\nu = \frac{\beta(N-q)}{q}$. Then*

$$\begin{aligned}\Pr[\textsc{Lucky}(\mathcal{Q})] \leq & \frac{q^2}{\gamma(N-q)} + 2q2^{n/2}e^{q2^{n/2}\tau(1-\ln\tau)/(N-q)} \\ & + qNe^{qN\nu(1-\ln\nu)/(N-q)}.\end{aligned}$$

Proof. This is a special case of the results discussed in Section 5.4. A direct proof can also be found in [161, Appendix B]. □

6.2.5. Proof of Theorem 6.2

The proof of Theorem 6.2 now follows with Proposition 6.4 by adding up the individual results from Propositions 6.5, 6.8, 6.9, and 6.12 - 6.15. □

Part II.

Preimage Security of Double Length Compression Functions

7 Results Summary

7.1. Introduction

As discussed in the first part of this thesis, collision resistance has been successfully resolved for a lot of double call double length compression functions using a blockcipher from Block$(2n, n)$. On the other hand, the corresponding situation for preimage resistance has been far less satisfactory. Up to now, it was an open problem to prove preimage resistance for values of q higher than 2^n for either Abreast-DM, Tandem-DM, or Hirose-DM. This is not to say that no dedicated preimage security proofs have appeared in the literature. For instance, Lee *et al.* [107] provide a preimage resistance bound for Tandem-DM that is a lot closer to 2^n than a straightforward implication [149] of their collision bound would give. However, a barrier occurs once 2^n queries are reached: namely, a blockcipher 'loses randomness' after being queried $\Omega(2^n)$ times on the same key (for example, when $2^n - 1$ queries have been made to a blockcipher under a given key, the answer to the last query under that key is deterministic). Going beyond the 2^n barrier seemed to require either a very technical probabilistic analysis, or some brand new idea. In this thesis, we show a new idea which delivers tight bounds in a quite pain-free and non-technical fashion.

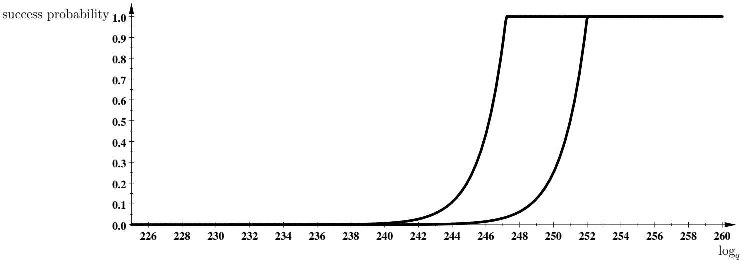


Figure 7.1.: Preimage bounds for the classical constructions: Abreast-DM, Tandem-DM (left), and Hirose-DM (right).

We prove that various compression functions that turn a block-cipher of $2n$-bit key into a double-block-length hash function, have preimage resistance close to the optimal 2^{2n} in the ideal cipher model. Our analysis covers many relevant proposals, such as Abreast-DM, Hirose-DM, and Tandem-DM. Bounds for the case $n = 128$ are depicted in Figure 7.1 and summarized in Table 7.1.

At the heart of our result are so-called 'super queries', a new technique to restrict the advantage of an adaptive preimage-finding adversary.

To build some intuition for our result, let us start by considering the much easier problem of constructing a $3n$-bit to $2n$-bit compression function H based on two $3n$-bit to n-bit smaller underlying primitives f and f'. An obvious approach is simply to concatenate the outputs of f and f', that is let $H(B) = f(B)\|f'(B)$ for $B \in \{0,1\}^{3n}$. If f and f' are modeled as independently sampled, ideally random functions, then it is not hard to see that H behaves ideally as well. In particular, it is preimage resistant up to 2^{2n} queries (to f and f').

Comp. Function	Preimage bound	Section	Published
Hirose-DM	$2^{2n-5} = 2^{251}$	8.2	[3]
Weimar-DM	$2^{2n-5} = 2^{251}$	8.3	[54]
Abreast-DM	$2^{2n-10} = 2^{246}$	8.4	[3]
Tandem-DM	$2^{2n-10} = 2^{246}$	8.5	[3]

Table 7.1.: Overview of our preimage security results for double length compression functions. The results are in the ideal cipher model.

When switching to a block cipher-based scenario, it is natural to replace f and f' in the construction above by E, resp. E', both run in Davies–Meyer mode. In other words, for block ciphers E and E' both with $2n$-bit keys and operating on n-bit blocks, define $H(A\|B) = (E_B(A) \oplus A)\|(E'_B(A) \oplus A)$ where $A \in \{0,1\}^n$ and $B \in \{0,1\}^{2n}$. While there is every reason to believe this construction maintains preimage resistance up to 2^{2n} queries, the standard proof technique against adaptive adversaries falls short significantly. Indeed, the usual argument goes that the i-th query an adversary makes to E using key K will return an answer uniform from a set of size at least $2^n - (i-1)$ and thus the probability of hitting a prespecified value is at most $1/(2^n - (i-1)) < 1/(2^n - q)$. Unfortunately, once q approaches 2^n, the denominator tends to zero (rendering the bound useless). As a result, one cannot hope to prove anything beyond 2^n queries using this method. This restriction holds even for a typical bound of type $q/(2^n - q)^2$.

When considering *non-adaptive* adversaries only, the situation is far less grim. Such adversaries need to commit to all queries in advance, which allows bounding the probability of each individual query hitting a prespecified value by 2^{-n}. While obviously there are dependencies (in the answers), these can safely be ignored when a union bound is later used to combine the various individual queries. Since the

q offset has disappeared from the denominator, the typical bound $q/(2^n)^2$ *would* give the desired security.

Our solution, then, is to force an adaptive adversary to behave non-adaptively. As this might sound a bit cryptic, let us be more precise. Consider an adversary adaptively making queries to the blockcipher, using the same key throughout. As soon as the number of queries *to this key* passes a certain threshold, we give the remaining queries to the blockcipher using this very key *for free*. We will refer to this event as a *super query*. Since these free queries are all asked in one go, they can be dealt with non-adaptively, preempting the problems that occur (in standard proofs) due to adaptive queries. Nonetheless, for every super query we need to hand out a very large number of free queries, which can aid the adversary. Thus we need to limit the amount of super queries an adversary can make by setting the threshold that triggers a super query sufficiently high. In fact, we set the threshold at exactly half[1] the total number of queries that can be made under a given key (i.e., it is set to $2^n/2$ queries). This effectively doubles the adversary's query budget, since for every query the adversary makes it can get another one later for free (if it keeps on making queries under the same key), but such a doubling of the number of queries does not lead to an unacceptable deterioration of the security bound.

With this new technique in hand, we can prove in Section 8.1 that the construction H given above has indeed an asymptotically optimal preimage resistance bound (a generalization of this result is given in Section 8.6). Afterwards, we revisit the proofs of preimage resistance of the three main double-block-length, double-call constructions: Hirose-DM (Section 8.2), Abreast-DM (Section 8.4) and Tandem-DM (Section 8.5).

An additional technical problem is that these compression functions make two calls to the same blockcipher each, as opposed to using two calls to independent block ciphers. Ideally, to get a good

[1] The optimized threshold turns out to be very near one half, but a bit less; we set the threshold at a half for simplicity in our proofs.

bound, one would like to query the two calls necessary for a single compression function evaluation in conjunction (this would allow using the randomness of both calls simultaneously, potentially leading to a denominator 2^{2n} as desired for preimage resistance). For instance, in the context of collision resistance for Hirose-DM and Abreast-DM corresponding queries are grouped in cycles (of length 2 and 6, respectively) and all queries in a cycle are made simultaneously: if the adversary makes one query in a cycle, the remaining queries are handed out for free. Care has to be taken that these free queries and the free queries due to super queries do not reinforce each other to untenable levels.

For Hirose-DM, there are no problems as the free queries introduced by a super query necessarily consist of full cycles only. The corresponding (upper) bound on the preimage finding advantage is $16q/2^{2n}$ which is as desired, up to a small factor. For Abreast-DM, however, the cyclic nature can no longer be exploited: any super query introduces many partial cycles, yet freely completing these might well trigger a new super query, etc. Luckily, the original preimage proof for Tandem-DM [107] (which does not involve cycles) provides a way out of this conundrum. The downside, however, is that our preimage bound for Abreast-DM, and Tandem-DM is slightly less tight than that for Hirose's scheme. Ignoring negligible terms, it grows roughly as $16\sqrt{q}/2^n$. Although this is faster than one might wish for (as can be seen in Figure 7.1), it does imply that $\Omega(2^{2n})$ queries are required to find a preimage with constant probability.

7.2. Proof Model

As our preimage security notion for a blockcipher based double call, double length compression function H, we adopt everywhere preimage resistance in the information theoretic setting as it was introduced in Section 1.1.1 on Page 11.

We let $\mathcal{Q}_i = \{(K_j, X_j, Y_j)\}_{j=1}^{i}$ be the first i elements of the query history; thus $\mathcal{Q} = \mathcal{Q}_q$.

For TANDEM-DM, it turns out that the everywhere preimage resistance notion is slightly too strong, as there is one weak point (namely 0^{2n}) in the range, for which finding preimages is a bit easier. A simple adaptation of the everywhere preimage resistance definition is to disallow the adversary to choose $(V, \widehat{V}) = 0^{2n}$ as the target point [107]; we denote the corresponding advantage as

$$\mathbf{Adv}_H^{\mathrm{EPRE}\neq 0}(q) \ .$$

(We will still use the same predicate $\mathsf{Preim}(\mathcal{Q})$ though.)

As for our collision security proofs, we again assume that the adversary never makes a query to which it already knows the answer. By this it is meant, for example, that one can assume that the adversary never makes a query $E_K(X)$, obtaining an answer Y, and then makes the query $E_K^{-1}(Y)$ (which will necessarily be answered by X). In the current context, where we consider adversaries making 2^n queries or more, this assumption should be more precisely restated as *the adversary never makes a query that will result in a triple* (K, X, Y) *which is already present in the query history.* (This latter assumption can be made without loss of generality using the fact that $E_K(\cdot)$ is a permutation.) Indeed, if an adversary has made $2^n - 1$ queries under a key K, the result of the last query under that key is predetermined, and thus the adversary already knows the answer to this query. However, one should not forbid the adversary from making this query, since the query may be necessary to complete a preimage.

Our preimage security proofs, again, use the notion of 'free queries' which already has been discussed, *e.g.*, on Page 52.

8 Applications

8.1. Example Application

Before we apply the new technique of super queries to the analysis of three well-known constructions that compress $3n$ bits to $2n$ bits and that each call the same blockcipher twice, we demonstrate our technique at the following simplest possible example. We consider the construction H^{SPL}, compressing $3n-1$ bits to $2n$ bits that makes two blockcipher calls. Given a blockcipher $E \in \textsc{Block}(2n, n)$, an input block $M \in \{0,1\}^n$, and a key prefix $K \in \{0,1\}^{2n-1}$ we define

$$\mathrm{H}^{\mathrm{SPL}}(M, K) = (E_{K\|0}(M) \oplus M, E_{K\|1}(M) \oplus M)$$

where $\|$ denotes the concatenation of two bit strings. If we consider the ideal cipher model, the two blockcipher calls are independent. Hence H^{SPL} is exactly the construction mentioned within the introduction (Section 7.1). H^{SPL} can be seen as a simple special case of a scenario discussed in Section 8.6, where two different block ciphers are called.

Theorem 8.1. *Let* $\mathrm{H}^{\mathrm{SPL}} : \{0,1\}^n \times \{0,1\}^{2n-1} \to \{0,1\}^n \times \{0,1\}^n$ *be the block cipher-based compression function as defined above. Then*

$$\mathbf{Adv}^{\mathrm{EPRE}}_{\mathrm{H}^{\mathrm{SPL}}}(q) \leq 8q/N^2.$$

In particular, to achieve an advantage of $1/2$ *the adversary has to make at least* 2^{2n-4} *queries.*

Proof. Let $(V, \widehat{V}) \in \{0,1\}^n \times \{0,1\}^n$ be the point to invert (chosen by the adversary before it makes any queries to E). We upper bound the probability that, in q queries, the adversary finds a point $A \in \{0,1\}^n$ and a key prefix $K \in \{0,1\}^{2n-1}$ such that $\mathrm{H}^{\mathrm{SPL}}(A, K) = (V, \widehat{V})$. On top of the q queries the adversary wants to make, we give it several queries for free, as follows.

Normal forward query If the adversary queries for $E_{K\|0}(X)$ (resp. $E_{K\|1}(X)$) for some key prefix $K \in \{0,1\}^{2n-1}$ and $X \in \{0,1\}^n$, we also give it for free $E_{K\|1}(X)$ (resp. $E_{K\|0}(X)$).

Normal inverse query Now, given the inverse query $E^{-1}_{K\|0}(Y)$ (resp. $E_{K\|1}(Y')$) with answer X for some $K \in \{0,1\}^{2n-1}$ and $Y, Y' \in \{0,1\}^n$, the corresponding query $E_{K\|1}(X)$ (resp. $E_{K\|0}(X)$) is given for free.

We note that, as a result of these additional queries, the elements $(K\|0, X, Y)$ and $(K\|1, X, Y')$ are always added to the query history $\mathcal{Q}$ as a pair. We call such a pair an *adjacent query pair* with respect to the key prefix $K \in \{0,1\}^{2n-1}$.

We now give further free queries to the adversary, in the fashion described next. After each adjacent query pair has been completed (namely, after the adversary has received the response to both its query and its associated free query, and after these have been placed in the query history), we check whether the key prefix used for the latest query is such that the (current) query history contains exactly $N/2$ adjacent query pairs with this key prefix. If so, we give *all* remaining adjacent query pairs under this key for free to the adversary. There will be exactly $N/2$ such query pairs. We insert these $N/2$ free query pairs into the query history pair-by-pair (to maintain, mostly for conceptual simplicity, the adjacent pair structure of the query history). We note that, after these free queries have been inserted into the query history, the adversary cannot make any more queries under this key prefix, since the adversary is assumed never to make a query to which it knows the answer. When $N/2$ free query pairs are given to the adversary in the fashion just described, we say that a *super query* occurs. This can be summed up as follows:

Super query Given $N/2$ adjacent query pairs to E all using the same key prefix $K \in \{0,1\}^{2n-1}$, all the remaining queries using the same key prefix K are given for free.

We say that an adjacent query pair $(K\|0, X, Y)$, $(K\|1, X, Y')$ is *winning*, or *successful*, if $X \oplus Y = V$ and $X \oplus Y' = \widehat{V}$. Thus the adversary obtains a preimage of $(V, \widehat{V})$ precisely if it obtains a winning adjacent query pair. This can occur in one of two ways: either the winning query pair is part of a super query, or not. We let $\mathsf{SuperQueryWin}(\mathcal{Q})$ denote the event that the adversary obtains a winning query pair that is part of a super query, and $\mathsf{ForwardQueryWin}(\mathcal{Q})$ the event that the adversary obtains a winning query pair of normal queries. It thus suffices to upper bound

$$\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})] + \Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})].$$

Here, probabilities are taken (as usual) over the adversary's randomness (if any) and over the randomness of the ideal cipher.

We first upper bound $\Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})]$. Note that when the adversary makes, say, a forward query $E_{K\|0}(X)$, at most $N/2-1$ queries have been previously answered to the key $K\|0$ and at most $N/2-1$ queries have been previously answered to the key $K\|1$, since otherwise a super query for the key prefix K would have occurred. Thus the values $Y = E_{K\|0}(X)$ and $Y' = E_{K\|1}(X)$ come uniformly and independently at random from a set of size at least $N/2+1 \geq N/2$, and there is a chance of at most $(1/(N/2))^2 = 4/N^2$ that we obtain a winning pair of adjacent queries. The same is true if the adversary makes a forward query $E_{K\|1}(X)$ or an inverse query $E^{-1}_{K\|0}(Y)$ or an inverse query $E^{-1}_{K\|1}(Y')$. Since the adversary makes q queries in total, we therefore have

$$\Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})] \leq 4q/N^2. \tag{8.1}$$

We now bound $\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})]$. Say a super query is about to occur on key prefix $K \in \{0,1\}^{2n-1}$, meaning that the value of $E_{K\|0}(\cdot)$ and $E_{K\|1}(\cdot)$ is already known on exactly $N/2$ points. Let us denote this set of points by $\mathcal{X}$, and let $\mathcal{Y} = E_{K\|0}(\mathcal{X})$ and $\mathcal{Y}' = E_{K\|1}(\mathcal{X})$. Further, let $\mathcal{R} = \{0,1\}^n \setminus \mathcal{X}$, $\mathcal{S} = \{0,1\}^n \setminus \mathcal{Y}$, and $\mathcal{S}' = \{0,1\}^n \setminus \mathcal{Y}'$. Note that $|\mathcal{X}| = |\mathcal{Y}| = |\mathcal{Y}'| = |\mathcal{R}| = |\mathcal{S}| = |\mathcal{S}'| = N/2$.

Now let a point $A \in \mathcal{R}$ in the domain of the super query be arbitrarily fixed, and let us estimate the probability that point A induces a winning pair under E. If $A \oplus V \in \mathcal{Y}$ or if $A \oplus \widehat{V} \in \mathcal{Y}'$, this probability is zero. Consequently, let us suppose that $A \oplus V \in \mathcal{S}$ and $A \oplus \widehat{V} \in \mathcal{S}'$.

The probability that $E_{K\|0}(A) = A \oplus V$ and $E_{K\|1}(A) = A \oplus \widehat{V}$ equals $1/(N/2) \cdot 1/(N/2)$. Thus we find that the probability of the super query producing a winning pair of adjacent queries is at most

$$N/2 \cdot \left(\frac{1}{N/2}\right)^2 = \frac{1}{N/2}$$

We now observe that at most $q/(N/2)$ super queries can ever occur, since each super query requires a setup cost of $N/2$ queries. Thus

$$\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})] \leq 4q/N^2. \tag{8.2}$$

Summing up (8.1) and (8.2) completes the proof. □

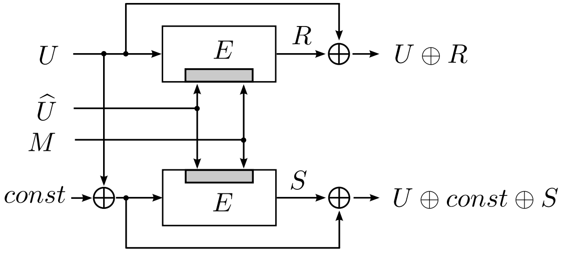


Figure 8.1.: Notations used in this proof for the HIROSE-DM compression function. All wires carry n-bit values. The top and bottom block ciphers, which are the same blockcipher, have $2n$-bit key and n-bit input/output. The wires M, U and $\widehat{U}$ are the inputs to the compression function. The bottom left-hand wire is not an input; it carries an arbitrary nonzero constant *const*.

8.2. Hirose-DM

Theorem 8.2. *Let $H^{\mathrm{HDM}} : \{0,1\}^{3n} \to \{0,1\}^{2n}$ be the* HIROSE-DM *compression function as depicted in Figure 8.1 and defined in Section 1.4 on Page 29. Then*

$$\mathbf{Adv}^{\mathrm{EPRE}}_{H^{\mathrm{HDM}}}(q) \leq 8q/N^2 + 8q/N(N-2).$$

In particular, $\mathbf{Adv}^{\mathrm{EPRE}}_{H^{\mathrm{HDM}}}(q)$ is upper bounded by approximately $16q/N^2$.

Proof. Let $(V, \widehat{V}) \in \{0,1\}^n \times \{0,1\}^n$ be the point to invert (chosen by the adversary before it makes any queries to E). We upper bound the probability that, in q queries, the adversary finds a point $(M, U, \widehat{U}) \in (\{0,1\}^n)^3$ such that $H^{\mathrm{HDM}}(M, U, \widehat{U}) = (V, \widehat{V})$.

When the adversary makes a *forward query* $E_{\widehat{U}\|M}(U)$ we give it for free, also, the answer to the query $E_{\widehat{U}\|M}(U \oplus \mathit{const})$. Moreover, when the adversary makes a *backward query* $E^{-1}_{\widehat{U}\|M}(R)$, resulting in an answer $U = E^{-1}_{\widehat{U}\|M}(R)$, we give it for free the answer to the forward query $E_{\widehat{U}\|M}(U \oplus \mathit{const})$. Also, we assume that the adversary never makes a query to which it knows the answer. Thus the elements of the adversary's query history $\mathcal{Q}$ can be paired into adjacent pairs of the form $(\widehat{U}\|M, U, R), (\widehat{U}\|M, U \oplus \mathit{const}, S)$. We call such a pair an "adjacent query pair". Furthermore, we define *super queries* analogously to the definition used in the proof of Theorem 8.1. That is, as soon as the (current) query history contains exactly $N/2$ queries with the same key, *all* remaining queries under this key are given for free to the adversary. We say that an adjacent query pair $(\widehat{U}\|M, U, R)$, $(\widehat{U}\|M, U \oplus \mathit{const}, S)$ is "winning", or "successful", if $U \oplus R = V$ and $U \oplus \mathit{const} \oplus S = \widehat{V}$, or if $U \oplus R = \widehat{V}$ and $U \oplus \mathit{const} \oplus S = V$. Thus the adversary obtains a preimage of $(V, \widehat{V})$ precisely if it obtains a winning adjacent query pair. This can occur in one of two ways: either the winning query pair is part of a super query, or not. We let $\mathsf{SuperQueryWin}(\mathcal{Q})$ denote the event that the adversary obtains a winning query pair that is part of a super query, and $\mathsf{ForwardQueryWin}(\mathcal{Q})$ the event that the adversary obtains a winning query pair of normal queries. It thus suffices to upper bound

$$\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})] + \Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})].$$

Here, probabilities are taken (as usual) over the adversary's randomness (if any) and over the randomness of the ideal cipher.

We first upper bound $\Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})]$. Note that when the adversary makes, say, a forward query $E_{\widehat{U}\|M}(U)$, at most $N/2-2$ queries (counting free queries) have been previously answered with the key $\widehat{U}\|M$, since otherwise a super query for the key $\widehat{U}\|M$ would have occurred. Thus the value $R = E_{\widehat{U}\|M}(U)$ comes uniformly at random from a set of size at least $N/2+2 \geq N/2$, and there is chance at most $2/(N/2) = 4/N$ that either $U \oplus R = V$ or $U \oplus R = \widehat{V}$ (this is also true if $V = \widehat{V}$). If, say, $U \oplus R = V$, there is further chance at most $1/(N/2) = 2/N$ that the free query $E_{\widehat{U}\|M}(U \oplus \mathit{const})$ returns $U \oplus \mathit{const} \oplus \widehat{V}$, since the answer to the free query comes uniformly at random from a set of size at least $N/2 + 1 \leq N/2$. Other cases (*e.g.*, when $U \oplus R = \widehat{V}$, and when the adversary makes a backward query $E^{-1}_{\widehat{U}\|M}(R)$) are similarly analyzed, showing that the adversary's chance of triggering the event $\mathsf{ForwardQueryWin}(\mathcal{Q})$ at any given query is at most $(4/N)(2/N) = 8/N^2$. Since the adversary makes q queries total, we therefore have

$$\Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})] \leq 8q/N^2. \tag{8.3}$$

We now bound $\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})]$. Say a super query is about to occur on key $\widehat{U}\|M$, meaning that the value of $E_{\widehat{U}\|M}(\cdot)$ is already known on exactly $N/2$ points paired into $N/4$ query pairs. Let U and $U \oplus \mathit{const}$ be in the domain of the super query. (We say that a point $B \in \{0,1\}^n$ is "in the domain of the super query" if $E_{\widehat{U}\|M}(B)$ is not yet known, and will be queried as part of the super query; note that a point $A \in \{0,1\}^n$ is in the domain of the super query if and only if $A \oplus \mathit{const}$ is in the domain of the super query.) Then, the probability that $E_{\widehat{U}\|M}(U) = R$ is either 0 if R is not in the range of the super query (meaning there is a normal query $E_{\widehat{U}\|M}(B) = R$ already present in the query history when the super query is made), or else is exactly $2/N$, since the value of $E_{\widehat{U}\|M}(U)$ returned by the super query is uniform at random in a set of size $N/2$. Thus by a similar argument on S, the probability that $E_{\widehat{U}\|M}(U) \in \{R, S\}$ is at most $4/N$. Conditioning on the event $E_{\widehat{U}\|M}(U) \in \{R, S\}$, the probability that $E_{\widehat{U}\|M}(U \oplus \mathit{const}) \in \{R, S\}$ is at most $1/(N/2 - 1)$, since $E_{\widehat{U}\|M}(U \oplus \mathit{const})$ is sampled uniformly at random from a set of size $N/2 - 1$, once the value $E_{\widehat{U}\|M}(U)$ is known. Thus the probability that the super query returns values such that the adjacent query pair $(\widehat{U}\|M, U, \cdot)$, $(\widehat{U}\|M, U \oplus \mathit{const}, \cdot)$ is winning is at most $4/N(N/2 - 1)$.

But $U, U \oplus const$ were two arbitrary paired domain points; taking a union bound over $N/4$ such pairs in the domain of the super query, we find that the probability of the super query producing a winning pair of adjacent queries is at most

$$(N/4) \cdot (4/N(N/2-1)) = 1/(N/2-1).$$

We now observe that at most $q/(N/4)$ super queries can ever occur, since each super query requires a "setup" cost of $N/4$ queries. Thus

$$\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})] \leq 4q/N(N/2-1). \tag{8.4}$$

Summing up (8.3) and (8.4) completes the proof. □

8.3. Weimar-DM

We now give a preimage security analyis of Weimar-DM as introduced in Chapter 3.

Theorem 8.3. *Let $N = 2^n$. Then,* $\mathbf{Adv}^{\mathrm{EPRE}}_{H^{\mathrm{WDM}}}(q) \leq 16q/N^2$.

It is easy to see that $\mathbf{Adv}^{\mathrm{EPRE}}_{H^{\mathrm{WDM}}}(2^{2n-5}) = 1/2$ and therefore our bound is asymptotically optimal for a $2n$-bit compression function.

Proof. Let $(V, \widehat{V}) \in \{0,1\}^n \times \{0,1\}^n$ be the point to invert (chosen by the adversary before it makes any queries to E). We upper bound the probability that, in q queries, the adversary finds a point $(M, U, \widehat{U}) \in (\{0,1\}^n)^3$ such that $H^{\mathrm{WDM}}(M, U, \widehat{U}) = (V, \widehat{V})$.

When the adversary makes a (normal) *forward query* $E_{M\|U}(\widehat{U})$ we give it for free, also, the answer to the query $E_{\overline{M\|U}}(\widehat{U})$. Moreover, when the adversary makes a (normal) *backward query* $E^{-1}_{M\|U}(R)$, resulting in an answer $\widehat{U} = E^{-1}_{M\|U}(R)$, we give it for free the answer to the forward query $E_{\overline{M\|U}}(\widehat{U})$. As discussed, we assume that the adversary never makes a query to which it knows the answer. Thus the elements of the adversary's query history $\mathcal{Q}$ can be matched into adjacent pairs of the form $(M\|U, \widehat{U}, R), (\overline{M\|U}, \widehat{U}, S)$. We call such a pair an *adjacent query pair.*

We now give further free queries to the adversary, in the fashion described next. After each adjacent query pair has been completed (namely, after the adversary has received the response to both its query and its associated free query, and after these have been placed in the query history), we check whether the key prefix used for the latest query is such that the (current) query history contains exactly $N/2$ adjacent query pairs with this key prefix. If so, we give *all* remaining adjacent query pairs under this key for free to the adversary. There will be exactly $N/2$ such query pairs. We insert these $N/2$ free query pairs into the query history pair-by-pair (to maintain, mostly for conceptual simplicity, the adjacent pair structure of the query history). We note that after these free queries have been inserted into the query history, the adversary cannot make any more queries under this key prefix, since, the adversary is assumed never to make a query to which it knows the answer. When $N/2$ free query pairs are given to the adversary in the fashion just described, we say that a *super query* occurs. This can be summed up as follows.

Super query Given $N/2$ adjacent query pairs to E all using the same key $K \in \{0,1\}^{2n}$, all the remaining $N/2$ queries using the same key K and the remaining $N/2$ queries using key $\overline{K}$ are given for free.

We say that an adjacent query pair $(M\|U, \widehat{U}, R), (\overline{M\|U}, \widehat{U}, S)$ is *successful*, if $\widehat{U} \oplus R = V$ and $\widehat{U} \oplus S = \widehat{V}$, or if $\widehat{U} \oplus R = \widehat{V}$ and $\widehat{U} \oplus S = V$. Thus the adversary obtains a preimage of $(V, \widehat{V})$ precisely if it obtains a successful adjacent query pair. This can occur in one of two ways: either the winning query pair is part of a super query, or not. We let $\mathsf{SuperQueryWin}(\mathcal{Q})$ denote the event that the adversary obtains a winning query pair that is part of a super query, and $\mathsf{ForwardQueryWin}(\mathcal{Q})$ the event that the adversary obtains a winning query pair of normal queries (either forward or backward). It thus suffices to upper bound

$$\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})] + \Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})].$$

Here, probabilities are taken (as usual) over the adversary's randomness (if any) and over the randomness of the ideal cipher.

We first upper bound $\Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})]$. Note that when the adversary makes, say, a forward query $E_{M\|U}(\widehat{U})$, at most $N/2-2$ queries (counting free queries) have been previously answered with the key $M\|U$, since otherwise a super query for the key $M\|U$ would have occurred. Thus the value $R = E_{M\|U}(\widehat{U})$ comes uniformly at random from a set of size at least $N/2+2 \geq N/2$, and there is chance at most $2/(N/2) = 4/N$ that either $\widehat{U} \oplus R = V$ or $\widehat{U} \oplus R = \widehat{V}$ (this is also true if $V = \widehat{V}$). If, say, $\widehat{U} \oplus R = V$, there is further chance at most $1/(N/2) = 2/N$ that the free query $E_{\overline{M\|U}}(\widehat{U})$ returns $\widehat{U} \oplus \widehat{V}$, since the answer to the free query comes uniformly at random from a set of size at least $N/2+1 > N/2$. Other cases (*e.g.*, when $\widehat{U} \oplus R = \widehat{V}$, and when the adversary makes a backward query $E^{-1}_{M\|U}(R)$) are similarly analyzed, showing that the adversary's chance of triggering the event $\mathsf{ForwardQueryWin}(\mathcal{Q})$ at any given query is at most $(4/N)(2/N) = 8/N^2$. Since the adversary makes q queries total, we therefore have

$$\Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})] \leq 8q/N^2. \tag{8.5}$$

We now bound $\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})]$. Assume that a super query is about to occur on keys $M\|U$ and $\overline{M\|U}$ meaning that the value of $E_{M\|U}(\cdot)$ and $E_{\overline{M\|U}}(\cdot)$ are already known on exactly $N/2$ points. Let us denote this set of points by $\mathcal{X}$ and let $\mathcal{Y} = E_{M\|U}(\mathcal{X})$ and $\mathcal{Y}' = E_{\overline{M\|U}}(\mathcal{X})$. Further, let $\mathcal{R} = \{0,1\}^n \backslash \mathcal{X}$, $\mathcal{S} = \{0,1\}^n \backslash \mathcal{Y}$, and $\mathcal{S}' = \{0,1\}^n \backslash \mathcal{Y}'$. Clearly, $|\mathcal{X}| = |\mathcal{Y}| = |\mathcal{Y}'| = |\mathcal{R}| = |\mathcal{S}| = |\mathcal{S}'|$.

Now fix a point $A \in \mathcal{R}$ in the domain of the super query. We now estimate the probability that this point A induces a successful pair. This can only be the case if

1. $A \oplus V \in \mathcal{S}$ and $A \oplus \widehat{V} \in \mathcal{S}'$ or
2. $A \oplus \widehat{V} \in \mathcal{S}$ and $A \oplus V \in \mathcal{S}'$.

The probability that $E_{M\|U}(A) = A \oplus V$ and $E_{\overline{M\|U}}(A) = A \oplus \widehat{V}$ equals $1/(N/2)^2$. The same is true for the probability that $E_{M\|U}(A) = A \oplus \widehat{V}$ and $E_{\overline{M\|U}}(A) = A \oplus V$. Thus the total probability to be successful in a super query is at most

$$2 \cdot N/2 \cdot \left(\frac{1}{N/2}\right)^2 = \frac{2}{N/2}.$$

Since at most $q/(N/2)$ super queries can ever occur, we have

$$\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})] \leq 8q/N^2. \tag{8.6}$$

The sum of (8.5) and (8.6) gives our claim. □

8.4. Abreast-DM

Theorem 8.4. *Let $H^{\mathrm{ADM}} : \{0,1\}^{3n} \to \{0,1\}^{2n}$ be the* Abreast-DM *compression function as depicted in Figure 8.2 and defined in Section 1.4 on Page 28. Let $\alpha > 0$ be an integer. Then*

$$\mathbf{Adv}^{\mathrm{epre}}_{H^{\mathrm{ADM}}}(q) \quad \leq \quad \frac{16\alpha}{N} + \frac{8q}{N^2(N-2)} + 2 \cdot \left(\frac{2eq}{\alpha N}\right)^{\alpha} + \frac{4q}{\alpha N}.$$

Proof. Let $(V, \widehat{V}) \in \{0,1\}^n \times \{0,1\}^n$ be the point to invert, chosen by the adversary before any queries are made to E.

Unlike in the proof for Hirose-DM, we do not give the adversary a free query after each query it makes. However, we still give

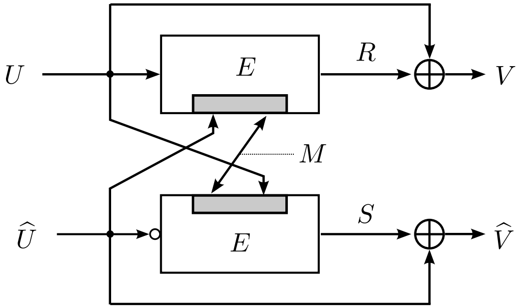


Figure 8.2.: The Abreast-DM compression function. The wires $U, \widehat{U}, M$ are the inputs to the compression function. The empty circle at the left side of the bottom blockcipher denotes bit complementation.

the adversary super queries for free. More precisely, whenever the adversary has made $N/2$ queries under a given key $K\|L$, and after the $(N/2)$-th such query has been answered and placed in the query history, we give the remaining $N/2$ queries under the key $K\|L$ for free to the adversary, in any order. In this case, we say that a super query occurs; every query in the query history is either part of a super query, or not. In the latter case we call the query a normal query. (Thus in this theorem, normal queries are exactly the non-free queries.) Unlike in the proof of Theorem 8.2, there is no notion of an adjacent query pair. However, like in the proof of Theorem 8.2, we alert the reader to the fact that a super query consists of a set of $N/2$ queries, whereas a normal query is a single query.

We define an event $\mathsf{Lucky}(\mathcal{Q})$ on the query history; $\mathsf{Lucky}(\mathcal{Q})$ occurs if

$$|\{(K\|L, X, Y) \in \mathcal{Q} : X \oplus Y = V\}| > 2\alpha,$$

or if

$$|\{(K\|L, X, Y) \in \mathcal{Q} : \overline{X} \oplus Y = \widehat{V}\}| > 2\alpha.$$

The adversary obtains a preimage of $(V, \widehat{V})$ precisely if it obtains queries of the form $(\widehat{U}\|M, U, R)$, $(M\|\widehat{U}, \overline{\widehat{U}}, S)$ such that $U \oplus R = V$ and $\widehat{U} \oplus S = \widehat{V}$, where $\overline{\widehat{U}}$ is the bit-by-bit complement of $\widehat{U}$. It is easy to check that these two queries must be distinct, otherwise one obtains the contradiction $\overline{\widehat{U}} = U = M = \widehat{U}$. We call two such queries a winning pair of queries. Note, of course, that the queries in a winning pair need not be adjacent in the query history. We speak of the first and second query in a winning pair referring to the order in which they appear in the query history.

Let $\mathsf{WinNormal}(\mathcal{Q})$ be the event that the adversary obtains a winning pair in which the second query is a normal query. We then let $\mathsf{WinSuper}_1(\mathcal{Q})$ be the event that the adversary obtains a winning pair in which the second query is part of a super query and the first is either normal or part of a super query, but is not part of the *same* super query as the second. Finally let $\mathsf{WinSuper}_2(\mathcal{Q})$ be the event that the adversary obtains a winning pair in which both queries of the pair are part of the same super query. It is then clear that if the adversary wins, one of the events

$$\mathsf{WinNormal}(\mathcal{Q}),\ \mathsf{WinSuper}_1(\mathcal{Q})\,,\ \text{ or }\ \mathsf{WinSuper}_2(\mathcal{Q})$$

occurs. In particular, thus, one of the four events

$$\mathsf{Lucky}(\mathcal{Q}),\ \mathsf{WinNormal}(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q}),\ \mathsf{WinSuper}_1(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q}),$$

or

$$\mathsf{WinSuper}_2(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})$$

must occur if the adversary wins. We upper bound the probability of each of these four events and sum the upper bounds in order to obtain an upper bound on the adversary's advantage.

We start by upper bounding $\Pr[\mathsf{Lucky}(\mathcal{Q})]$. For this we introduce two new events. Let $\mathcal{Q}_{\mathrm{n}}$ be the restriction of $\mathcal{Q}$ to normal queries, and let $\mathcal{Q}_{\mathrm{s}}$ be the restriction of $\mathcal{Q}$ to queries that are part of super queries. Let $\mathsf{Lucky}_{\mathrm{n}}(\mathcal{Q})$ be the event that either

$$|\{(K\|L, X, Y) \in \mathcal{Q}_{\mathrm{n}} : X \oplus Y = V\}| > \alpha,$$

or

$$|\{(K\|L, X, Y) \in \mathcal{Q}_{\mathrm{n}} : \overline{X} \oplus Y = \widehat{V}\}| > \alpha.$$

The event $\mathsf{Lucky}_{\mathrm{s}}(\mathcal{Q})$ is likewise defined with respect to $\mathcal{Q}_{\mathrm{s}}$. Obviously, $\mathsf{Lucky}(\mathcal{Q}) \implies \mathsf{Lucky}_{\mathrm{n}}(\mathcal{Q}) \vee \mathsf{Lucky}_{\mathrm{s}}(\mathcal{Q})$, so it suffices to upper bound $\mathsf{Lucky}_{\mathrm{n}}(\mathcal{Q})$ and $\mathsf{Lucky}_{\mathrm{s}}(\mathcal{Q})$ and to sum these upper bounds.

Since every answer to a normal query, forward or backward, comes at random from a set of size at least $N/2$, and since at most q normal queries are made, we have that

$$\Pr[\mathsf{Lucky}_{\mathrm{n}}(\mathcal{Q})] \le 2 \cdot \binom{q}{\alpha} \left(\frac{2}{N}\right)^{\alpha} \le 2 \cdot \left(\frac{2eq}{\alpha N}\right)^{\alpha}.$$

To upper bound $\Pr[\mathsf{Lucky}_{\mathrm{s}}(\mathcal{Q})]$, note that when a super query is made on key $K\|L$, the expected number of points $X \in \{0,1\}^n$ in the domain of the super query such that $X \oplus E_{K\|L}(X) = V$ is at most $(N/2) \cdot (2/N) = 1$, since for each such individual point the probability that $X \oplus E_{K\|L}(X) = V$ is either 0 (if $X \oplus V$ is not in the range of the super query) or $2/N$. Moreover, there occur at most $q/(N/2) = 2q/N$ super queries, since it costs $N/2$ queries to setup a super query for a given key. Thus the expectation of the random variable

$$|\{(K\|L, X, Y) \in \mathcal{Q}_{\mathrm{s}} : X \oplus Y = V\}|,$$

taken over the coin tosses of the adversary and the randomness of E, is at most $2q/N \cdot 1 = 2q/N$. It then follows by Markov's inequality that the probability that

$$|\{(K\|L, X, Y) \in \mathcal{Q}_{\mathrm{s}} : X \oplus Y = V\}| > \alpha$$

is at most $2q/\alpha N$. Then by a union bound and a symmetric argument (for $\overline{X} \oplus Y = \widehat{V}$), we obtain that $\Pr[\mathsf{Lucky}_{\mathrm{s}}(\mathcal{Q})] \le 4q/\alpha N$. Summing the upper bounds for $\Pr[\mathsf{Lucky}_{\mathrm{n}}(\mathcal{Q})]$ and $\Pr[\mathsf{Lucky}_{\mathrm{s}}(\mathcal{Q})]$, we thus obtain that

$$\Pr[\mathsf{Lucky}(\mathcal{Q})] \le 2 \cdot \left(\frac{2eq}{\alpha N}\right)^{\alpha} + \frac{4q}{\alpha N}. \tag{8.7}$$

We now upper bound $\Pr[\mathsf{WinNormal}(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})]$. For this we use a wish list argument similar to that of [107]. As the adversary makes queries, we maintain two sequences $\mathcal{W}_{\mathrm{T}}$ and $\mathcal{W}_{\mathrm{B}}$ called *wish lists*. These are initially empty. For each query $(K\|L, X, Y)$ added to the query history (whether normal or part of a super query) we update the wish lists as follows:

1. If $X \oplus Y = V$ then $(L\|X, \overline{K}, K \oplus \widehat{V})$ is added to $\mathcal{W}_{\mathrm{B}}$.

2. If $\overline{X} \oplus Y = \widehat{V}$ then $(\overline{X}\|K, L, L \oplus V)$ is added to $\mathcal{W}_{\mathrm{T}}$.

We emphasize that $\mathcal{W}_{\mathrm{B}}$ and $\mathcal{W}_{\mathrm{T}}$ are sequences, not sets. The following properties are easy to check: (i) a query never adds itself to a wish list (namely, the queries inserted into the wish lists – if any – as a result of query $(K\|L, X, Y)$ being added to the query history, are distinct from $(K\|L, X, Y)$ itself); (ii) the elements of $\mathcal{W}_{\mathrm{T}}$ are all distinct from one another, and the elements of $\mathcal{W}_{\mathrm{B}}$ are all distinct from one another—namely, the same triple is never added twice to a wish list; (iii) the adversary obtains a winning pair precisely if a query is ever added to its query history that is already a member of one of its wish lists before the updating of the wish lists for that query (by property (i), however, we could equally well say after the updating of the wish lists for that query). Moreover, as long as $\neg\mathsf{Lucky}(\mathcal{Q})$ holds, the wish lists never exceed length 2α.

Let $E_{K\|L}(X)$ be a query made to E during the adversary's attack (either a normal query, or as part of a super query). If, at the moment the query is being made, there is an element of the form $(K\|L, X, Y)$ in (at least) one of the wish lists for some $Y \in \{0,1\}^n$, then we say this wish list element is being 'wished for' when the query $E_{K\|L}(X)$ is made. We similarly say the wish list element $(K\|L, X, Y)$ is being wished for if the query $E^{-1}_{K\|L}(Y)$ is made (note that in this case, the query $E^{-1}_{K\|L}(Y)$ is necessarily normal, since a super query is, by default, implemented by forward queries). We note, importantly, that any wish list element can only be wished for once, since $E_{K\|L}(\cdot)$ is a permutation.

We now let $\mathsf{NormalWishGranted}_{\mathrm{T},i}$ be the event that a normal query $(K\|L, X, Y)$, when added to the query list, is equal to the i-th element of $\mathcal{W}_{\mathrm{T}}$ (presuming $\mathcal{W}_{\mathrm{T}}$ has length at least i when the query is added). Likewise define $\mathsf{NormalWishGranted}_{\mathrm{B},i}$ with respect to the list $\mathcal{W}_{\mathrm{B}}$. Then, by the above remarks,

$$\begin{aligned}\mathsf{WinNormal}(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q}) \implies & \bigvee_{i=1}^{2\alpha} \mathsf{NormalWishGranted}_{\mathrm{T},i} \\ & \vee \bigvee_{i=1}^{2\alpha} \mathsf{NormalWishGranted}_{\mathrm{B},i},\end{aligned}$$

so by a union bound

$$\begin{aligned}\Pr[\mathsf{WinNormal}(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})] \leq & \sum_{i=1}^{2\alpha} \Pr[\mathsf{NormalWishGranted}_{\mathrm{T},i}] + \\ & \sum_{i=1}^{2\alpha} \Pr[\mathsf{NormalWishGranted}_{\mathrm{B},i}].\end{aligned}$$

Because each wish list element can only be wished for once and because a normal query is answered at random uniformly from a set of size at least $N/2$, we have

$$\Pr[\mathsf{NormalWishGranted}_{\mathrm{T},i}] \leq 2/N, \Pr[\mathsf{NormalWishGranted}_{\mathrm{B},i}] \leq 2/N,$$

and therefore

$$\Pr[\mathsf{WinNormal}(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})] \leq 2 \cdot (4\alpha/N) = 8\alpha/N. \qquad (8.8)$$

We now upper bound $\Pr[\mathsf{WinSuper}_1(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})]$. We keep the same definition of the wish lists $\mathcal{W}_{\mathrm{T}}$ and $\mathcal{W}_{\mathrm{B}}$ as above. We let $\mathsf{SuperWishGranted}^1_{\mathrm{T},i}$ be the event that a query $(K\|L, X, Y)$ that is part of a super query is equal to the i-th element of $\mathcal{W}_{\mathrm{T}}$, where $\mathcal{W}_{\mathrm{T}}$ has length $\geq i$ before any of the super queries under key $K\|L$ have been made. The event $\mathsf{SuperWishGranted}^1_{\mathrm{B},i}$ is similarly defined. By the definition of $\mathsf{WinSuper}_1(\mathcal{Q})$ we have that

$$\begin{aligned}\Pr[\mathsf{WinSuper}_1(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})] \le \sum_{i=1}^{2\alpha} &\Pr[\mathsf{SuperWishGranted}^1_{\mathrm{T},i}] \\ &+ \sum_{i=1}^{2\alpha} \Pr[\mathsf{SuperWishGranted}^1_{\mathrm{B},i}].\end{aligned}$$

Assume, for a given i, that the i-th element of $\mathcal{W}_{\mathrm{T}}$ (say) is $(K\|L, X, Y)$, and that a super query is about to be made for the key $K\|L$, and that X is in the domain of the super query. Then the probability that $E_{K\|L}(X) = Y$ is at most $2/N$ (more precisely, it is exactly $2/N$ unless Y is not in the super query's range, in which case it is 0). Thus, arguing similarly for the list $\mathcal{W}_{\mathrm{B}}$, we obtain that

$$\Pr[\mathsf{SuperWishGranted}^1_{\mathrm{T},i}] \le 2/N, \qquad \Pr[\mathsf{SuperWishGranted}^1_{\mathrm{B},i}] \le 2/N.$$

Therefore,

$$\Pr[\mathsf{WinSuper}_1(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})] \le 8\alpha/N. \tag{8.9}$$

We finally bound $\Pr[\mathsf{WinSuper}_2(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})]$. Actually, we just upper bound $\Pr[\mathsf{WinSuper}_2(\mathcal{Q})]$, and do not use a wish list argument. Note the event $\mathsf{WinSuper}_2(\mathcal{Q})$ can only occur when a super query is made on a key of the form $L\|L$, and then occurs only if both L and $\overline{L}$ are in the domain of the super query and if $E_{L\|L}(L) \oplus L = V$ and $E_{L\|L}(\overline{L}) \oplus L = \widehat{V}$. It is easy to see that the probability (when the super query is made) that these latter equalities hold is at most $(2/N)\cdot(1/(N/2-1))$. Since at most $q/(N/2)$ super queries are made, we therefore have

$$\begin{aligned}\Pr[\mathsf{WinSuper}_2(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})] &\le \Pr[\mathsf{WinSuper}_2(\mathcal{Q})] \\ &\le 4q/N^2(N/2-1).\end{aligned} \tag{8.10}$$

Finally, we obtain the theorem by summing (8.7), (8.8), (8.9), and (8.10). □

Corollary 8.5. *We have*

$$\mathbf{Adv}^{\text{epre}}_{H^{\text{ADM}}}(2^{2n-10}) \quad \leq \quad 1/2 + o(1),$$

where the $o(1)$ *term tends to 0 as* $n \to \infty$*.*

Proof. By setting $\alpha = q^{1/2}/2$ (note that α is allowed to depend on q), the bound from Theorem 8.4 simplifies to

$$\frac{16q^{1/2}}{N} + \frac{8q}{N^2(N-2)} + 2 \cdot \left(\frac{4eq^{1/2}}{N}\right)^{q^{1/2}/2}.$$

Suppose that $q = (cN)^2$ for some $0 < c < 1$, then this bound can be rewritten as

$$16c + \frac{8c^2}{N-2} + 2 \cdot (4ec)^{cN/2}.$$

For $4ec < 1$ this tends $16c$, so setting $c = 1/32$ gives us the claimed result.

8.5. Tandem-DM

Our new bound for Tandem-DM is identical to the bound we gave for Abreast-DM, so in particular 2^{2n-10} queries are needed to obtain a preimage with probability $\sim 1/2$. Also, the proof works very much the same way (Corollary 8.7).

Theorem 8.6. *Let* $H^{\text{TDM}} : \{0,1\}^{3n} \to \{0,1\}^{2n}$ *be the* Tandem-DM *compression function as depicted in Figure 8.3 and defined in Section 1.4 on Page 28. Let* $\alpha > 0$ *be an integer. Then*

$$\mathbf{Adv}^{\text{epre}\neq 0}_{H^{\text{TDM}}}(q) \quad \leq \quad \frac{16\alpha}{N} + \frac{8q}{N^2(N-2)} + 2 \cdot \left(\frac{2eq}{\alpha N}\right)^{\alpha} + \frac{4q}{\alpha N}.$$

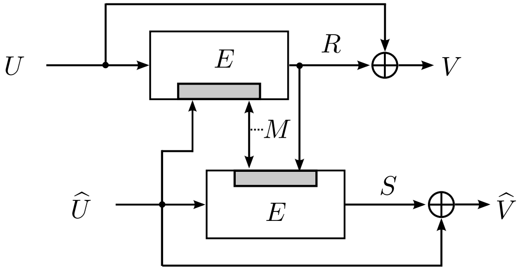


Figure 8.3.: The TANDEM-DM compression function. The wires $U, \widehat{U}, M$ are the inputs to the compression function.

Proof. Let $(V, \widehat{V}) \neq (0^n, 0^n)$ be the point to invert, chosen by the adversary before making any queries to E. We manage free queries exactly as for ABREAST-DM; more precisely, when $N/2$ queries are made to E under a given key, we give the remaining $N/2$ queries under that key for free to the adversary, and this constitutes a "super query". No other free queries are given.

In the case of TANDEM-DM, the adversary obtains a preimage of $(V, \widehat{V})$ precisely if it obtains queries of the form $(\widehat{U} \| M, U, R)$, $(M \| R, \widehat{U}, S)$ such that $U \oplus R = V$, $\widehat{U} \oplus S = \widehat{V}$. It is easy to see that these two queries must be distinct, otherwise we would have $U = \widehat{U} = M = R = S$ and therefore $(V, \widehat{V}) = (0^n, 0^n)$. We call two queries as above a "winning pair" of queries, where the two elements of a winning pair need not be adjacent in the query history (and could be in any order). We speak again of the "first" and "second" query in a winning pair referring to the order in which they appear in the query history.

We define the events $\mathsf{Lucky}(\mathcal{Q})$, $\mathsf{WinNormal}(\mathcal{Q})$, $\mathsf{WinSuper}_1(\mathcal{Q})$, and $\mathsf{WinSuper}_2(\mathcal{Q})$ as in the proof of Theorem 8.4 (but with respect, of course, to the new definition of "winning pair"). If the adversary wins, one of the events

$$\mathsf{Lucky}(\mathcal{Q}),\ \mathsf{WinNormal}(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q}),\ \mathsf{WinSuper}_1(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q}),$$

or

$$\mathsf{WinSuper}_2(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})$$

must occur. We upper bound the probability of each of these events separately.

As in the case of Theorem 8.4, we have

$$\Pr[\mathsf{Lucky}(\mathcal{Q})] \leq 2 \cdot \left(\frac{2eq}{\alpha N}\right)^{\alpha} + \frac{4q}{\alpha N}. \tag{8.11}$$

To upper bound $\Pr[\mathsf{WinNormal}(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})]$, we again use wish lists. There are two wish lists, $\mathcal{W}_{\mathrm{T}}$ and $\mathcal{W}_{\mathrm{B}}$, that are both initially empty and which are updated after each new query $(K\|L, X, Y)$ is placed into the query history, according to the following rules:

1. If $X \oplus Y = V$ then $(L\|Y, K, K \oplus \widehat{V})$ is added to $\mathcal{W}_{\mathrm{B}}$.

2. If $X \oplus Y = \widehat{V}$ then $(X\|K, L \oplus V, L)$ is added to $\mathcal{W}_{\mathrm{T}}$.

The same four properties from Theorem 8.4 are easy to check: (i) a query never "adds itself" to a wish list (this uses $(V, \widehat{V}) \neq (0^n, 0^n)$); (ii) the elements within each wish list are all distinct from one another; (iii) the adversary obtains a winning pair precisely if it obtains a query that is already in one of its wish lists (at the moment of insertion of that query into the query history). And, by definition of $\mathsf{Lucky}(\mathcal{Q})$, the wish lists never exceed length 2α as long $\neg\mathsf{Lucky}(\mathcal{Q})$ holds.

Let $\mathsf{NormalWishGranted}_{\mathrm{T},i}$ and $\mathsf{NormalWishGranted}_{\mathrm{B},i}$ be defined as in the proof of Theorem 8.4. Then, using exactly the same analysis as in the proof of Theorem 8.4, we have that

$$\Pr[\mathsf{NormalWishGranted}_{\mathrm{T},i}] \leq 2/N, \Pr[\mathsf{NormalWishGranted}_{\mathrm{B},i}] \leq 2/N,$$

and therefore

$$\Pr[\mathsf{WinNormal}(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})] \leq 8\alpha/N. \tag{8.12}$$

Then, also arguing word for word as in the proof of Theorem 8.4, we find that

$$\Pr[\mathsf{WinSuper}_1(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})] \leq 8\alpha/N. \tag{8.13}$$

We finally bound $\Pr[\mathsf{WinSuper}_2(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})]$. Note the event $\mathsf{WinSuper}_2(\mathcal{Q})$ can only occur when a super query occurs for a key of the form $(L\|L)$, and when that super query results in the triples $(V\oplus L, L\|L, L)$, $(L, L\|L, L\oplus\widehat{V})$ being added to the query history. The probability that $E_{L\|L}(V \oplus L) = L$ is at most $2/N$, and, conditioned on the event that $E_{L\|L}(V \oplus L) = L$, the probability that $E_{L\|L}(L) = L \oplus \widehat{V}$ is at most $1/(N/2-1)$. Since at most $2q/N$ super queries occur, we thus find that

$$\begin{aligned}\Pr[\mathsf{WinSuper}_2(\mathcal{Q}) \wedge \neg\mathsf{Lucky}(\mathcal{Q})] &\leq \Pr[\mathsf{WinSuper}_2(\mathcal{Q})] \\ &\leq 4q/N^2(N/2-1). \end{aligned}\tag{8.14}$$

The theorem follows by summing (8.11), (8.12), (8.13) and (8.14). □

As for Abreast-DM, we have the following corollary (with the same proof):

Corollary 8.7. *We have*

$$\mathbf{Adv}^{\mathrm{epre}\neq 0}_{H^{\mathrm{TDM}}}(2^{2n-10}) \quad \leq \quad 1/2 + o(1),$$

where the $o(1)$ term tends to 0 as $n \to \infty$.

8.6. Generalization

In this section, we generalize the result from Section 8.1. That is we state some general conditions such that the arguments given in the proof of Theorem 8.1 still apply. Note that this is somewhat related to our generic framework given in Section 1.4.1 and – even more – to the framework in [133].

Let $E, E' \in \textsc{Block}(k, n)$. We define a parallel-call compression[1] function $H^{\text{PAR}} : \{0,1\}^{k+n} \to \{0,1\}^{2n}$ by means of three helper functions $F, F' : \{0,1\}^{k+n} \to \{0,1\}^{k} \times \{0,1\}^{n}$, and $G : \{0,1\}^{k+n} \times \{0,1\}^{n} \times \{0,1\}^{n} \to \{0,1\}^{2n}$. If $U \in \{0,1\}^{k+n}$ is the input to the compression function, the digest V is computed as $(K, X) \leftarrow F(U), (K', X') \leftarrow F'(U)$, followed by $Y \leftarrow E_K(X), Y' \leftarrow E'_{K'}(X')$, and finally $V = G(U, Y, Y')$.

(Since two distinct block ciphers E and E' are used to create Y and Y', respectively, one obviously needs to use a slightly different definition of preimage resistance, where the adversary has access to both block ciphers and these are sampled independently from each other.)

Theorem 8.8. *Let H^{PAR} be a parallel-call compression function using two distinct block ciphers and with the following structural properties:*

- **P1** *F and F' are both bijections, thus defining a 1-1 correspondence π between inputs (K, X) and (K', X') (so $(K', X') = \pi(K, X)$).*
- **P2** *$G(U, Y, Y')$ as a function from (Y, Y') to V is bijective for all U.*
- **P3a** *G (combined with F) is such that K, Y and V uniquely determine a corresponding X. Namely, for all $\tilde{X}, Y'$, and U satisfying $G(U, Y, Y') = V$ and $(K, \tilde{X}) = F(U)$, we have $\tilde{X} = X$.*
- **P3b** *G (combined with F') is such that K', Y', and V uniquely determine a corresponding X'.*

[1] Actual compression requires $k > n$, however, our result is valid regardless.

P4 *F and F' are key-consistent, that is there exists a permutation ξ such that if $(K', X') = \pi(K, X)$ then $K' = \xi(K)$. (Here we assumed property* **P1** *is already satisfied.)*

Then

$$\mathbf{Adv}_H^{\mathrm{pre}}(q) \leq 8q/N^2.$$

In particular, an adversary must ask at least $N^2/16 = 2^{2n-4}$ queries to have chance of 1/2 for obtaining a collision.

Proof. Let V be the point that the adversary has chosen to invert (before gaining access to the block ciphers). Analogously to the proof of Theorem 8.1, the adversary gets on top of the q queries several queries for free, as follows:

Normal forward query If the adversary queries $E_K(X)$, we also give it for free $E'_{K'}(X')$, where $(K', X') = \pi(K, X)$ (and vice versa, *i.e.*, given a query $E'_{K'}(X')$ the appropriate $E_K(X)$ query with $(K, X) = \pi^{-1}(K', X')$ is added).

Normal backward query Given an backward query $E_K^{-1}(Y)$ with answer X, the corresponding query $E'_{K'}(X')$ is given for free (and vice versa, given an backward E' query, the appropriate E query is added).

Super query Given $N/2$ queries to E all using the same key K, all the remaining queries using that key K are given for free. Moreover, all remaining queries to E' using the corresponding key $K' = \xi(K)$ are given for free as well.

We also use the term "super query" to denote the *set* of $N/2$ free queries that are returned to the adversary. Every adjacent query pair in the query history is either part of a super query, or not. In the latter case, we call the adjacent query pair a "normal" adjacent query pair; we also say a single query is "normal" to indicate it is not part of a super query.[2] Moreover, we keep track in the query history (*e.g.*, by an additional bit of data attached to each query), of which queries are normal, and of which queries belong to a super query (with every query being one or the other and not both).

Thanks to property **P1**, any pair uniquely corresponds to some U (for which $(K, X) = F(U)$ and $(K', X') = F'(U)$) and we call it winning iff $H^{\mathrm{PAR}}(U) = V$, *i.e.* iff $V = G(U, Y, Y')$. To find a preimage, an adversary needs to create a winning pair in its query history (composed of both – the queries it asks explicitly and the free queries). We can distinguish between three types of winning pairs, depending on the free queries involved. Let $\mathsf{ForwardQueryWin}(\mathcal{Q})$ denote the event that the adversary obtains a winning pair that is not part of a super query and whose first query was a forward query. Let $\mathsf{BackwardQueryWin}(\mathcal{Q})$ denote the event that the winning pair is not part of a super query whose first query was an backward query, and finally, let $\mathsf{SuperQueryWin}(\mathcal{Q})$ denote the event that the adversary obtains a winning pair that is part of a super query. Then, the preimage-finding advantage is upper bounded by

$$\begin{aligned}\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})] + \Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})] \\ + \Pr[\mathsf{BackwardQueryWin}(\mathcal{Q})].\end{aligned}$$

We bound these three probabilities; a simple sum then leads to the theorem claim.

To bound $\Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})]$, we first consider a pair of queries $E_K(X), E'_{K'}(X')$. Property **P2** implies that only a single pair Y_1, Y_2 can lead to a preimage for V. Since there have been at most $N/2$ queries to E under K and, similarly, at most $N/2$ queries to E' under K', the probability of hitting this single value (Y_1, Y_2) is upper bounded by $1/(N - N/2)^2 = 4/N^2$ (here we also use that E and E' are independent).

[2] We point out the following discrepancy, which might otherwise cause confusion: a "super query" is a set of $N/2$ queries in the query history; but a "normal query" is a single query in the query history.

Bounding $\Pr[\mathsf{BackwardQueryWin}(\mathcal{Q})]$ goes along similar lines. Assume *wlog.* that the pair has been received by query $E_K^{-1}(Y)$. Then, property **P3a** implies that there exists a unique X that might lead to a preimage. Moreover (regardless of the actual $\tilde{X}$ obtained), there is a unique answer Y' to $E'_{K'}(X')$ (where $(K', X') = \pi(K, \tilde{X})$) that combines with K, X, and Y to complete the preimage. Since there have been at most $N/2$ queries to E under K and, similarly, at most $N/2$ queries to E' under K', the probability of hitting both X and Y' is upper bounded by $1/(N - N/2)^2 = 4/N^2$ (again exploiting the independence of E and E'). The case that the pair was received by an $(E')_{K'}^{-1}(Y')$ is analogous, based on property **P3b**.

Note that a query is either a forward or an backward query (but not both), so by an union bound, the (combined) total contribution of these two events to the adversary's advantage is at most $4q/N^2$.

It remains bounding $\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})]$. We first note that key consistency (property **P4**) ensures that only "regular" queries can count towards causing a super query. Since the threshold is $N/2$, this immediately implies that an adversary can only ever cause $2q/N$ super queries. For any individual super query, we claim that the success probability is upper bounded by $2/N$, *i.e.*, $\Pr[\mathsf{SuperQueryWin}(\mathcal{Q})] \leq 4q/N^2$. A super query consists of $N/2$ query pairs. For each pair, we can use the exact same derivation as for $\Pr[\mathsf{ForwardQueryWin}(\mathcal{Q})]$ to argue that it succeeds with probability at most $4/N^2$. A union bound over the $N/2$ pairs constituting a super query gives the claimed $2/N$ bound. □

Part III.

On Ideal World Models for Hash Functions

9 Results Summary

9.1. Introduction

The results of the third part of this thesis were published in [62]. Here, we take in a broader perspective. Until now, we tried to build good compression functions from 'ideal' ciphers inside. Now, we discuss the problem of building a 'good' hash function from 'ideal' compression functions. But instead of analyzing standard-model properties as collision security or preimage security separately, we aim for a 'perfect' hash function, namely, we want it to behave like a random oracle. For this, we call a hash function *secure* if it is *indifferentiable from a random oracle* (a formal definition is given in Section 10.1).

One purpose of our presentation is to inspire a discussion about the practical relevance of the notion *secure*. More precisely: Does a hash function that is not *secure* actually indicate a structural flaw in a hash function, whereas being *secure* guarantees the absence of them?

So taking in a practical point of view, we examine to what extent a structure of a hash function, proven secure using the indifferentiability framework, relates with instantiations satisfying this structure. This perspective is justified by the objective of Coron *et al.* [30] to deliver a criteria for the design of practical hash functions that can distinguish between good hash structures and bad hash structures. On a merely abstract level – *i.e.*, if one views a hash function as a sole random oracle – the hash structure is trivially secure. Instantiated as one single collision resistant hash function, it is trivially insecure. We examine what happens in between these two poles. We show that one is able to prove one and the same practical hash function secure and insecure at the same time. These hash functions *do not* differ in their implementations but only on the level of abstraction. Also, we show how a slight modification to a secure hash function, *e.g.*, a post-processing using some one way function, can drive it insecure, whereas a post-processing using an easily invertible function apparently preserves its security. We are able to derive some weird features that a secure hash function must offer. Moreover, as we can prove different structures that correspond to one and the same instantiated hash function secure and insecure, we are faced with an open problem what to conclude for the security of the practical hash function.

Section 9.2 gives a detailed outline and further motivates this discussion. Section 10.1 introduces the concept of "indifferentiability from a random oracle" as a security notion for hash functions and compares it to other known ideal world security models for hash functions. In Section 10.5 we derive some mandatory design principles for hash functions being secure in the indifferentiability framework. In Section 10.6 we discuss our findings in greater detail.

9.2. (In)Security in the Indifferentiability World

In the following sections we examine various constructions that are secure in the indifferentiability framework (details on indifferentiability follow in Section 10.1) involving one or more random oracles and show how slight modifications to them (or partial instantiations) drive them insecure (at least in this framework).

We now motivate our research and summarize some of our results in Table 9.1. Furthermore, we give a short example in which way our results correlate with the design of practical hash functions.

Section	**Secure**	**Insecure** (partial instantiation or modification)	**Insecure** (extension)	**Secure**
10.2	$\mathcal{RO}$	X	$\mathcal{RO} \circ X$ $X \circ \mathcal{RO}$	$\mathcal{RO} \circ W$ $W \circ \mathcal{RO}$
10.2	$\mathcal{RO} \circ \mathcal{RO}$	$\mathcal{RO} \circ X$ $X \circ \mathcal{RO}$	$\mathcal{RO} \circ \mathcal{RO} \circ X$ $X \circ \mathcal{RO} \circ \mathcal{RO}$	$\mathcal{RO} \circ \mathcal{RO} \circ W$ $W \circ \mathcal{RO} \circ \mathcal{RO}$
10.3	$\mathcal{RO} \circ MD_{\mathcal{RO}}$ (NMAC)	$\mathcal{RO} \circ MD_Z$ $X \circ MD_{\mathcal{RO}}$	$\mathcal{RO} \circ MD_{\mathcal{RO}} \circ X$ $X \circ \mathcal{RO} \circ MD_{\mathcal{RO}}$	$\mathcal{RO} \circ MD_{\mathcal{RO}} \circ W$ $W \circ \mathcal{RO} \circ MD_{\mathcal{RO}}$
10.4	$\mathcal{RO}_i \circ X \circ \mathcal{RO}_i$ (MCM)	$\mathcal{RO}_x \circ X \circ Y$ $Y \circ X \circ \mathcal{RO}_x$ $X \circ \mathcal{RO}_x \circ Y$	$\mathcal{RO}_i \circ X \circ \mathcal{RO}_i \circ Y$ $Y \circ \mathcal{RO}_i \circ X \circ \mathcal{RO}_i$	$\mathcal{RO}_i \circ X \circ \mathcal{RO}_i \circ W$ $W \circ \mathcal{RO}_i \circ X \circ \mathcal{RO}_i$

Table 9.1.: $\mathcal{RO}$ denotes a random oracle (with fixed or variable length input), $\mathcal{RO}_i$ an injective random 'oracle', $\mathcal{RO}_x$ a random oracle ($\mathcal{RO}_x$ is a fixed or variable input length, injective or not, random oracle), X,Y, and Z collision resistant one-way functions (CROWF), W is an easily invertible function, MD_R is a Merkle-Damgård iteration with R being used as the compression function.

9.3. Motivational Example

Say we want to design a secure hash function and come up with the idea to design our hash function as a concatenation of a pre-processing function modeled as a random oracle $\mathcal{RO}$ and a collision resistant one way function (CROWF) X. Consequently, our hash function $\mathcal{H}$ for a message M is

$$\mathcal{H}(M) := (\mathcal{RO} \circ X)(M).$$

So we try to proof its security in the indifferentiability framework and come to the conclusion that this hash function is in fact insecure (refer to Theorem 10.3 (iii)). In the indifferentiability framework we have at least three straightforward approaches to get $\mathcal{H}$ secure:

1. Remove the CROWF X: $\mathcal{H}_1(M) = (\mathcal{RO})(M)$.
2. "Strengthen" X and make it a random oracle: $\mathcal{H}_2(M) = (\mathcal{RO} \circ \mathcal{RO})(M)$.
3. "Weaken" X and make it an easily invertible function W: $\mathcal{H}_3(M) = (\mathcal{RO} \circ W)(M)$.

The hash functions $\mathcal{H}_1, \mathcal{H}_2$, and $\mathcal{H}_3$ can be proven secure in the indifferentiability framework (see Theorem 10.3 (i) and (ii)).

Indifferentiability was devised as a tool to see subtle real world weaknesses while in the random oracle world. But we can prove $\mathcal{H}$ insecure and $\mathcal{H}_3$ secure. In the real world (*i.e.*, comparing the instantiated hash functions) $\mathcal{H}_3$ is (almost) sure to be substantially weaker than $\mathcal{H}$. Additionally, the hash functions $\mathcal{H}$ and $\mathcal{H}_2$ could be implemented using exact the same lines of code but one is proved to be insecure, the other one seems to be secure. What shall we conclude for the security of our instantiated hash function in the real world? How can we conclude that $\mathcal{H}$ has some real world weaknesses that $\mathcal{H}_2$ has not. Note that mixing complexity-theoretic and ideal building blocks is common and can, *e.g.*, be found in [140].

10 Ambiguous Security Recommendations

10.1. Preliminaries

For hash functions a random oracle serves as a reference model. It offers all the properties a hash function should have. This section gives an overview on all 'known methods' for comparing a hash function with a random oracle: indifferentiability and three weaker models: preimage awareness, indifferentiability from a public-use random oracle and indistinguishability.

Extending our initial discussion at the beginning of Section 1.1, we let $\mathcal{RO}$ denote a random oracle, that takes as input binary strings of any length and returns for each input a random infinite string, *i.e.*, it is a map $\mathcal{RO} : \mathbb{Z}_2^* \to \mathbb{Z}_2^\infty$, chosen by selecting each bit of $\mathcal{RO}(x)$ uniformly and independently, for every x. As in [30] we will only consider random oracles $\mathcal{RO}$ truncated to a fixed output length $\mathcal{RO} : \mathbb{Z}_2^* \to \mathbb{Z}_2^n$.

Indifferentiability from a Random Oracle. The indifferentiability framework was introduced by Maurer et al. in [118] and is an extension to the classical notion of indistinguishability. Coron et al. [30] applied it to iterated hash function constructions and demonstrated for several iterated hash function constructions that they are indifferentiable from a random oracle if the compression function is a fixed input length (FIL) random oracle. Here, we give a brief introduction on these topics. For a more in-depth treatment, we refer to the original papers. In the context of iterated hashing, the adversary – called distinguisher D – shall distinguish between two systems as illustrated in Figure 10.1.

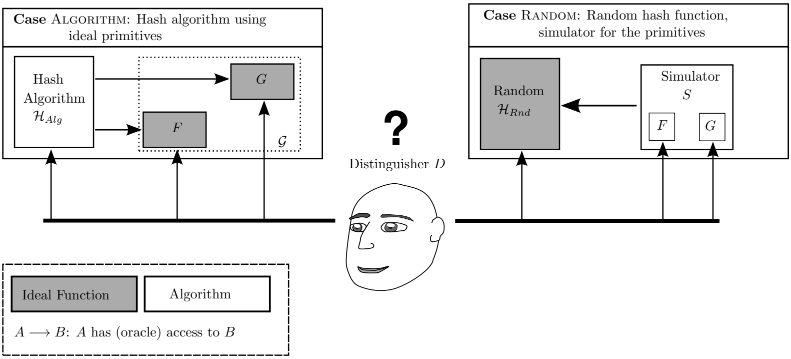

Case ALGORITHM: Hash algorithm using ideal primitives
Hash Algorithm $\mathcal{H}_{Alg}$
G
F
$\mathcal{G}$
Case RANDOM: Random hash function, simulator for the primitives
Random $\mathcal{H}_{Rnd}$
Simulator S
F
G
?
Distinguisher D
Ideal Function
Algorithm
$A \longrightarrow B$: A has (oracle) access to B

Figure 10.1.: Defining $\mathcal{H}_{Alg}$ being indifferentiable from a random oracle $\mathcal{H}_{Rnd} := \mathcal{RO}$

The system on the left (Case ALGORITHM) is the hash algorithm $\mathcal{H}_{Alg}$ using some ideal components (*i.e.*, FIL random oracles) contained in the set $\mathcal{G}$. The adversary can make queries to $\mathcal{H}_{Alg}$ as well as to the functions contained in $\mathcal{G}$. The system on the right consists of a random oracle (with truncated output) $\mathcal{H}_{Rnd} := \mathcal{RO}$ providing the same interface as the system on the left. To be indifferentiable to the system on the left, the system on the right (Case RANDOM) also needs a subsystem offering the same interface to the adversary as the ideal compression functions contained in $\mathcal{G}$. A simulator S is needed and its task is to simulate the ideal compression functions so that no distinguisher can tell whether it is interacting with the system on the left or with the one on the right. The output of S should look consistent with what the distinguisher can obtain from the random oracle $\mathcal{H}_{Rnd}$. In order to achieve that, the simulator can query the random oracle $\mathcal{H}_{Rnd}$. Note that the simulator does not see the distinguisher's queries to the random oracle. Formally, the indifferentiability of $\mathcal{H}_{Alg}$ from a random oracle $\mathcal{H}_{Rnd}$ is satisfied if:

Definition 10.1. *[30] A Turing machine $\mathcal{H}_{Alg}$ with oracle access to a set of ideal primitives contained in the set $\mathcal{G}$ is said to be (t_D, t_S, q, ϵ) indifferentiable from an ideal primitive $\mathcal{H}_{Rnd}$, if there exists a simulator S, such that for any distinguisher D it holds that:*

$$|Pr[D^{\mathcal{H}_{Alg},\mathcal{G}} = 1] - Pr[D^{\mathcal{H}_{Rnd},S} = 1]| < \epsilon.$$

The simulator has oracle access to $\mathcal{H}_{Rnd}$ and runs in time at most t_S. The distinguisher runs in time at most t_D and makes at most q queries. Similarly, $\mathcal{H}_{Alg}$ is said to be indifferentiable from $\mathcal{H}_{Rnd}$ if ϵ is a negligible function of the security parameter k (for polynomial bounded t_D and t_S).

Now, it is shown in [118] that if $\mathcal{H}_{Alg}$ is indifferentiable from a random oracle, then $\mathcal{H}_{Alg}$ can replace the random oracle in any scheme, and the resulting scheme is at least as secure in the ideal compression function model (*i.e.*, case ALGORITHM) as in the random oracle model (*i.e.*, case RANDOM).

'Non-Optimal' Ideal World Models. In [37], Dodis *et al.* have presented two ideal world security models that are strictly weaker than indifferentiability: *preimage awareness* and *indifferentiability from a public-use random oracle.* But both model a hash function fairly inadequate. A function that is preimage aware is not guaranteed to be secure against such trivial attacks as, *e.g.*, the Merkle-Damgård length-extension attack. And a function that is indifferentiable from a public-use random oracle has to ’publish’ any oracle query and might be only of limited use in the context of some signature schemes.

If a hash function is indistinguishable from a random oracle, an attacker that can query $\mathcal{H}_{Alg}$ – but has no access to the compression functions contained in $\mathcal{G}$ – cannot distinguish it from a random oracle. For hash function constructions, indistinguishability makes little sense as, for any concrete hash function, the compression functions in $\mathcal{G}$ are public and hence accessible to the adversary. As opposed to blockcipher constructions, there is no secret key or any other information the attacker has not. For them, being indistinguishable from a random permutation seems to suffice (at least in the ideal cipher model).

Therefore, we will focus in this work on ’indifferentiability from a random oracle’ since this seems to be the only security model known that is applicable in all contexts of cryptographic hash functions. The (open) challenge is to find an ideal world security model that is strong enough to defeat all known attacks but it should not be so strong that it leads to real world ambiguities. As we will show, the notion of indifferentiability has such ambiguities, namely we can prove one and the same real world hash function secure and insecure at the same time.

Security definitions that are based on a random oracle. Note that by assuming *ideal primitives* even in the Algorithm case, this definition is inherently based on the random oracle model. In the standard model we cannot assume ideal primitives (at least not without allowing an exponentially-sized memory to store a description of the function), so this notion of security only makes sense in the random oracle model.

Nevertheless, as we understand [30], a part of their motivation was to introduce a formalism for aiding the design of practical hash functions. Showing the above kind of “security” in the random oracle model ought to indicate the absence of structural flaws in the hash function.

On the other hand, if one can efficiently differentiate a hash function (using ideal primitives) from a random oracle, this appears to indicate a weakness in the hash function structure. With this reasoning, we again follow the example of Coron *et al.*, who debunk certain hash function structures as insecure by pointing out efficient differentiation attacks [30, Sections 3.1 and 3.2].

10.2. Composition

We start by investigating a rather simple construction to decide on the security of a hash function where a pre-processing or post-processing function is available.

Definition 10.2. *Let*

$$F^{(*\to n)}, G^{(*\to n)} : \{0,1\}^* \to \{0,1\}^n \textit{ and}$$
$$F^{(n\to n)}, G^{(n\to n)} : \{0,1\}^n \to \{0,1\}^n$$

be random oracles. A subscript i denotes an injective function/oracle whereas a subscript x denotes that we explicitly do not care whether the function/oracle is injective or not. Let

$$P^{(*\to n)}, Q^{(*\to n)} : \{0,1\}^* \to \{0,1\}^n \textit{ and}$$
$$P^{(n\to n)}, Q^{(n\to n)} : \{0,1\}^n \to \{0,1\}^n$$

be collision resistant one way functions. Let

$$W^{(n\to n)} : \{0,1\}^n \to \{0,1\}^n$$

be a function that is easily invertible, i.e. *there exists an polynomial algorithm $W^{-1} : \{0,1\}^n \to \{0,1\}^n$ such that $W^{-1}(W(A)) = A$ holds for any $A \in \{0,1\}^n$.*

(i) The hash function $H_{\mathcal{RO}\circ\mathcal{RO}} : \{0,1\}^* \to \{0,1\}^n$ *for a message* $M \in \{0,1\}^*$ *is defined by*

$$H_{\mathcal{RO}\circ\mathcal{RO}}(M) := G^{(n\to n)}(F^{(*\to n)}(M)).$$

(ii) Modification/Partial instantiation I: The hash function $H_{\mathcal{RO}\circ X} : \{0,1\}^* \to \{0,1\}^n$ *for a message* $M \in \{0,1\}^*$ *is defined by*

$$H_{\mathcal{RO}\circ X}(M) := F^{(n\to n)}(P^{(*\to n)}(M)).$$

(iii) Modification/Partial instantiation II: The hash function $H_{X\circ\mathcal{RO}} : \{0,1\}^* \to \{0,1\}^n$ *for a message* $M \in \{0,1\}^*$ *is defined by*

$$H_{X\circ\mathcal{RO}}(M) := P^{(n\to n)}(F^{(*\to n)}(M)).$$

(iv) Extension I: The hash function $H_{\mathcal{RO}\circ\mathcal{RO}\circ X} : \{0,1\}^* \to \{0,1\}^n$ *for a message* $M \in \{0,1\}^*$ *is defined by*

$$H_{\mathcal{RO}\circ\mathcal{RO}\circ X}(M) := F^{(n\to n)}(G^{(n\to n)}(P^{(*\to n)}(M))).$$

(v) Extension II: The hash function $H_{X\circ\mathcal{RO}\circ\mathcal{RO}} : \{0,1\}^* \to \{0,1\}^n$ *for a message* $M \in \{0,1\}^*$ *is defined by*

$$H_{X\circ\mathcal{RO}\circ\mathcal{RO}}(M) := P^{(n\to n)}(F^{(n\to n)}(G^{(*\to n)}(M))).$$

Theorem 10.3. *In the indifferentiability framework the following statements must hold:*

(i) $H_{\mathcal{RO}\circ\mathcal{RO}}$ *is secure (i.e., indifferentiable from a random oracle),*

(ii) $H_{\mathcal{RO}\circ X}$ is insecure (i.e., differentiable from a random oracle),

(iii) $H_{X\circ\mathcal{RO}}$ is insecure,

(iv) $H_{\mathcal{RO}\circ\mathcal{RO}\circ X}$ is insecure,

(v) $H_{X\circ\mathcal{RO}\circ\mathcal{RO}}$ is insecure.

Recall that for proving a hash function insecure we have to state an efficient distinguisher which can decide with non-negligible probability if the hash function is an algorithm utilizing at least one random oracle (the ALGORITHM case) or is a random oracle by itself (the RANDOM case).

Proof. Let H denote the hash oracle.

(i) Let $H := (\mathcal{RO}_2 \circ \mathcal{RO}_1)(M)$ be the definition of the hash function. We have to describe an efficient simulator S who is able to simulate the random oracles $\mathcal{RO}_1$ and $\mathcal{RO}_2$. The simulator has access to the hash oracle $H_{\mathcal{RO}}$.
Description of the Simulator S:

1. $\mathcal{RO}_1$ oracle queries: For all queries we keep a record. If we have answered the same query before we return the same value again. Else we choose a random value and add it to our database $DB \overset{add}{\longleftarrow} [query, random]$.
2. $\mathcal{RO}_2$ oracle queries: If $[?, query] \in DB$, then use the first entry to ask the hash oracle $H_{\mathcal{RO}}$ and return the answer. Else choose a random value and add it to our database $DB \overset{add}{\longleftarrow} [random, query]$. Use the new chosen random value to ask the hash oracle $H_{\mathcal{RO}}$ and return the answer.

Clearly, S is efficient. Furthermore, any distinguisher D cannot differentiate it from a random oracle, since the distribution of the outputs are the same in both cases.

Remark: The proof can be easily generalized to all functions of type $H_{\mathcal{RO}\circ\cdots\circ\mathcal{RO}}(M)$.

(ii) This result is essentially equivalent to the Coron et al. insecurity result regarding the composition of a CROWF with a random oracle [30], but we state a version of it here for completeness. We describe a distinguisher D to win this game, regardless of the simulator S:

1. Choose a random message $M \in \{0,1\}^*$.
2. Compute $u = P(M)$.
3. Ask the F-oracle for $v = F(u)$.
4. Ask the hash-oracle for $z = H(M)$.
5. If $z = v$ output ALGORITHM, else output RANDOM.

Analysis: Clearly, D is efficient. In the ALGORITHM world we always have

$$z = H(M) = F(P(M)) = F(u) = v,$$

so it always outputs 'ALGORITHM' if it interacts with the algorithm and the ideal primitive F.
A simulator trying to fool the distinguisher D does not know M as he receives the $F(u)$-oracle call. In order to answer correctly he has to come up with $M = P^{-1}(u)$ to ask the hash oracle for $H(M)$. As P is a CROWF this is not possible. Any such simulator can be used to invert P since in order to answer the $F(u)$ query correctly, it must be able to compute $v' = P^{-1}(u)$ and then return the value $H(v')$ to the adversary.

Furthermore, it is information-theoretically impossible to recover $M \in \{0,1\}^*$ from u.

(iii) Again, we describe a distinguisher D to win this game, regardless of the simulator S:

1. Choose a random message $M \in \{0,1\}^*$.

2. Ask the hash-oracle for $z = H(M)$.
3. Ask the F-oracle for $u = F(M)$.
4. Compute $v = P(u)$.
5. If $z = v$ output ALGORITHM, else output RANDOM.

Analysis: Again, D is obviously efficient and always outputs ALGORITHM in the algorithm world as we have

$$z = H(M) = P(F(M)) = P(u) = v.$$

In the random world, D learns a random target z and needs to find u with $z = P(u)$ whereas $u = F(M)$. So any simulator able to answer the F-oracle correctly can be used to invert P which is a CROWF.

(iv) The proof is essentially the same as in (ii).

(v) The proof is essentially the same as in (iii).

□

Paradox.

As we have proven in (ii) and (iii) the hash functions $H_{\mathcal{RO} \circ X}$ and $H_{X \circ \mathcal{RO}}$ are in fact insecure, if X is a collision resistant one way function. What happens if we substitute that CROWF with an easily invertible function? It turns out that both hash functions get secure again. Taking preimage resistance as an example, we are not able to attach an 'additional line of defense' (namely the function X) for preimage attacks, without losing the property of being 'indifferentiable from a random oracle'. Note that we do *not* point out any paradox in the indifferentiability framework itself. But, indeed, we do show several ambiguities we inevitably have to face if we try to apply this framework as a guide for designing secure and practical hash functions. This situations occur if we try to decrease the size of the gap between a practical hash function and its ideal world mapping.

Theorem 10.4. *Using the same notations as in Definition 10.2 it must hold in the indifferentiability framework:*

(i) The hash function

$$H_{\mathcal{RO}\circ W}(M) := F^{(n\to n)}(W^{(*\to n)}(M)),$$

is secure if W is an invertible function.

(ii) The hash function

$$H_{W\circ\mathcal{RO}}(M) := W^{(n\to n)}(F^{(*\to n)}(M)),$$

is secure if W is an invertible function.

Proof. Let H denote the hash oracle.

(i) Note that the function $W : \{0,1\}^* \to \{0,1\}^n$ is unlikely to be uniquely invertible in practice (normally, it will be information-theoretically impossible). But if we assume W's invertibility we can easily give a simulator S thus proving the hash function secure.

Description of Simulator S:

- F-oracle queries (parameter A): As we can invert W we can easily calculate $M = W^{-1}(A)$ and ask the hash oracle $v = H(M)$ and return v to the caller.

(ii) Now we have the function $W : \{0,1\}^n \to \{0,1\}^n$. Again, we can give a simple simulator S. *Description of Simulator S:*

- F-oracle queries (parameter M): Ask the hash oracle $z = H(M)$ and return $W^{-1}(z)$ to the caller.

□

Recall the example of Section 9.2. Let us start with the hash function $H_{\mathcal{RO}\circ X}$. For X being a CROWF we have shown $H_{\mathcal{RO}\circ X}$ to be insecure. If we *strengthen* X to be a random oracle our hash function gets secure. If we weaken X, *i.e.*, let X be an invertible function, our hash function gets secure again. The same is true if we begin our discussion with $H_{X\circ\mathcal{RO}}$.

Furthermore, all the theoretical hash functions $H_{\mathcal{RO}\circ\mathcal{RO}}$, $H_{X\circ\mathcal{RO}}$, and $H_{\mathcal{RO}\circ X}$ are a valid model for the same practical hash function, employing two functions F and G and defined by $H(x) = F(G(x))$. Should we conclude that this construction is secure, since we can prove their security if we model both F and G as random oracles? Or should we conclude that this construction is insecure, as we can disprove security in two other cases?

Insecurity by (partial) Instantiation.

Another point of view of the problem is given here. We start with the proven-to-be-secure hash function $H_{\mathcal{RO}\circ\mathcal{RO}}$. So one should assume that there are no structural weaknesses found in our construction. In order to get a practical hash function we have to instantiate the random oracle by efficient collision resistant one way functions. Instead of instantiating both random oracles at the same time we choose to instantiate them one after the other. Our intermediate result is either $H_{X\circ\mathcal{RO}}$ or $H_{\mathcal{RO}\circ X}$. Both of them were proved the be insecure in Theorem 10.3 – but formally we still have not left the random oracle world. Informally, we start now with an insecure hash function and instantiate the other random oracle. Thus, regarding the structural soundness of our construction, we get entirely contradicting messages again.

10.3. NMAC

Definition 10.5 (NMAC-Hash). *Let*

$$C^{(n+m\to n)} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n \text{ and}$$
$$D^{(n\to n)} : \{0,1\}^n \to \{0,1\}^n$$

be oracles, $M = (M_1, M_2 \ldots, M_L) \in (\{0,1\}^m)^L$ *be a padded message, and* $H_0 \in \{0,1\}^n$ *an arbitrary initial value. The Merkle-Damgård hash function* $MD_C : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$ *is defined by*

$$H_1 = C(H_0, M_1), \ldots, H_L = C(H_{L-1}, M_L),$$

$$MD_C(H_0, M_1, \ldots, M_L) = H_L.$$

The hash function $NMAC_{C,D} : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$ *is defined for fixed* $H_0 \in \{0,1\}^n$ *as follows:*

$$NMAC_{C,D}(M_1, \ldots, M_L) = D(MD_C(H_0, M_1, \ldots, M_L)).$$

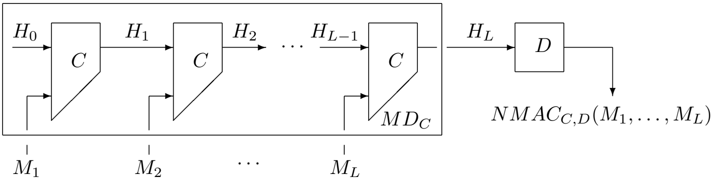


Figure 10.2.: The NMAC-Hash $NMAC_{C,D}$ – an extension to the plain Merkle-Damgård hash function MD_C.

Definition 10.6. *Let F, G, P, and Q be as in Definition 10.2. Additionally, let*

$$F^{(n+m\to n)} : \{0,1\}^{n+m} \to \{0,1\}^{n}$$

be a random oracle.

(i) The hash function $H_{\mathcal{RO}\circ MD_{\mathcal{RO}}} : \{0,1\}^{m\cdot L} \to \{0,1\}^{n}$ for a padded message $M \in \{0,1\}^{m\cdot L}$ is defined by

$$\begin{aligned} H_{\mathcal{RO}\circ MD_{\mathcal{RO}}}(M) &= NMAC_{F^{(n+m\to n)},G^{(n\to n)}}(M) \\ &= G^{(n\to n)}(MD_{F^{(n+m\to n)}}(M)). \end{aligned}$$

(ii) Modification/Partial instantiation I: The hash function $H_{\mathcal{RO}\circ MD_{X}} : \{0,1\}^{m\cdot L} \to \{0,1\}^{n}$ for a padded message $M \in \{0,1\}^{m\cdot L}$ is defined by

$$\begin{aligned} H_{\mathcal{RO}\circ MD_{X}}(M) &= NMAC_{P^{(n+m\to n)},F^{(n\to n)}}(M) \\ &= F^{(n\to n)}(MD_{P^{(n+m\to n)}}(M)). \end{aligned}$$

(iii) Modification/Partial instantiation II: The hash function $H_{X\circ MD_{\mathcal{RO}}} : \{0,1\}^{m\cdot L} \to \{0,1\}^{n}$ for a padded message $M \in \{0,1\}^{m\cdot L}$ is defined by

$$\begin{aligned} H_{X\circ MD_{\mathcal{RO}}}(M) &= NMAC_{F^{(n+m\to n)},P^{(n\to n)}}(M) \\ &= P^{(n\to n)}(MD_{F^{(n+m\to n)}}(M)). \end{aligned}$$

(iv) Extension I: Let

$$R^{(*\to m\cdot L)} : \{0,1\}^{*} \to \{0,1\}^{m\cdot L}$$

be a padding function. The hash function $H_{\mathcal{RO}\circ MD_{\mathcal{RO}}\circ R} : \{0,1\}^{*} \to \{0,1\}^{n}$ for an (unpadded) message $M \in \{0,1\}^{*}$ is defined by

$$H_{\mathcal{RO}\circ MD_{\mathcal{RO}}\circ X}(M) = NMAC_{F^{(n+m\to n)},G^{(n\to n)}}(R^{(*\to m\cdot L)}(M)).$$

(v) *Extension II: The hash function* $H_{X\circ\mathcal{RO}\circ MD_{\mathcal{RO}}} : \{0,1\}^{m\cdot L} \rightarrow \{0,1\}^n$ *for a padded message* $M \in \{0,1\}^{m\cdot L}$ *is defined by*

$$H_{X\circ\mathcal{RO}\circ MD_{\mathcal{RO}}}(M) = P^{(n\rightarrow n)}(NMAC_{F^{(n+m\rightarrow n)},G^{(n\rightarrow n)}}(M)).$$

We now proceed to give a proof for the following theorem.

Theorem 10.7. *Using the indifferentiability framework it must hold:*

(i) $H_{\mathcal{RO}\circ MD_{\mathcal{RO}}}$ *is secure,*

(ii) $H_{\mathcal{RO}\circ MD_X}$ *is insecure,*

(iii) $H_{X\circ MD_{\mathcal{RO}}}$ *is insecure,*

(iv) $H_{\mathcal{RO}\circ MD_{\mathcal{RO}}\circ X}$ *is insecure,*

(v) $H_{X\circ\mathcal{RO}\circ MD_{\mathcal{RO}}}$ *is insecure.*

In [30], the plain Merkle-Damgård hash function with ideal compression functions $MD_{\mathcal{RO}}$ was shown to be insecure.

Proof. (i) Although this result was discussed in [30], the proof seems to be missing in the published version of the paper. Our task is to describe an efficient simulator S which can emulate the oracles F and G in the Random case. Namely, we will

- give a description of S,
- show that S is efficient, and
- show that the probability that an efficient distinguisher is successful (in differentiating this Random case from the Algorithm case) is negligible.

Description of the Simulator S.

1. F **oracle queries:** In order to simulate $H' = F(H, M)$ we use the natural approach:

   - If we have answered the same query before we return the same value H' again.
   - Else we choose the answer to the queries of H' at random.

   But we have to take care that the randomly chosen answer must not be equal to an answer of any prior query. If this does happen, we call this event Bad.

   If Bad has happened, it follows that a message chain is indicated where, with overwhelming probability, should be none. The probability for Bad is negligible as will be shown later on.

   For all oracle queries we perform record keeping. There are two reasons for this: First, to detect when Bad happens and, second, we need to know the values to answer the G oracle queries. As a result we have a record of numerous chains

   $$[H_0, M_{k_1}] \mapsto [H_{k_1}, M_{k_2}] \mapsto \cdots \mapsto [H_{k_l}, \perp].$$

   We call H_{k_j} accessible if there exists a chain starting with an initial value H_0, $H_{k_j} \neq H_0$, that contains H_{k_j}.

   Remarks: All H_{k_j} are different (or Bad would have occurred). Initially, there does exist one chain for the initialization value: $[H_0, \perp]$.

2. G **oracle queries:** For simulating oracle queries for $H' = G(H)$ we consider two cases:

   a) H is not accessible: choose a random value $H' \in \{0,1\}^n$. Add a new chain $[H', \perp]$ into the record and return H'.

b) H is accessible: There exists one specific chain that contains H. Therefore, we can extract the related message $M = (M_{k_1}, \ldots M_{k_j})$ from the chain and dispatch it to the the random oracle $\mathcal{H}_{\text{Rnd}}$. We return $H' = \mathcal{H}_{\text{Rnd}}(M)$.

Efficiency. The operations are clearly efficient. They all can be performed using standard techniques.

Success probability of the distinguisher. We will now show that, if the event Bad does not occur, our simulator fools the adversary by proving that the distribution of the adversary's output in the Algorithm case (without simulator S) is identical to the distribution of the adversary's output in the Random case (with simulator S). Afterwards, we will show that the probability of Bad is negligible. This concludes the proof.

Lemma 10.8. *The simulator S only aborts if* Bad *occurs.*

Proof: This is directly implied by the description of the simulator S. □

Lemma 10.9. *If the simulator does not abort the adversary's view of* Algorithm *and* Random *is identical.*

Proof:

1. Random*: Only independent random values are returned, Either by a random oracle, the hash function, or random values chosen by the simulator (simulating F and G as being random oracles). If the same query has been answered before, the same value is returned again.*

2. Algorithm*: Here, F and G are random oracles and, consequently, also the hash function.*

In either case the distinguisher can only see random values. □

Lemma 10.10. *The probability that* Bad *occurs is negligible.*

Proof: If Bad *has not happened after the first q queries to the simulator (either for F or G), then the probability that it happens on the $(q+1)$-st query is at most $(q+1) \cdot 2^{-n}$. This is because there are at most $(q+1)$ answers including the initial value H_0. Therefore, if the distinguisher makes a total of q queries, the probability of the event* Bad *is at most $(q^2+q) \cdot 2^{-n}$.* This completes the proof. □

(ii) This proof is essentially the same as the proof of Theorem 10.3 (ii). We only have to take care for the Merkle-Damgård construction. Here, we only give the distinguisher D:

1. Choose a random message $M \in \{0,1\}^*$.
2. Compute $u = MD_P(M)$.
3. Ask the F-oracle for $v = F(u)$.
4. Ask the hash-oracle for $z = H(M)$.
5. If $z = v$ output algorithm, else output random.

(iii) This proof is essentially the same as the proof of Theorem 10.3 (iii). We only have to take care for the Merkle-Damgård construction. Here, we only give the distinguisher D:

1. Choose a random message $M \in \{0,1\}^*$.
2. Ask the hash-oracle for $z = H(M)$.
3. Use the F-oracle to calculate $u = MD_F(M)$.
4. Compute $v = P(u)$.

5. If $z = v$ output ALGORITHM, else output RANDOM.

(iv) The proof is essentially the same as in 10.3 (iv). So if this hash function is secure, we could use our simulator to invert the padding function R efficiently.

(v) The proof is essentially the same as in 10.3 (v). So if this hash function is secure, we could use our simulator to invert the CROWF G efficiently.

□

Paradox.

Again, as discussed in Section 10.2, we get a secure hash function if we substitute the CROWF by an invertible function. For the hash functions (ii)-(v) similar results as given in Theorem 10.4 can easily be stated.

Part (iv) of the theorem might be somewhat surprising, at least in this form. Most of the padding functions have the property of being easily invertible. But in the indifferentiability world this is a must-have feature for secure hash functions. If the padding function R is not efficiently invertible, $NMAC$ would be insecure.

In [30] the authors do not care about the padding function. But this turns out to be somewhat shortsighted in the case of indifferentiable secure hash functions. Even such a simple and (in the analysis phase) easily to be forgotten function can drive a hash function insecure if it is added.

10.4. Mix-Compress-Mix

We now analyze the Mix-Compress-Mix (MCM) construction given by Ristenpart et al. [140]. In short, it is built as $\mathcal{RO}_i - X - \mathcal{RO}_i$. Note that there was given a more efficient instantiation by Lehmann and Tessaro [108].

Definition 10.11. *Fix numbers $\eta > 0$, $\tau \geq 0$, $L > 0$, and a hash key $K = (k_1, k, k_2)$. Let*

$$\begin{aligned} \epsilon_1(k_1, \cdot) &: \{0,1\}^{\leq L} \to \{0,1\}^{\leq L+\tau}, \\ H(k, \cdot) &: \{0,1\}^{\leq L+\tau} \to \{0,1\}^{\eta}, \\ \epsilon_2(k_2, \cdot) &: \{0,1\}^{\eta} \to \{0,1\}^{\eta+\tau}, \end{aligned}$$

whereas ε_1 has stretch τ, i.e.*, $|\varepsilon_1(k_1, M)| = |M| + \tau$. The functions ϵ_1 and ϵ_2 are injective. The hash function $MCM_{\epsilon_1,H,\epsilon_2} : \{0,1\}^{\leq L} \to \{0,1\}^{\eta+\tau}$ for a fixed key $K = (k_1, k, k_2)$ is computed by*

$$MCM_{\epsilon_1,H,\epsilon_2}(M) = \epsilon_2(k_2, H(k, \epsilon_1(k_1, M))).$$

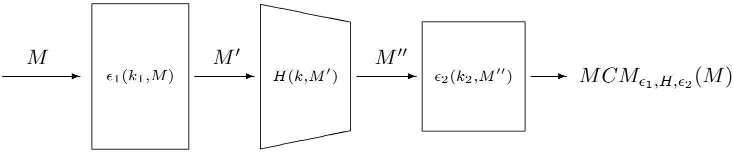


Figure 10.3.: The MCM construction $MCM_{\epsilon_1,H,\epsilon_2}(M)$. H is a CROWF, ϵ_1, ϵ_2 are injective ("*mixing*") functions, $K = (k_1, k, k_2)$ is the previously fixed hash key.

Definition 10.12. *As in Definition 10.2, a subscript i denotes an injective function/oracle whereas a subscript x denotes that we explicitly do not care whether the function/oracle is injective or not.*

(i) The hash function $H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i} : \{0,1\}^{\leq L} \to \{0,1\}^{\eta+\tau}$ for a message $M \in \{0,1\}^{\leq L}$ is defined by

$$H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i}(M) = MCM_{F_i^{(\leq L \to \leq L+\tau)}, D^{(\leq L+\tau \to \eta)}, G_i^{(\eta \to \eta+\tau)}}(M)$$

whereas F *and* G *both have stretch* τ.

(ii) *Modification/Partial Instantiation I: The hash function* $H_{\mathcal{RO}_x \circ X \circ Y} : \{0,1\}^{\leq L} \rightarrow \{0,1\}^{\eta+\tau}$ *for a message* $M \in \{0,1\}^{\leq L}$ *is defined by*

$$H_{\mathcal{RO}_x \circ X \circ Y}(M) = MCM_{F_i^{(\leq L \rightarrow \leq L+\tau)}, D^{(\leq L+\tau \rightarrow \eta)}, E_i^{(\eta \rightarrow \eta+\tau)}}(M)$$

whereas F *and* E *both have stretch* τ.

(iii) *Modification/Partial Instantiation II: The hash function* $H_{Y \circ X \circ \mathcal{RO}_x} : \{0,1\}^{\leq L} \rightarrow \{0,1\}^{\eta+\tau}$ *for a message* $M \in \{0,1\}^{\leq L}$ *is defined by*

$$H_{Y \circ X \circ \mathcal{RO}_x}(M) = MCM_{E_i^{(\leq L \rightarrow \leq L+\tau)}, D^{(\leq L+\tau \rightarrow \eta)}, G_i^{(\eta \rightarrow \eta+\tau)}}(M)$$

whereas E *and* G *both have stretch* τ.

(iv) *Modification III: The hash function* $H_{X \circ \mathcal{RO} \circ Y} : \{0,1\}^{\leq L} \rightarrow \{0,1\}^{\eta+\tau}$ *for a message* $M \in \{0,1\}^{\leq L}$ *is defined by*

$$H_{X \circ \mathcal{RO} \circ Y}(M) = MCM_{D^{(\leq L \rightarrow \leq L+\tau)}, F^{(\leq L+\tau \rightarrow \eta)}, E^{(\eta \rightarrow \eta+\tau)}}(M)$$

whereas D *and* E *both have stretch* τ.

(v) *Extension I: The hash function* $H_{Y \circ \mathcal{RO}_i \circ X \circ \mathcal{RO}_i} : \{0,1\}^{\leq L} \rightarrow \{0,1\}^{\eta+\tau}$ *for a message* $M \in \{0,1\}^{\leq L}$ *is defined by*

$$\begin{aligned} & H_{Y \circ \mathcal{RO}_i \circ X \circ \mathcal{RO}_i}(M) \\ & = E^{(\eta+\tau \rightarrow \eta+\tau)}(MCM_{F_i^{(\leq L \rightarrow \leq L+\tau)}, D^{(\leq L+\tau \rightarrow \eta)}, G_i^{(\eta \rightarrow \eta+\tau)}}(M)) \end{aligned}$$

whereas F *and* G *both have stretch* τ.

(vi) *Extension II: The hash function* $H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i \circ Y} : \{0,1\}^{\leq L} \rightarrow \{0,1\}^{\eta+\tau}$ *for a message* $M \in \{0,1\}^{\leq L}$ *is defined by*

$$\begin{aligned} & H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i \circ Y}(M) \\ & = MCM_{F_i^{(\leq L \rightarrow \leq L+\tau)}, D^{(\leq L+\tau \rightarrow \eta)}, G_i^{(\eta \rightarrow \eta+\tau)}}(E^{(\leq L \rightarrow \leq L)}(M)) \end{aligned}$$

whereas F_i *and* G_i *both have stretch* τ.

Theorem 10.13. *(i) $H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i}$ is secure if X is a $\Delta-$regular function (i.e., every image of H has approximately the same number of preimages, for details see [140]).*

(ii) $H_{\mathcal{RO}_x \circ X \circ Y}$ is insecure.

(iii) $H_{Y \circ X \circ \mathcal{RO}_x}$ is insecure.

(iv) $H_{X \circ \mathcal{RO} \circ Y}$ is insecure.

(v) $H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i \circ Y}$ is insecure.

(vi) $H_{Y \circ \mathcal{RO}_i \circ X \circ \mathcal{RO}_i}$ is insecure.

Proof. (i) This proof is given in [140, Theorem 3.2]. Note that the condition of injective random oracles is only needed for their special conclusion (proving collision resistance in the standard model) – it is not compulsory to prove the MCM construction secure in the indifferentiability framework.

(ii) The proof is essentially the same as in Theorem 1 (ii).

(iii) The proof is essentially the same as in Theorem 1 (iii).

(iv) This can be proved either by the proof of Theorem 1 (ii) or (iii)

(v) The proof is essentially the same as in Theorem 1 (ii).

(vi) The proof is essentially the same as in Theorem 1 (iii).

10.5. Shady Design Principles for Secure Hash Functions

Definition 10.14. *Let $k \in \mathbb{N}$. Let $S^1 : \{0,1\}^* \rightarrow \{0,1\}^m$, $S^j : \{0,1\}^m \rightarrow \{0,1\}^m$, $1 < j < k$, and $S^k : \{0,1\}^m \rightarrow \{0,1\}^n$ be functions. The hash function H for a message M is defined by*

$$H(M) = (S^k \circ S^{k-1} \circ \ldots \circ S^1)(M)$$

Here, we generally do not care whether the functions S^j, $1 \leq i \leq k$, are ideal or not. Using the technique applied above it is easy to show that the following theorem holds.

Theorem 10.15. *(Design Principles for Secure Hash Functions) Let $\mathcal{H}$ be defined as in Definition 10.14. Let H be a secure (indifferentiable) hash function. Then it must hold:*

(i) S^1 is not a one way function.

(ii) S^k is not a one way function.

Proof. (i) If S^1 is a one way function we could use the simulator (as H is secure) to invert S^1. The proof is essentially the same as the proof of Theorem 10.3 (ii).

(ii) If S^k is a one way function we could use the simulator (as H is secure) to invert S^k. The proof is essentially the same as the proof of Theorem 10.3 (iii).

□

This simple design principle is mandatory to all (indifferentiable) secure hash functions. Note that in the MCM construction, the one way function is in the middle of two random oracles. Applying Theorem 10.15 we can conclude: It is possible to prove a structure secure only if the 'first' and the 'last' functions are random oracles or easily invertible functions, but we might be able – under some circumstances – to choose some functions different to a random oracle.

10.6. Discussion

The Random Oracle Model

All ideal world notions and their definitions are inherently based on the random oracle model. Before going into details on indifferentiability itself in Section 10.6.1 let us recall some results from the literature on the random oracle model. As mentioned, there had been quite a few uninstantiability results, defining schemes provably secure in the random oracle model, but insecure when instantiated by any efficient function. One can argue that all of these constructions are malicious. They are designed to be insecure. But either one relies on heuristics and intuitions, or one relies on proofs. If one puts proofs above all other aspects, then counter-examples do invalidate the proofs.

10.6.1. The Ambiguity of Indifferentiability in the Design of Practical Hash Functions

The ideas from Coron et al. [30] have been very influential and inspiring for a lot of researchers. Namely, there have been quite a few proposals for hash function structures provably "indifferentiable from a random oracle", often in addition to other security requirements as, *e.g.*, in [11, 16, 110].

But there seems to be some contradiction in the reasoning from [30]: The *same* formalism can be used to indicate the structural soundness of an implementation, as well as the presence of structural weaknesses.

The contradiction is not on the formal level – we do not claim any flaw in the theorems or proofs of [30]. If all components (*e.g.*, compression functions) of a hash function are ideal (*i.e.*, random oracles) we do not get ambiguous results. If all components are non-ideal we cannot use the indifferentiability framework to prove anything. But if some of the primitives are ideal and some are not (as for example in [11]) we can get ambiguous results for security proofs. Our research seems to indicate that the indifferentiability model is of limited use for proving the security of

- mixed-model hash functions (using complexity-theoretic and ideal components at the same time) *and*
- practical hash functions (*e.g.*, as the SHA-3 candidates).

One might conclude that if any possible description of a structure is insecure in the indifferentiability framework (*e.g.*, in the case for Merkle-Damgård) then the hash structure is flawed. But it is not clear what we shall conclude for a concrete instantiation if one modeling is secure but another is not.

Taking a secure function (using only ideal components) we have shown in Section 10.2 and 10.3 how slight modifications (*i.e.*, adding a pre-processing or post-processing function) or partial instantiations (*i.e.*, starting our way towards an instantiated hash function) might possibly drive them insecure.

But in addition to an inherent theoretical motivation, the notion of security in [30] has also been motivated by the need to decide if the structure of a hash function is sound or flawed. A criterion for good hash function structures is very valuable for hash function designers, indeed. On the strictly theoretical side, there is nothing wrong, and studying this kind of security remains an interesting topic for theoretical cryptography.

10.6.2. Final Remarks

The random oracle model makes it possible to design cryptographic functions secure only in the ideal world. As discussed in Section 10.1, the notions of indistinguishability, preimage awareness, and indifferentiability from a public-use random oracle seem to be too weak for designing a secure, practical, and general purpose hash structure.

The right level of abstraction. If we state the discussion of Section 10.6.1 somewhat different we can come up with the following: For designing a hash function one might come up with a model/structure that describes the hash function on an abstract level. Then, one might try to find an indifferentiability proof for this structure – given that some of the components are ideal. This process usually involves some sort of tweaking of the structure in order to 'find' the proof. Therefore, we state that this structure is secure. But if we start with an implementation (*i.e.*, a practical hash function) and want to evaluate its security in terms of indifferentiability, we are faced with the problem of the right level of abstraction/kind of modeling. If we abstract away all the details and come up with a structure only consisting of a random oracle, *all* hash functions are trivially secure (again, in terms of indifferentiability). If we abstract nothing, the indistinguishability framework does not have an answer to our question since we have no ideal components. But if we start abstracting some of the components we might be faced with the problem of finding some abstractions that are secure, and some that are not. And we might not know what to conclude for the security of the implementation.

Open Problems. It remains an open problem to derive an ideal world criterion to support the design of general purpose practical hash functions – telling us if the internal structure of a hash function is flawless or not. Certainly, a security proof (*i.e.*, a proof of a hash function being indifferentiable from a random oracle, when modeling some or all the internal functions as random oracles) is comforting. But pursuing this kind of security property requires great care since authors of a new hash function could be tempted to change, *e.g.*, some one-way final transform of their hash function into an easily invertible transformation. This could enable a theoretical security proof in the first place, while at the same time, practically weaken the hash function.

Designers of practical hash functions, who accept the indifferentiability framework at face value, may be tempted to make poor design decisions. The indifferentiability framework suggests corrections to structures which sometimes make only sense in the ideal world but that have no real-word mapping. Even worse, the danger is that these very corrections drive the corresponding real-world hash function less secure.

Authors of new hash functions are well advised to prove other security properties, such as the established collision resistance, preimage resistance, and second-preimage resistance under some reasonable standard-model assumptions, perhaps in addition to analyzing theoretical security properties, such as the indifferentiability from a random oracle.

Part IV.

On-Line Authenticated Encryption for Practical Applications

11 Results Summary

On-Line Authenticated Encryption (OAE) combines privacy with data integrity and is on-line computable. Most block cipher-based schemes for Authenticated Encryption can be run on-line and are provably secure against *nonce-respecting* adversaries. But they fail badly for more general adversaries. This is not a theoretical observation only – in practice, the reuse of nonces (a *number used once*) is a frequent issue. A prominent example is the PlayStation 3 'jailbreak' [76], where application developers used a constant that was actually supposed to be a nonce for a digital signature scheme.

In recent years and initiated by a seminal work of Rogaway and Shrimpton [150], cryptographers developed *misuse-resistant* schemes for Authenticated Encryption. These guarantee excellent security even against general adversaries which are allowed to reuse nonces. Their disadvantage is that encryption can be performed in an off-line way, only.

We consider OAE schemes dealing both with nonce-respecting *and* with general adversaries by introducing McOEx, a construction based only on a simple blockcipher $E \in \textsc{Block}(n, n)$. It provably guarantees reasonable security against general adversaries as well as standard security against nonce-respecting adversaries.

As an interesting side-effect of our analysis, it seems that this actually is the first practical scheme in literature that does require some sort of related key security of a blockcipher. This draws attention especially to AES where the relevance of related-key attacks is still debated.

11.1. Introduction

On-Line Authenticated Encryption (OAE). Application software often requires a network channel that guarantees the privacy and authenticity of data being communicated between two parties. Cryptographic schemes able to meet both of these goals are commonly referred to as Authenticated Encryption (AE) schemes. The ISO/IEC 19772:2009 standard for AE [78] defines generic composition (Encrypt-then-MAC [10]) and five dedicated AE schemes: OCB2 [145], SIV [150] (denoted as "Key Wrap" in [78]), CCM [40], EAX [14], and GCM [119]. To integrate an AE-secure channel most seamlessly into a typical software architecture, application developers expect it to encrypt in an *on-line* manner meaning that the i-th ciphertext block can be written before the $(i+1)$-th plaintext block has to be read. A restriction to off-line encryption, where usually the entire plaintext must be known in advance (or read more than once), is an encumbrance to software architects.

Nonces and their reuse. Goldwasser and Micali [69] formalized encryption schemes as stateful or probabilistic, because otherwise important security properties are lost. Rogaway [144, 146, 148] proposed a unified point of view, by always defining a cryptographic scheme as a deterministic algorithm that takes a user supplied nonce. So the application programmer – and not the encryption scheme – is responsible for flipping coins or maintaining state. This reflects cryptographic practice since the algorithm itself is often implemented by a multi-purpose cryptographic library which is more or less application-agnostic.

In theory, the concept of a nonce is simple. In practice, it is challenging to ensure that a nonce is *never* reused. Flawed implementations of nonces are ubiquitous [20, 76, 97, 156, 168]. Apart from implementation failures, there are fundamental reasons why software developers can not always prevent nonce-reuse. A persistently stored counter, which is increased and written back each time a new nonce is needed, may be reseted by a backup – usually after some previous data loss. Similarly, the internal and persistent state of an application may be duplicated when a virtual machine is cloned, etc.

Related Work and Our Contribution. We aim to achieve *both simultaneously*: security against nonce-reusing adversaries (sometimes also called nonce-misusing adversaries) *and* support for on-line encryption in terms of an AE scheme. Apart from generic composition (Encrypt-then-Mac, EtM), none of the ISO/IEC 19772:2009 schemes – in fact, no previously published AE scheme at all – achieves both of these goals, cf. Table 11.1. In this table, we classify a vast variety of provably secure block cipher-based AE schemes with respect to their on-line ability and against which adversaries (nonce-respecting versus nonce-reusing) they have been proven secure.

Since EtM is not a concrete scheme but merely a generic construction technique, there are some challenges left in order to make it secure in both ways: First, an appropriate on-line cipher has to be chosen. Second, a suitable, on-line computable, secure deterministic MAC must be selected. And, third, the EtM scheme requires at least two *independent* keys to be secure and, since two schemes are used in parallel, is likely to waste resources in terms of run time and – important for hardware designers – in terms of space. Since EtM first has to be turned into an OAE scheme by making the appropriate choices, we do not include it in our analysis.

In order to close the apparent gap in the upper-right of Table 11.1, we considered the schemes in the upper-left, whether they would fit.

As it turned out, we actually found nonce-reuse attacks for *all* of those schemes. An overview of the workload of these attacks is given in Table 11.2. The attack details are given in Chapter 12. We present a new construction method for efficient AE schemes, called McOEx, that is able to fill this gap.

security:	**nonce-respecting adv.**	**nonce-reusing adv.**
on-line	CCFB[113] CHM[79] CIP[80] CWC[98] EAX[14] GCM[119] IACBC[85] IAPM[85] **McOEx**[53] OCB1-3[100, 145, 148] RPC[25] TAE[111] XCBC[67]	**McOEx**[53]
off-line	BTM[81] CCM[40] HBS[82] SIV[150] SSH-CTR[135]	BTM[81] HBS[82] SIV[150]

Table 11.1.: Classification of provably secure block cipher-based AE Schemes. CCM and SSH-CTR are considered off-line because encryption requires prior knowledge of the message length. Note that the McOEx scheme, because of being on-line, satisfies a slightly weaker security definition against nonce-reusing adversaries than SIV, HBS, and BTM.

Initial Value (IV) based AE schemes maximally forgiving of repeated IV's have been addressed in [150], coining the notion of "misuse resistance" and proposing SIV as a solution. SIV and related

	privacy attack	authenticity attack
CCFB [113]	$O(1)$	$O(1)$
CCM [40]	$O(1)$	$\ll 2^{(n/2)}$ [63]
CHM [79]	$O(1)$	$O(1)$
CIP [80]	$O(1)$	$O(1)$
CWC [98]	$O(1)$	$O(1)$
EAX [14]	$O(1)$	$O(1)$
GCM [119]	$O(1)$	$O(1)$
IACBC [85]	$O(1)$	$O(1)$

	privacy attack	authenticity attack
IAPM [85]	$O(1)$	$O(1)$
OCB1 [148]	$O(1)$	$O(1)$
OCB2 [145]	$O(1)$	$O(1)$
OCB3 [100]	$O(1)$	$O(1)$
RPC [25]	$O(1)$	$O(1)$
TAE [111]	$O(1)$	$O(1)$
XCBC [67]	$O(2^{n/4})$	?

Table 11.2.: Overview of our **nonce-reuse** attacks on published AE schemes, excluding SIV, HBS, and BTM, which have been explicitly designed to resist nonce-reuse. Almost all attacks achieve an advantage close to 1. For the attacks, the relevant workloads, *e.g.*, W, are provided – meaning that an adversary, restricted to W units of time and W chosen plaintexts/ciphertexts is successful (with overwhelming probability). Details are given in Chapter 12.

schemes (HBS [82] and BTM [81]) actually provide excellent security against nonce-reusing adversaries, though there are other potential misuse cases we discuss in Section 12.2. The main disadvantage of the mentioned schemes is that they are inherently off-line: For encryption, one must either keep the entire plaintext in memory, or read the plaintext twice.

Ideally, an adversary seeing the encryptions of two (equal-length) plaintexts P_1 and P_2 can not even decide if $P_1 = P_2$ or not. When using a nonce more than once, deciding about $P_1 = P_2$ is easy. SIV and its relatives ensure that nothing else is feasible for nonce-reusing adversaries. In the case of on-line encryption, where the first few bits of the encryption of a lengthy message must not depend on the last few bits of that message, there is unavoidably something beyond $P_1 = P_2$. The adversary can compare any two ciphertexts for their longest common prefix, and then conclude about common prefixes of the secret plaintexts. Our notion of *misuse resistance* means that this is all the adversary might gain. Even in the case of a nonce-reuse, the adversary (1) can't do anything beyond determining the length of common plaintext prefixes and (2) the scheme still provides the usual level authenticity for AE (INT-CTXT). The first property is common for on-line ciphers/permutations (OPRP) [6]. Recently, [154] studied the design of on-line ciphers from tweakable block ciphers bearing some similarities to our approach, especially to TC3. In contrast to McOEx, the constructions from [154] provide no authentication and the McOEx scheme can easily work with a 'normal' blockcipher and does not require a tweakable blockcipher.

Design Principles for AE Schemes. The question how to provide authenticated encryption (without stating that name) when given a secure on-line cipher, is studied in [7], the revised and full version of [6]. The first idea in [7] only provides security if all messages are of the same length. The second idea repairs that by prepending the message's length to the message, at the cost of being off-line, since the message length must be known at the beginning of the encryption process. The third idea is to prepend and append a random W to a message M and then to perform the on-line encryption of $(W||M||W)$. This looks promising, but the same W is used for two different purposes, putting different constraints on the generation of W. For privacy, it suffices that W behaves like a nonce, not requiring secrecy or unpredictability. Even if W is not a nonce, but the same W is used for the encryption of several messages, all the adversary can determine are the lengths of common plaintext prefixes, as we required for nonce-reuse. On the other hand, authenticity actually assumes a *secret or unpredictable* W, rather than a nonce. If the adversary can guess W before choosing a message, it asks for the authenticated encryption of $(M||W)$. Then it can predict the authenticated encryption of M without actually asking for it.

The McOEx scheme replaces the "random" W by a proper nonce and a *key-dependent* and *header-dependent* value τ, performing a nonce-dependent on-line encryption of $(M||\tau)$. The encryption can also depend on some associated data which is authenticated in the process of encryption turning McOEx in an OAEAD *(On-Line Authenticated Encryption with Associated Data)* scheme.

Roadmap. In Section 11.2 we show how the McOEx scheme works and provide performance data when instantiated with either the AES-128 or Threefish-512 as the underlying blockcipher. The security analysis and formal notations are provided in Chapter 13 after showing some attack details on how nonce-misuse attacks affect popular AE schemes in Chapter 12.

11.2. McOEx Instances

We give two concrete instances of our OAE schemes including performance data. One instance, McOEx-AES uses AES-128 as the core component while McOEx-Threefish uses the blockcipher Threefish-512, a cipher with 512-bit block size and key size, which is the core working component inside the SHA-3 finalist Skein[126]. We assume in our discussion that the plaintext is padded using the 10* padding procedure as discussed in Section 1.2.2.

Let $E \in \textsc{Block}(k, n)$. Note that for our chosen instances, AES-128 and Threefish-512, we have $n = k$. The pseudo code for these two McOEx instances is given in Table 11.4.

The algorithms, **EncryptAuthenticate** and **DecryptAuthenticate** are given for messages that are aligned on n-bit boundaries, *i.e.*, $M = (M_1, \ldots, M_L) \in (\{0,1\}^n)^L$ and a header $H = (H_1, \ldots, H_{L_H}) \in (\{0,1\}^n)^{L_H}$ for some integers $L, L_H \geq 1$. Note that the header H is required to be at least one block in length since this then represents essentially the initial value. If the header is longer than one block, we might opt for header precomputation and actually choose the *last* block of the header (H_{L_H}) to be our 'nonce'. See Figure 11.1 and Table 11.4.

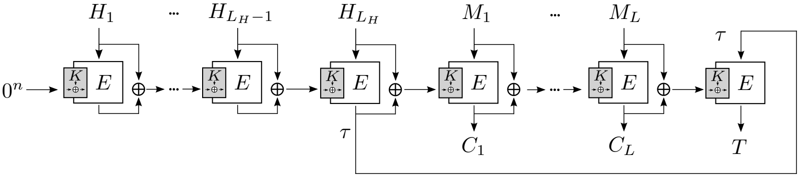


Figure 11.1.: The McOEx-AES/McOEx-Threefish encryption process. The key used for the blockcipher E is computed by the injective function $\varphi(K,V) = K \oplus V$ with the secret key K and the chaining value V as inputs. The tag returned is the n-bit value T. The output is $(T, C_1, \ldots, C_L)$. The decryption process works in a similar way from 'left to right', only the blockcipher component E_φ is replaced by its counterpart E_φ^{-1} apart from one exception: when processing the header to compute τ.

Block cipher	impl. type	Message length in bytes (no extra header)							
		16	40	128	256	576	1500	4096	32768
AES-128	software	62.9	42.6	26.8	24.3	22.9	22.3	22	21.8
AES-128	AES-NI	72.2	19.4	12.7	11.7	11.2	10.9	10.8	10.7
Threefish-512	software	82.3	33.9	13.3	10.1	8.3	7.6	7	6.8

Table 11.3.: Performance values (cycles-per-byte), measured on an Core i5 540M for AES-128 and Threefish-512. The AES software implementation is based on Gladman [66], whereas the hardware implementation is based on the Intel AES-NI Sample Library [32]. The Threefish implementation uses the NIST/SHA-3 reference source as provided by the Skein authors [126]. Any implementation is some sort of 'naive' and no special optimizations have been performed.

```
EncryptAuthenticate($H, M$)
1. $U \leftarrow \varphi(K, 0^n)$
2. for $i = 1, \ldots, L_H - 1$ do
       $U \leftarrow \varphi(K, H_i \oplus E_U(H_i))$
3. $\tau \leftarrow E_U(H_{L_H})$
4. $U \leftarrow \varphi(K, H_{L_H} \oplus \tau)$
5. for $i = 1, \ldots, L$ do
       $C_i \leftarrow E_U(M_i)$
       $U \leftarrow \varphi(K, M_i \oplus C_i)$
6. $T \leftarrow E_U(\tau)$
7. return $(C_1, \ldots, C_L, T)$
```

```
DecryptAuthenticate($H, C, T$)
1. $U \leftarrow \varphi(K, 0^n)$
2. for $i = 1, \ldots, L_H - 1$ do
       $U \leftarrow \varphi(K, H_i \oplus E_U(H_i))$
3. $\tau \leftarrow E_U(H_{L_H})$
4. $U \leftarrow \varphi(K, H_{L_H} \oplus \tau)$
5. for $i = 1, \ldots, L$ do
       $M_i \leftarrow E_U^{-1}(C_i)$
       $U \leftarrow \varphi(K, M_i \oplus C_i)$
6. if $T = E_U(\tau)$ then
   return $(M_1, \ldots, M_L)$ else return $\perp$
```

Table 11.4.: McOEx using a blockcipher $E \in$ Block(n, n); $H_1, \ldots H_{L_H - 1}$ is the optional header; H_{L_H} is the mandatory initial value; $M, C \in (\{0,1\}^n)^L$ is the plaintext/ciphertext and $T \in \{0,1\}^n$ is the tag; $\varphi(A, B) = A \oplus B$

Remarks. Note that we actually do need some form of (restricted) related key resistance for the blockcipher E in the case of using a blockcipher from Block (128,128) – since the adversary can 'partially control' some relations among keys used in the computation. Interestingly, the search for high-performance ciphers seems still not to be over. It is worth mentioning, that software Threefish-512 performs considerably better than hardware AES-128. This seems to be mainly due to the slow key scheduler implementation of AES-128 and the shorter block size. Naturally, this leads to the question whether a 128-bit block size might be somewhat disadvantageous in nowadays big-memory environments.

12

Misuse Attacks on Authenticated Encryption Schemes

12.1. Schemes Without Claimed Resistance Against Nonce-Misuse

Cipher-block-chaining (CBC) is an unauthenticated encryption mode which is sometimes used as the encryption component of an AE scheme. But one can easily distinguish CBC encryption from a good on-line cipher, if the nonce (or the IV) is constant. The attack from [6] only needs three chosen plaintexts. Counter mode, which has been very popular among the designers of AE schemes, fails terribly in nonce-misuse settings, since it generates exactly the same key stream twice when the nonce is repeated. It was to be expected that a scheme using counter mode (CTR) or CBC inherits the nonce-misuse issue from that mode. But, as it turned out, common AE schemes also fail at the authenticity frontier, see Table 11.2 for an overview. This is an unpleasant surprise, since the cryptographic community has known well deterministic MACs for a long time – so why is the authenticity provided by most authenticated encryption schemes so much more fragile than the authenticity provided by well-known MACs?

The following two attack patterns will be used in most of our attacks.

Repeated Key Stream. Many AE schemes generate a key stream $S = F_K(V)$ of length $|M|$, depending on the secret key K and the nonce V, and encrypt the message M by computing the ciphertext $C = S \oplus M$, typically by applying a blockcipher in CTR mode. If the same nonce is used more than once, the following attack straightforwardly breaks the privacy:

- Encrypt a plaintext M under the nonce V to a ciphertext C with tag T.
- Encrypt a plaintext $M' \neq M$ under the same V to a ciphertext C' and a tag T'.
- It turns out that $C' = C \oplus M \oplus M'$ holds.

Linear Tag. Many AE schemes, which generate a key stream $S = F_K(V)$ as above, apply the encrypt-then-authenticate paradigm and allow to rewrite the authentication tag T as

$$T = f(V) \oplus g(C),$$

where V is the nonce, C is the ciphertext, and f and g are some key-dependent functions. This enables the adversary to mount the following attack:

- Encrypt the plaintext M under the nonce V to (C, T) with $T = f(V) \oplus g(C)$.
- Encrypt the plaintext $M' \neq M$ with $|M'| = |M|$ under the nonce $V' \neq V$ to (C', T') with the tag $T' = f(V') \oplus g(C')$.
- Set $M'' := M' \oplus C' \oplus C$. Encrypt M'' under the nonce V' to (C'', T''). Observe $C'' = C$, thus $T'' = f(V') \oplus g(C)$.

- Set $T^* = T \oplus T' \oplus T'' = f(V) \oplus g(C')$, The adversary accepts (C', T^*) under V.

Two-Pass AE(AD) Modes: CWC [98], GCM [119], CCM [40], EAX [14], CHM [79] . All the common two-pass AE(AD) modes, CHM, CWC, GCM, CCM, and EAX use CTR mode as the underlying encryption operation and are thus vulnerable to the repeated key stream attack pattern. Four of them, CHM, CWC, GCM, and EAX, are designed according to the encrypt-then-authenticate paradigm, and are thus vulnerable to the linear tag attack pattern. The designers of CCM followed authenticate-then-encrypt, which seems to defend against the linear tag pattern. Forgery attacks against CCM have been presented in [63], though.

Mixed AE(AD) Modes: RPC [25] and CCFB [113]. RPC combines the CTR and electronic codebook mode. Given an n-bit blockcipher E under a key K and a c-bit counter cnt, RPC takes an $(n-c)$-bit plaintext block M_i and computes the ciphertext block $C_i := E_K(M_i || (\text{cnt} + i) \bmod 2^c)$. Authentication is performed locally for each ciphertext block: During decryption, RPC computes $(M_i || X_i) = E_K^{-1}(C_i)$ and accepts M_i as authentic if and only if $X_i = (\text{cnt} + i) \bmod 2^c$. The nonce defines cnt.

Under nonce-misuse, the same sequence $(\text{cnt}+i) \bmod 2^c$ of counter values is used for different messages. This makes it easy to attack the privacy – essentially, when encrypting messages of m $(n-c)$-bit blocks, RPC degrades into m independent electronic codebooks. Also, given two authentic ciphertexts, $(C_1^0, \ldots, C_L^0)$ and $(C_1^1, \ldots, C_L^1)$, any ciphertext $(C_1^{\sigma(1)}, \ldots, C_L^{\sigma(L)})$ with $\sigma(i) \in \{0, 1\}$ is valid, since authenticity is verified locally for each $C_i^{\sigma(i)}$. Similarly to RPC, CCFB is a combination of the CTR mode and the cipher feedback mode (CFB). Unlike RPC, CCFB generates a single "global" authentication tag. Variants of the repeated key stream pattern and the linear tag pattern apply to CCFB.

One-Pass AE(AD) Modes: IAPM [85], OCB1[148], OCB2[145], OCB3[100], TAE [111]. Given a nonce IV and a secret key K, the IAPM mode [85] encrypts a plaintext $(M_1, \ldots, M_m)$ to a ciphertext $(C_1, \ldots, C_m)$ and an authentication tag T as follows.

Initial step: Generate $m+2$ values $s_0, s_1, \ldots, s_{m+1}$ depending on IV and K, but not on the plaintext $(M_1, \ldots, M_m)$.

Encryption: For $i \in \{1, \ldots, m\}$: $C_i := E_K(M_i \oplus s_i) \oplus s_i$.

Authentication tag: $T := E_K(s_{m+1} \oplus \sum_{1 \le i \le m} M_i) \oplus s_0$.

Similarly to RPC, IAPM behaves like a set of m independent electronic codebooks and is vulnerable to the same distinguishing attack. A forgery can exploit the fact that two different equal-length messages $(M_1, \ldots, M_m)$ and $(M'_1, \ldots, M'_m)$, encrypted under the same nonce, have the same authentication tag $T = E_K(s_{m+1} \oplus \sum_{1 \le i \le m} M_i) \oplus s_0 = E_K(s_{m+1} \oplus \sum_{1 \le i \le m} M'_i) \oplus s_0$ if $\sum_{1 \le i \le m} M_i = \sum_{1 \le i \le m} M'_i$.

As much as our attacks are concerned, OCB1–3 and TAE are quite similar to IAPM, and the attacks are the same.

More One-Pass Modes: IACBC [85] and XCBC [67]. Given a nonce IV and a secret key K, the IACBC mode [85] encrypts a plaintext $(M_1, \ldots, M_m)$ to $(C_1, \ldots, C_m)$ and an authentication tag T as follows.

Initial step: Generate $m+1$ values $s_0, s_1, \ldots, s_m$ depending on V and K, but not on the plaintext $(M_1, \ldots, M_m)$.

Encryption: $x_0 := IV$; For $i \in \{1, \ldots, m\}$: $x_i := E_K(M_i \oplus x_{i-1})$, $C_i := x_i \oplus s_i$.

Authentication tag: $T := E_K(x_m \oplus \sum_{1 \le i \le m} M_i) \oplus s_0$.

The following nonce-misuse attack distinguishes IACBC encryption from an on-line permutation and also provides an existential forgery. For simplicity, we only consider 1-block messages $IV =: V \neq W$,

which we also use as nonces: Encrypt W under V to (C_1, T). Encrypt V under W to (C_1', T'). Encrypt V under V to (C_1'', T''). Set $C_1''' := C_1 \oplus C_1' \oplus C_1''$ and $T''' := T \oplus T' \oplus T''$.
(C_1''', T) is a valid encryption of W under W.

Given a nonce V and secret keys K and K', XCBC encrypts a plaintext $(M_1, \ldots, M_m)$ to a ciphertext $(C_1, \ldots, C_m)$ and an authentication tag T as follows.

Initial step: Generate $m+1$ values $s_1, \ldots s_{m+1}$ depending on V and K, but not on the plaintext $(M_1, \ldots, M_m)$.

Encryption:

1. $C_0 := E_K(V)$; $x_0 := E_{K'}(V)$;
2. Generate an additional message word $M_{m+1} := x_0 \oplus M_1 \oplus \cdots \oplus M_m$ for authentication.
3. For $i \in \{1, \ldots, m+1\}$: $x_i := E_K(M_i \oplus x_{i-1})$, $C_i := (x_i + s_i) \bmod 2^n$.

The best attack we have found for XCBC is not quite as damaging as the attacks on the other schemes, as the attack workload is at $O(2^{n/4})$, and the attack only provides a distinguisher, not a forger. For this nonce-misuse chosen-plaintext attack, we ignore the authentication tag:

1. Generate $2^{n/4}$ encryptions of messages M_1^i under a nonce V to C_1^i.

   Statistically, expect one pair $i \neq j$ such that the least significant $n/2$ bits of C_1^i are identical to the least significant $n/2$ bits of C_1^j.

2. Generate $2^{n/4}$ encryptions of messages (M_1^i, M_2^k) and (M_1^j, M_2^ℓ) under V to (C_1^i, C_2^k) and (C_1^j, C_2^ℓ), where the least significant $n/2$ bits of M_2^k and M_2^ℓ are the same.

   Statistically, expect one pair $k \neq \ell$ such that $C_2^k = C_2^\ell$ holds.

3. Choose an arbitrary M_3. Encrypt (M_1^i, M_2^k, M_3) and (M_1^j, M_2^ℓ, M_3) under V to $(C_1^i, C_2^k, C_3^{i,k})$ and $(C_1^j, C_2^\ell, C_3^{j,\ell})$.

   Observe $C_3^{i,k} = C_3^{j,\ell}$.

12.2. Off-Line Schemes Defeating Nonce-Misuse

We now give attacks on SIV [150], HBS [82], and BTM [81]. These type of attacks is different than the ones presented in the previous section.

Given a nonce N, a message M, and associated Data H, these schemes perform two steps:

1. Generate the authentication tag T from H, M, and N.

2. Encrypt M in CTR mode, using T as the nonce.

This is inherently off-line, because one must finish Step 1 before one can start Step 2. All of SIV, HBS, and BTM perform CTR mode encryption, but employ different MAC schemes to generate the tag T.

This usage of the CTR mode is vulnerable in an *on-line decryption misuse* case, where, during decryption, a would-be plaintext is compromised before the tag has been verified. A chosen-ciphertext adversary can exploit that to determine an unknown key stream and then to decrypt an unknown message.

Another misuse case may apply when nonce-misuse is possible and the sender reads the message twice, once for each of the two steps – if there is any chance that *the message has been modified* between the two read operations.

Note that both misuse cases become quite harmless if one replaces the CTR mode encryption by the application of an on-line permutation.

13 Security Analysis of McOEx

13.1. Preliminaries

Length of Longest Common Prefix (LLCP$_n$) The length of a string $x \in \{0,1\}^n$ is denoted by $|x| := n$. For integers $n, \ell, d \geq 1$, set $D_n^d = (\{0,1\}^n)^d$, $D_n^* := \bigcup_{d \geq 0} D_n^d$, and $D_{\ell,n} = \bigcup_{0 \leq d \leq \ell} D_n^d$. Note that D_n^0 only contains the empty string. For $M \in D_n^d$; we write $M = (M_1, \ldots, M_d)$ with $M_1, \ldots, M_d \in D_n$. For $P, R \in D_n^*$, say, $P \in D_n^p$ and $R \in D_n^r$, we define the *length of the longest common n-prefix* of P and R as

$$\mathrm{LLCP}_n(P, R) = \max_i \{P_1 = R_1, \ldots, P_i = R_i\}.$$

For a non-empty set $\mathcal{Q}$ of strings in D_n^*, we define $\mathrm{LLCP}_n(\mathcal{Q}, P)$ as $\max\limits_{q \in \mathcal{Q}}\{\mathrm{LLCP}_n(q, P)\}$. For example, if $P \in \mathcal{Q}$, then $\mathrm{LLCP}_n(\mathcal{Q}, P) = |P|/n$.

For convenience, we introduce a notation for a *restriction on a set.* If $\mathcal{Q} = \{0,1\}^a \times \{0,1\}^b \times \{0,1\}^c$, we write $\mathcal{Q}_{|b,c} = \{(B, C) \,|\, \exists A :$

$(A, B, C) \in \mathcal{Q}\}$. This generalizes in the obvious way.

On-Line Permutations We aim for larger permutations that not only permute single blocks but can handle multiple/variable block messages. Such a permutation, from D_n^* to D_n^*, is (n-)on-line if the i-th block of the output is determined completely by the first i blocks of the input.

Definition 13.1. *Let $n, k \geq 0$, $K \in \{0,1\}^k, H_{L_H} \in D_n$. A function $\Pi : \{0,1\}^k \times D_n^* \to D_n^*$ is an (n-)on-line permutation if for any fixed K, H_{L_H} the function $\Pi(K, H_{L_H}, \cdot)$ is a permutation and there exists for any message $M = (M_1, M_2, \ldots, M_m)$ a family of functions $\widetilde{\pi}^i : \{0,1\}^k \times \{0,1\}^n \times D_n^i \to D_n$, $i = 1, \ldots, m$, such that*

$$\begin{aligned}\Pi(K, H_{L_H}, M) = \;& \widetilde{\pi}_K^1(H_{L_H}, M_1)||\widetilde{\pi}_K^2(H_{L_H}, M[1..2]) \\ & ||\ldots|| \\ & \widetilde{\pi}_K^{m-1}(H_{L_H}, M[1..m-1])||\widetilde{\pi}_K^m(H_{L_H}, M[1..m]),\end{aligned}$$

where $M[a \ldots b] := M_a||M_{a+1}||\ldots||M_b$ with "$||$" being the concatenation of strings, holds.

An encryption scheme is (n-)on-line if the encryption function is (n-)on-line. A thorough discussion of on-line encryption and its properties can be found in [6].

Authenticated Encryption (With Associated Data) An authenticated encryption (AE) scheme is a tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Its aim is to provide privacy and data integrity. The key generation function $\mathcal{K}$ takes no input and returns a randomly chosen key K from the key space, *e.g.*, from $\{0,1\}^k$. The encryption algorithm $\mathcal{E}$ and the decryption algorithm $\mathcal{D}$ are deterministic algorithms that map values from $\{0,1\}^k \times \mathcal{H} \times D_n^*$ to a string or – if the input is invalid – the value $\perp$. The header $\mathcal{H}$ consists either only of the initial value/nonce $H_{L_H} \in D_n$ (if no associated data is to be authenticated/checked in the encryption/decryption process) or is a combination of H_{L_H} and a value from D_n^*. So $\mathcal{H} \subset D_n^+$ in either case. For sake of convenience, we usually write $\mathcal{E}_K^H(M)$ for $\mathcal{E}(K, H, M)$ and $\mathcal{D}_K^H(M)$ for $\mathcal{D}(K, H, M)$, where the message M is chosen from D_n^*, $H \in \mathcal{H}$, and a key from the key space. We require $\mathcal{D}_K^H(\mathcal{E}_K^H(M)) = M$ for any possible K, M, H, and define the tag size for a message $M \in D_n^*$ and header $H \in \mathcal{H}$ as $\textsc{tag}(H, M) := |\mathcal{E}_K^H(M)| - |M|$. We denote an authenticated encryption scheme with the requirement that the initial vector H_{L_H} is only used once in a *nonce based* scheme. Otherwise, we call such a scheme *deterministic*. Similarly, we call an adversary *nonce-respecting* (NR) if no nonce is used twice for any query. Otherwise, the adversary is called *nonce-ignoring* (NI).

13.2. Security Notions

(On-line) Authenticated Encryption schemes (OAE) try to achieve privacy and authenticity at the same time. Therefore, we need security notions to handle this twofold goal. For AE, there have been notions and their relations introduced for deterministic [151] and nonce based [10, 13, 86, 144, 148] AE schemes. In order to have one convenient toolset of notions, we adopt the notion of CCA3 security suggested in [151] as a *natural strengthening* of CCA2 security.

We parametrize our definition in order to define different – but closely related – notions by explicitly stating whether we mean an on-line or off-line scheme, $\omega \in \{\textsc{ae}, \textsc{aoe}\}$ and stating the adversary behavior as either nonce-respecting or nonce-ignoring, $\nu \in \{\textsc{nr}, \textsc{ni}\}$.

Definition 13.2 (CCA3(ω, ν)). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme with header space $\mathcal{H}$ and message space D_n^*, and fix an adversary A. The advantage of A breaking Π is*

```
Initialize$(\omega,\nu)$
  b $\xleftarrow{\$}$ $\{0,1\}$;
  if(b=1) then
    K ← $\mathcal{K}()$;

Encrypt$(H,M)$
  if ($\nu$ = NR and $H_{L_H} \in B$)
 then
    return $\bot$;
  if(b=1) then
    $C \leftarrow \mathcal{E}_K$(H,M);
  else
    $C \leftarrow \$^{\omega}(H,M)$;
  $B \leftarrow B \cup \{H_{L_H}\}$;
  $\boxed{\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(H,C)\};}$
  return C;
```

```
Finalize$(d)$
  return $(b = d)$;

Decrypt(H,C)
  if $((H,C) \in Q)$ then
    return $\bot$;
  if(b=1) then
    M ← $\mathcal{D}_K$(H,C);
  else
    M ← $\bot$(H,C);
  return M;
```

Figure 13.1.: $G_{\text{CPA}}(\omega,\nu)$ is the $\text{CPA}_{\Pi}^{(\omega,\nu)}$ game and $\boxed{G_{\text{CCA3}}(\omega,\nu)}$ the $\text{CCA3}_{\Pi}^{(\omega,\nu)}$ game, where $\Pi = (\mathcal{K},\mathcal{E},\mathcal{D})$. Game G_{CCA3} contains the code in the box while G_{CPA} does not. The oracle $\$^{\text{AE}}(H,M)$ returns a string of length $|M| + \text{TAG}(H,M)$, this string is on-line compatible if $\omega = \text{AOE}$. H_{L_H} denotes the last block of the header representing the nonce/initial value.

defined as

$$\mathbf{Adv}_{\Pi}^{\text{CCA3}(\omega,\nu)}(A) = \left|\Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[A^{\$^{\omega}(\cdot,\cdot),\bot(\cdot,\cdot)} \Rightarrow 1\right]\right|.$$

The adversary's random-bits oracle, $\$^{\text{AE}}(\cdot,\cdot)$ or $\$^{\text{AOE}}(\cdot,\cdot)$, return on a query with header $H \in \mathcal{H}$ and plaintext $X \in D_n^*$ a random string of length $|\mathcal{E}_K(M)|$ which is either on-line or not, depending on the variable ω. The $\perp(\cdot,\cdot)$ oracle returns $\perp$ on every input. We assume *wlog.* that the adversary A never asks a query which answer is already known. It is easy to see that we can rewrite the term given in Definition 13.2 as

$$\begin{aligned}
\mathbf{Adv}_{\Pi}^{\mathrm{CCA3}(\omega,\nu)}(A) =| \Pr &\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot)} \Rightarrow 1\right] && (13.1)\\
- \Pr &\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot)} \Rightarrow 1\right] && (13.2)\\
+ \Pr &\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{E}_K(\cdot,\cdot),\perp(\cdot,\cdot)} \Rightarrow 1\right] && (13.3)\\
- \Pr &\left[A^{\$^{\omega}(\cdot,\cdot),\perp(\cdot,\cdot)} \Rightarrow 1\right] |. && (13.4)
\end{aligned}$$

One can interpret (13.1)/(13.2) as the advantage that an adversary has on the integrity of the ciphertext and (13.3)/(13.4) as the advantage that an CPA adversary has on the privacy. Using this decomposition as a motivational starting point, we now define ciphertext integrity and what we mean by a CPA adversary on authenticated encryption schemes. From now on, our definitions are based on the game playing methodology. For example, we can restate Definition 13.2 using the game G_{CCA3} given in Figure 13.1 as

$$\mathbf{Adv}_{\Pi}^{\mathrm{CCA3}(\omega,\nu)}(A) = 2|\Pr[A^{G_{\mathrm{CCA3}}(\omega,\nu)} \Rightarrow 1] - 0.5|.$$

We denote $\mathbf{Adv}_{\Pi}^{\mathrm{CCA3}(\omega,\nu)}(q,t,\ell)$ as the maximum advantage over all CCA3(ω,ν) adversaries run in time at most t, ask a total of q queries to $\mathcal{E}$ and $\mathcal{D}$, and whose total query length is not more than ℓ blocks.

13.2.1. Privacy and Integrity Notions for Authenticated Encryption Schemes.

Similarly, we define the privacy and integrity of an authenticated (on-line) encryption scheme $\Pi = (\mathcal{K},\mathcal{E},\mathcal{D})$ with header space D_n^+, message space D_n^*, and tag-size function $\mathrm{TAG}(H,M)$ as follows.

Definition 13.3. *Let $G_{CPA}(\omega,\nu)$ be the $CPA_{\Pi}^{(\omega,\nu)}$ game given in Figure 13.1. Fix an adversary A. The advantage of A breaking Π is defined as*

$$\mathbf{Adv}_{\Pi}^{CPA(\omega,\nu)}(A) = 2|\Pr[A^{G_{CPA}(\omega,\nu)} \Rightarrow 1] - 0.5|.$$

Definition 13.4. *Let $G_{\text{INT-CTXT}}(\nu)$ be the $\text{INT-CTXT}_{\Pi}^{\nu}$ game given in Figure 13.2. Fix an adversary A. The advantage of A breaking Π is defined as*

$$\mathbf{Adv}_{\Pi}^{\text{INT-CTXT}(\nu)}(A) = \Pr[A^{G_{\text{INT-CTXT}}(\nu)} \Rightarrow 1].$$

We denote $\mathbf{Adv}_{\Pi}^{CPA(\omega,\nu)}(q,t,\ell)$ and $\mathbf{Adv}_{\Pi}^{\text{INT-CTXT}(\nu)}(q,t,\ell)$ as the maximum advantage over all $CPA(\omega,\nu)$ resp. INT-CTXT(ν) adversaries run in time at most t, ask a total of q queries to $\mathcal{E}$ and $\mathcal{D}$, and whose total query length is not more than ℓ blocks.

13.2.2. CCA3 is equal to INT-CTXT plus CPA.

We now give a generalization of Theorem 3.2 from Bellare and Namprempre [10]. It simply states the equivalence of a scheme being CCA3 secure and both INT-CTXT and CPA secure. These statements hold in the on-line and off-line case.

Theorem 13.5. *Let $\Pi = (\mathcal{K},\mathcal{E},\mathcal{D})$ be an authenticated encryption scheme. Fix $\omega \in \{\text{AE}, \text{AOE}\}$ and $\nu \in \{\text{NR}, \text{NI}\}$. Let A be a $\text{CCA3}(\omega,\nu)_{\Pi}$ adversary running in time t, making q queries with*

```
Initialize($\nu$)
    $K \leftarrow \mathcal{K}()$;

Encrypt(H,M)
    if ($\nu = $ NR and $H_{L_H} \in B$)
        then return $\perp$;
    $C \leftarrow \mathcal{E}_K$(H,M);
    $B \leftarrow B \cup \{H_{L_H}\}$;
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(H,C)\}$;
    return $C$;

Finalize()
    return win;

Verify(H,C)
    M $\leftarrow \mathcal{D}_K$(H,C);
    if (($H,C) \notin Q$ and $M \neq \perp$)
        then win $\leftarrow$ true;
    return ($M \neq \perp$);
```

Figure 13.2.: Game $G_{INT-CTXT}(\nu)$ is the INT-CTXT$_\Pi^\nu$ game where $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. H_{L_H} denotes the last block of the header representing the nonce/initial value.

a total length of at most ℓ blocks. Then, there are a CPA(ω,ν) *adversary A_p and an* INT-CTXT(ν) *adversary A_c such that*

$$\mathbf{Adv}_\Pi^{\mathrm{CCA3}(\omega,\nu)}(A) \leq \mathbf{Adv}_\Pi^{\mathrm{CPA}(\omega,\nu)}(A_p) + \mathbf{Adv}_\Pi^{\mathrm{INT\text{-}CTXT}(\nu)}(A_c).$$

Furthermore, A_c and A_p run in time $O(t)$ and both make at most q queries in each case.

Proof. Consider games G_0, G_1 and G_2 of Figure 13.3. For a fixed CCA3(ω,ν) adversary A on the scheme Π it holds that

$$\begin{aligned} \Pr[A_\Pi^{\mathrm{CCA3}(\omega,\nu)} \Rightarrow 1] &= \Pr[A^{G_0} \Rightarrow 1] \\ &= \Pr[A^{G_1} \Rightarrow 1] + (\Pr[A^{G_0} \Rightarrow 1] - \\ &\qquad \Pr[A^{G_1} \Rightarrow \texttt{true}]) \\ &\leq \Pr[A^{G_1} \Rightarrow 1] + \Pr[A^{G_1} \textit{sets } \texttt{bad}]. \end{aligned}$$

Since the `Decrypt` oracles of G_1 and G_2 always return $\perp$,

$$\Pr[A^{G_1} \Rightarrow 1] = \Pr[A^{G_2} \Rightarrow 1].$$

Now, we design two adversaries A_c and A_p so that

$$\Pr[A^{G_1} \mathit{sets}\ \mathtt{bad}] \leq \Pr[A_c{}_{\Pi}^{\text{INT-CTXT}(\nu)} \Rightarrow 1] \text{ and}$$
$$\Pr[A^{G_2} \Rightarrow 1] \leq \Pr[A_p{}_{\Pi}^{\text{CPA}(\omega,\nu)} \Rightarrow 1].$$

A_p**:** Adversary A_p simply runs A answering A's **Encrypt** queries using its own **Encrypt** oracle, and answers **Decrypt** queries with $\perp$. A_p outputs whatever A outputs.

A_c**:** Adversary A_c runs A answering A's **Encrypt** queries using its own **Encrypt** oracle. It submits A's **Decrypt** queries to it's **Verify** oracle and, regardless of the response, returns $\perp$. Note that the **Verify** oracle sets `win` to **true** if and only if a fresh **Decrypt** query is valid. Just such a query would set the variable `bad` to **true**. □

13.3. The McOEx Scheme

In this section, we give an analysis of the McOEx scheme introduced by Figure 11.1 and Table 11.4

We prove that McOEx actually achieves our two-fold goal. First, it guarantees a certain minimum, well defined, security against a nonce-ignoring adversary. And, second, we argue that McOEx is fully secure against a nonce-respecting adversary.

Definition 13.6 (McOEx). *Let $k, n \in \mathbb{N}$, $E \in \text{Block}(n,n)$ and $\varphi : \{0,1\}^k \times \{0,1\}^v \rightarrow \{0,1\}^k$ such that $\varphi(K,\cdot)$ is injective. The encryption function takes a header $H \in D_n^{L_H}$, a message M, and returns a ciphertext C and a tag $T \in D_n$. The decryption function*

```
Initialize($\omega, \nu$)
  $b \xleftarrow{\$} \{0,1\}$;
  if($b = 1$) then
    $K \leftarrow \mathcal{K}()$;

Encrypt($H, M$)
  if ($\nu = $ NR and $H_{L_H} \in B$) then
    return $\perp$;
  else
    $B \leftarrow B \cup \{H_{L_H}\}$;
  if($b = 1$) then
    $C \leftarrow \mathcal{E}_K$(H,M);
  else
    $C \leftarrow \$^{\omega}$(H,M);
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(H, C)\}$;
  return $C$;
```

```
Finalize($d$)
  return ($d = b$);

Decrypt($H, C$) Game $G_0, \boxed{G_1}$
  $M \leftarrow \perp$;
  if (($(H, C) \notin \mathcal{Q}$ and b=1) then
    $M \leftarrow \mathcal{D}_K$(H,C);
  if($M \neq \perp$) then
    bad $\leftarrow$ true; $\boxed{M \leftarrow \perp;}$
  return M;

Decrypt($H, C$) Game $G_2$
  return $\perp$;
```

Figure 13.3.: Games G_0, G_1, and G_2 for the proof of Theorem 13.5. Game G_1 contains the code in the box while G_0 does not. H_0 denotes the first block of the header representing the nonce/initial value.

takes a header $H \in D_n^{L_H}$, a ciphertext C and a tag $T \in D_n$ and returns either a plaintext M or the fail symbol $\perp$. Let $M, C \in D_N^L$ for some integer L, then the OAEAD McOEx *scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is defined by the encryption algorithm $\mathcal{E}$ as **EncryptAuthenticate**, the decryption algorithm $\mathcal{D}$ as **DecryptAuthenticate** – both given in Table 11.4 – and a key derivation function $\mathcal{K}$.*

We now proceed to show the security of McOEx. For this we use the results of Theorem 13.5 and show the INT-CTXT and CPA-PRP security separately.

Theorem 13.7. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a* McOEx *scheme as in*

Definition 13.6. Then,

$$\mathbf{Adv}_{\Pi}^{\text{CCA3(AOE,NI)}}(q,\ell,t) \leq \frac{2(q+\ell)(q+\ell+1)+3q+2\ell}{2^k-(q+\ell)} + 2\mathbf{Adv}_{E}^{\text{CCA-PRP}}(q+\ell).$$

Proof. The proof follows from Theorem 13.5 together with Lemmas 13.8 and 13.9 by simple adding up the individual bounds. □

Lemma 13.8. *Let* $\Pi = (\mathcal{K},\mathcal{E},\mathcal{D})$ *be a* McOEx *scheme as in Definition 13.6 Let* q *be the number of total queries an adversary* A *is allowed to ask and* ℓ *be an integer representing the total length in blocks of the queries to* $\mathcal{E}$ *and* $\mathcal{D}$*. Then,*

$$\mathbf{Adv}_{\Pi}^{\text{INT-CTXT(NI)}}(q,\ell,t) \leq \frac{(q+\ell)(q+\ell+1)}{2^n-(q+\ell)} + \frac{2q+\ell}{2^n-(q+\ell)} + \mathbf{Adv}_{E}^{\text{CCA-PRP}}(q+\ell).$$

Proof. Our bound is derived by game playing arguments. Consider games G_1-G_3 of Figure 13.4 and a fixed adversary A asking at most q queries with a total length of at most ℓ blocks. The functions **Initialize** and **Finalize** are identical for all games in this proof. Lets denote G_0 as the Game INT-CTXT(NI) as defined in Figure 13.2. Definition 13.4 states that

$$\mathbf{Adv}_{\Pi}^{\text{INT-CTXT(NI)}}(A) \leq \Pr[A^{G_0} \Rightarrow 1].$$

In G_1, the encryption and verify placeholders are replaced by their specific McOEx counterparts as of Definition 13.6. Clearly, $\Pr[A^{G_0} \Rightarrow$

```
Initialize()
    $K \xleftarrow{\$} \mathcal{K}()$;
    $B \leftarrow \varphi(K, 0^n)$;

Finalize()
    return win;

Encrypt$(H, M)$ Game $G_1$
    $L_H \leftarrow |H|/n$;  $L \leftarrow |M|/n$;
    $U \leftarrow \varphi(K, 0^n)$;
    for $i = 1, \ldots, L_H$ do
        $\tau \leftarrow E_U(H_i)$;
        $U \leftarrow \varphi(K, H_i \oplus \tau)$;
    for $i = 1, \ldots, L$ do
        $C_i \leftarrow E_U(M_i)$;
        $U \leftarrow \varphi(K, C_i \oplus M_i)$;
    $T \leftarrow E_U(\tau)$;
    $\mathcal{Q} \leftarrow (H, M, C, T)$;
    return $(C_1, \ldots, C_L, T)$;

Verify$(H, C, T)$ Game $G_1$
    $L_H \leftarrow |H|/n$;  $L \leftarrow |C|/n$;
    $U \leftarrow \varphi(K, 0^n)$;
    for $i = 1, \ldots, L_H$ do
        $\tau \leftarrow E_U(H_i)$;
        $U \leftarrow \varphi(K, H_i \oplus \tau)$;
    for $i = 1, \ldots, L$ do
        $M_i \leftarrow E_U^{-1}(C_i)$;
        $U \leftarrow \varphi(K, C_i \oplus M_i)$;
    if $(T = E_U(\tau)$ and
     $(H, C) \notin \mathcal{Q}_{|H,C})$ then
        win $\leftarrow$ true;
    $\mathcal{Q} \leftarrow (H, \bot, C, \bot)$;
    return $(T = E_U(\tau))$;

Encrypt$(H, M)$ Game $G_2$, $\boxed{G_3}$
    $L_H \leftarrow |H|/n$;  $L \leftarrow |M|/n$;
    $A \leftarrow A \cup H$;
    $p \leftarrow \mathrm{LLCP}_n(\mathcal{Q}_{|H,M}, (H, M))$;
    $U \leftarrow \varphi(K, 0^n)$;
    for $i = 1, \ldots, L_H$ do
        $\tau \leftarrow E_U(H_i)$;
        $U \leftarrow \varphi(K, H_i \oplus \tau)$;
        if $(U \in B$ and $i > p)$ then
            bad $\leftarrow$ true;
            $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
        $B \leftarrow B \cup U$;
    for $i = 1, \ldots, L$ do
        $C_i \leftarrow E_U(M_i)$;
        $U \leftarrow \varphi(K, C_i \oplus M_i)$;
        if $(U \in B$ and $i + L_H > p)$
        then
            bad $\leftarrow$ true;
            $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
        $B \leftarrow B \cup U$;
    $T \leftarrow E_U(\tau)$;
    $\mathcal{Q} \leftarrow (H, M, C, T)$;
    return $(C_1, \ldots, C_L, T)$;

Verify$(H, C, T)$ Game $G_2$, $\boxed{G_3}$
    $L_H \leftarrow |H|/n$;  $L \leftarrow |C|/n$;
    $p \leftarrow \mathrm{LLCP}_n(\mathcal{Q}_{|H,M}, (H, M))$;
    $U \leftarrow \varphi(K, 0^n)$;
    for $i = 1, \ldots, L_H$ do
        $\tau \leftarrow E_U(H_i)$;
        $U \leftarrow \varphi(K, H_i \oplus \tau)$;
        if $(U \in B$ and $i > p)$ then
            bad $\leftarrow$ true;
            $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
        $B \leftarrow B \cup U$;
    for $i = 1, \ldots, L - 1$ do
        $M_i \leftarrow E_U^{-1}(C_i)$;
        $U \leftarrow \varphi(K, C_i \oplus M_i)$;
        if $(U \in B$ and $i + L_H > p)$
            then bad $\leftarrow$ true;
            $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
        $B \leftarrow B \cup U$;
    $M_L \leftarrow E_U^{-1}(C_L)$;
    $U \leftarrow \varphi(K, C_L \oplus M_L)$;
    if $(U \in B$ and $H \notin A)$ then
        bad $\leftarrow$ true;
        $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
    if $(T = E_U(\tau)$ and
     $(H, C, T) \notin \mathcal{Q}_{|H,C,T})$ then
        win $\leftarrow$ true;
    $\mathcal{Q} \leftarrow (H, \bot, C, \bot)$;
    $B \leftarrow B \cup U$;
    return $(T = E_U(\tau))$;
```

Figure 13.4.: Games G_1-G_3 for the proof of Lemma 13.8. Game G_3 contains the code in the box while G_2 does not.

1] = $\Pr[A^{G_1} \Rightarrow 1]$. We now discuss the differences between G_1 and G_2. The set B is initialized to $\{\varphi(K, 0^n)\}$ and then collects new key-input values U which are computed during the encryption or verification process (in lines 204, 207, 213, 226, 229, 235, and 241). We note that, since φ is injective, a collision for the chaining values follows if there is a collision in the U values.

In lines 203 and 225, the function LLCP$_n$ is invoked. Finally, the variable bad is set to true if one of the if-conditions in lines 208, 215, 230, 237, or 243 is true. *None* of these modifications affect the values returned to the adversary and therefore,

$$\Pr[A^{G_1} \Rightarrow 1] = \Pr[A^{G_2} \Rightarrow 1].$$

For our further discussion, we require another game G_4 which is explained in more detail later in this proof.[1] It follows that

$$\begin{aligned}\Pr[A^{G_2} \Rightarrow 1] &= \Pr[A^{G_3} \Rightarrow 1] + |\Pr[A^{G_2} \Rightarrow 1] - \Pr[A^{G_3} \Rightarrow 1| \\ &\leq \Pr[A^{G_3} \Rightarrow 1] + \Pr[A^{G_3} \mathit{sets}\ \texttt{bad}] \\ &\leq \Pr[A^{G_4} \Rightarrow 1] + |\Pr[A^{G_3} \Rightarrow 1] - \Pr[A^{G_4} \Rightarrow 1]| \\ &\quad + \Pr[A^{G_3} \mathit{sets}\ \texttt{bad}] \qquad (13.5)\end{aligned}$$

We now proceed to upper bound any of the three terms contained in (13.5) – in right to left order. The success probability of game G_3 does not differ from the success probability of G_2 unless a chaining value U occurs twice. In this case, the adversary must (i) either have 'found' a collision for $E_{\varphi(K,X)}(Y) \oplus Y$, *i.e.*, it stumbles over (X, Y) and (X', Y') such that $E_{\varphi(K,X)}(Y) \oplus Y = E_{\varphi(K,X')}(Y') \oplus Y'$ or (ii) must have found a preimage of $\varphi(K, 0^n)$, which is always the starting point of our chain. Note that the value $\varphi(K, 0^n)$ is initially stored in the set B. In both cases, the variable bad would have been set to true, and it follows by [19] that

$$\Pr[A^{G_3} \mathit{sets}\ \texttt{bad}] \leq \frac{(q+\ell)(q+\ell+1)}{2^n - (q+\ell)} + \frac{q+\ell}{2^n - (q+\ell)}$$

[1] Since the difference is very minor, we do not provide an extra figure.

We now describe the new game G_4. It is equal to G_3 *except* that the blockcipher E (and E^{-1}) is replaced by a randomly chosen function **EncryptBlock** (and **DecryptBlock**), which is modeled as a pseudo random permutation (PRP). We assume that they are implemented via lazy sampling. More precisely, the call $E_K(A)$ is replaced by an invocation of $\mathbf{EncryptBlock}_K(A)$ and the call $E_K^{-1}(A)$ is replaced by an invocation of $\mathbf{DecryptBlock}_K(A)$. We now upper bound the difference between G_3 and G_4. So, by definition of G_4, we have

$$|\Pr[A^{G_3} \Rightarrow 1] - \Pr[A^{G_4} \Rightarrow 1]| \leq \mathbf{Adv}_E^{\text{CCA-PRP}}(q + \ell).$$

Finally, we have to upper bound the advantage of the adversary A to win the game G_4. A can only win this game if the condition in line 238 (resp. 438 for game G_4) is `true`. As usual, we assume *wlog.* that A does not ask a question if the answer is already known, which implies that $(H, C, T) \notin \mathcal{Q}_{|H,C,T}$. For our analysis we distinguish between three cases. So we formally adjust line 240 (*i.e.* choose as the tag computation operation either E or E^{-1}) such that we always have enough randomness left for our result.

Case 1: *H has already been used for an Encrypt or Verify query before and $U \in B$.* Since we already have computed τ in the past, the chance of success is upper bounded by $\Pr[E_U^{-1}(T) = \tau]$ which can be upper bounded by $1/(2^n - (q + \ell))$.

Case 2: *H has never been used before, also U has never been used as a chaining value.* Then, the tagging operation uses a 'new key' – essentially since φ is injective – and therefore, the output of $E_U(\tau)$ is uniformly distributed and the success probability is $\leq 1/2^n$.

Case 3: *$H \in A$, but U has never been used as a chaining value.* The chance of success is upper bounded by $\Pr[E_U^{-1}(T) = \tau]$ which can be upper bounded by $1/2^n$.

Note that the 'missing' fourth case has been explicitly excluded by line 240 (resp. 440). Since these three cases are mutually exclusive, we can upper bound the success probability for q queries as

$$\Pr[A^{G_4} \Rightarrow 1]| \leq \frac{q}{2^n - (q + \ell)}.$$

Our claim follows by adding up the individual bounds. □

Lemma 13.9. *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be a* McOEx *scheme as in Definition 13.6. Let q be the number of total queries an adversary A is allowed to ask and ℓ be an integer representing the total length of the queries to $\mathcal{E}$ and $\mathcal{D}$. Then,*

$$\mathbf{Adv}_{\Pi}^{\mathrm{CPA(\textsc{aoe,ni})}}(q, \ell, t) \leq \frac{(q+\ell)(q+\ell+1)}{2^n - (q+\ell)} + \frac{q+\ell}{2^n - (q+\ell)} + \mathbf{Adv}_{E}^{\mathrm{CPA\text{-}PRP}}(q+\ell).$$

Proof. Our bound is derived by game playing arguments. Consider games G_1 and G_2 of Figure 13.5. The functions **Initialize** and **Finalize** are identical for any of those games.

At first we investigate the differences between the CPA(AOE, NI) game from Figure 13.1 and G_1 from Figure 13.5. In G_1 we have replaced $\mathcal{E}$ by its definition of McOEx and $\$^w$ by an on-line encryption oracle **OnlinePermutation** (line 102) that just models a 'perfect' OPRP, *i.e.*, for two plaintexts with an equal prefix it returns two ciphertexts that also share a prefix of the same length. We again assume this oracle to be implemented by lazy sampling. Then, set B does only collect new chaining values (lines 115 and 123) in order to intercept the occurrence of two equal chaining values which do lead – due to the injectivity of φ – to two equal keys for the encryption of a block.

In line 105, the oracle LLCP$_n$ is invoked returning the length of the longest common prefix of (H, M) and $\mathcal{Q}_{|H,M}$. Finally, the variable `bad` is set to `true` if (one of) the conditions of lines 111/211 or 119/219 are both `true`. These changes do not affect the success probability of an adversary, because the output of the oracle remains unchanged. More precisely, the distribution of the output does not change. This means that – using a new game G_3 described shortly –

$$\mathbf{Adv}_{\Pi}^{\text{CPA(AOE, NI)}}(A) = 2 \cdot |\Pr[A^{G_1} \Rightarrow 1] - 0.5|.$$

Therefore, by common game playing arguments,

$$\begin{aligned}\Pr[A^{G_1} \Rightarrow 1] &= \Pr[A^{G_2} \Rightarrow 1] + |\Pr[A^{G_1} \Rightarrow 1] - \Pr[A^{G_2} \Rightarrow 1] \\ &\leq \Pr[A^{G_2} \Rightarrow 1] + \Pr[A^{G_2} \textit{sets } \mathtt{bad}] \\ &\leq \Pr[A^{G_3} \Rightarrow 1] + |\Pr[A^{G_2} \Rightarrow 1] - \Pr[A^{G_3} \Rightarrow 1]| \\ &\qquad + \Pr[A^{G_2} \textit{sets } \mathtt{bad}].\end{aligned}$$

The success probability of game G_2 does not differ from the success probability of G_1 unless a chaining value U occurs twice. In this case, the adversary must either have found a collision for $E_{\varphi(K,A)}(B) \oplus B$, *i.e.*, it has found (A, B) and (A', B') such that $E_{\varphi(K,A)}(B) \oplus B = E_{\varphi(K,A')}(B') \oplus B'$ or must have found a preimage of $\varphi(K, 0^n)$. In both cases, the variable `bad` would have been set to `true`, and it follows again by [19] that

$$\Pr[A^{G_2} \textit{sets } \mathtt{bad}] \leq \frac{(q+\ell)(q+\ell+1)}{2^n - (q+\ell)} + \frac{q+\ell}{2^n - (q+\ell)},$$

The aforementioned new game G_3 is equal to the game G_2 *except* that the blockcipher E (and E^{-1}) is replaced by a randomly chosen function **EncryptBlock** (and **DecryptBlock**), which is modeled as a pseudo random permutation (PRP). We assume that they are implemented via lazy sampling. More precisely, the call $E_K(A)$ is replaced by an invocation of $\mathbf{EncryptBlock}_K(A)$ and the call $E_K^{-1}(A)$ is replaced by an invocation of $\mathbf{DecryptBlock}_K(A)$. We now upper bound the difference between G_2 and G_3. So, by definition of G_4, we have

$$|\Pr[A^{G_2} \Rightarrow 1] - \Pr[A^{G_3} \Rightarrow 1]| \leq \mathbf{Adv}_{E}^{\text{CPA-PRP}}(q+\ell).$$

```
1   Initialize()
2       $b \xleftarrow{\$} \{0,1\}$;
3       $K \xleftarrow{\$} \mathcal{K}()$;
4       $B \leftarrow \{\varphi(K, 0^n)\}$;

5   Finalize(d)
6       return $(b = d)$;

100 Encrypt$(H, M)$ Game $G_1, \boxed{G_2}$
101     if $(b = 0)$ then
102         $C \leftarrow$ OnlinePermutation$(H, M)$;
103     else
104         $L_H \leftarrow |H|/n$;  $L \leftarrow |M|/n$;
105         $p \leftarrow \mathrm{LLCP}_n(\mathcal{Q}, (H, M))$;
106         $\mathcal{Q} \leftarrow \mathcal{Q} \cup (H, M)$;
107         $U \leftarrow \varphi(K, 0^n)$;
108         for $i = 1, \ldots, L_H$ do
109             $\tau \leftarrow E_U(H_i)$;
110             $U \leftarrow \varphi(K, H_i \oplus \tau)$;
111             if $(U \in B$ and $i > p)$
112             then
113                 bad $\leftarrow$ true;
114                 $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
115             $B \leftarrow B \cup U$;
116         for $i = 1, \ldots, L$ do
117             $C_i \leftarrow E_U(M_i)$;
118             $U \leftarrow \varphi(K, C_i \oplus M_i)$;
119             if $(U \in B$ and $i + L_H > p)$
120             then
121                 bad $\leftarrow$ true;
122                 $\boxed{U \xleftarrow{\$} \{0,1\}^n \setminus B;}$
123             $B \leftarrow B \cup U$;
124         return $C$;
```

Figure 13.5.: Games G_1 and G_2 for the proof of Lemma 13.9. Game G_2 contains the code in the box while G_1 does not. If G_2 is addressed, the numbering changes to $2xx$.

Finally, we have to upper bound the advantage for an adversary A to win the game G_3. Since the values of U cannot 'collide' and it is not possible to compute a preimage for any query, the algorithm for $b = 0$ *is* an OPRP, and therefore, the success probability to win G_3 for any adversary is 0.5, *i.e.*, it has no advantage in winning this game. □

Conclusions

We now conclude with a commented summary of the thesis contributions and some research outlook.

Summary

Blockcipher Based Double Length Hashing One of the main topics of this thesis has been the analysis and design of blockcipher based double length hash functions. During our research we helped establishing security bounds for three of the for classic constructions (Abreast-DM, Tandem-DM, MDC-2, MDC-4).

As we started our research in 2008, there had been surprisingly little public research on this topic. Apart from the initial introduction of Abreast-DM and Tandem-DM at Eurocrypt 1992 – where the authors claimed that they had not found any attacks on them – no security results were known. But, as experience tells for double length hash functions, this seemed to be a good sign, since attacks are not known either. This is especially true as a lot of blockcipher based hash function proposals – most of them have been designed in the early 90s – have been successfully attacked since at that time providing security proofs was rather uncommon. Our security analysis of Abreast-DM and Tandem-DM and its generalizations like Cyclic-DL, Serial-DL, and Generic-DL helped closing this gap. Besides establishing new security bounds for known constructions, we also introduced new designs, *e.g.*, Cube-DM, Add/k-DM, and Weimar-DM, that still are *the* blockcipher based double length double call compression functions with the best collision security bounds known in literature.

For MDC-4, the story is somewhat similar. It was made openly available in 1988. But, in contrast to Abreast-DM and Tandem-DM, it faced some attack-based cryptanalysis. Motivated by the analysis of MDC-2, we applied these techniques to MDC-4 and established a quite weak, but still non-trivial, collision security bound.

Ideal World Model for Hash Functions In the first two parts of this thesis, we tried to build good compression functions from 'ideal' ciphers inside. In the third part, we discuss the problem of building a 'good' hash function from 'ideal' compression functions. Although, being somewhat similar at this level of abstraction, we here have added complexity as a hash function must be able to deal with inputs of arbitrary size.

Trying to answer one of the more pressing questions in hash function research, 'when is an iterated hash function perfect', the notion of *indifferentiability of a hash function* was established by Dodis *et al.* But although researchers celebrated this as a huge breakthrough (and for sure it is) we have shown that security proofs in cryptography are quite special in nature. It is another example that it is not enough to prove that a (secret key) scheme is secure but one also has to take other factors into account. For example, depending on the level of abstraction, we can prove one and the same practical scheme secure and insecure. And choosing that 'right' level of abstraction is still more a matter of feeling and experience, *i.e.*, an art, than a science.

On-Line Authenticated Encryption The second main topic of this thesis, presented in the fourth part, is authenticated encryption. We introduced a new scheme called McOEx that is the first dedicated scheme in the pool of tenths of AE schemes that combines resistance against nonce-misuse and the ability of being on-line computable. While no scheme can cope with all possible misuse scenarios available to a novice application developer, it seems to cover a lot of practically relevant cases by still offering very pleasant performance.

Further Research

It is fair to say that the research for double length double call hash functions using block ciphers able to compress a $3n$-bit string to an n-bit string arrived at a mature state. But other blockcipher based compression function scenarios are much less clear. Some examples are:

- There is *no* blockcipher based double length compression function known using a blockcipher from $\mathrm{Block}(n, n)$ that comes with a decent collision security bound.
- Taken a blockcipher like AES-192, *i.e.*, from $\mathrm{Block}(3/2n, n)$, how can these types of ciphers impact efficient compression function design?
- There are virtually no results in literature that analyze multi-length scenarios. Stam gave a multi-length framework in [133] that might serve as a starting point. Such an analysis should help understand the interaction between the single components better.

In the field of hash function indifferentiability, the framework establishes itself as the defacto gold standard for *secure hash functions*.

Even though the random oracle model is very convenient for finding security proofs, further research should be undertaken especially for security definitions *relying* on random oracles. Since, in that case, even the *security definition* itself gets meaningless if the random oracle world is left.

Authenticated encryption is forecasted a vivid research future[2] as there are a lot of practically relevant issues that still need to be resolved. First of all, there seems to be some agreement that authenticated encryption truly *is* the right notion of security for a vast majority of communication issues. Based on this, some of the questions that arise are:

- Is it possible to design an integrated AE scheme that is more efficient than the combined ones currently discussed in cryptography?
- What about the hardware efficiency of such a combined scheme?
- Is it possible to design an AE scheme that can also be used as a *hash function*, *e.g.*, by simply discarding the ciphertext outputs and only taking the *tag* value? This might be one of the big steps what cryptography can do for the people in the future.
- How can on-line computability be handled efficiently in an AE scheme without neglecting security standards?
- Are there, apart from nonce-reuse, other relevant misuse scenarios research should take a look at?
- More generally, the issue of nonce-reuse should be discussed more thoroughly (in the secret-key community, as well as in the public-key community).

[2] Dagstuhl Seminar of Symmetric Cryptography 2012

Notations

List of Abbreviations

AES	**A**dvanced **E**ncryption **S**tandard
CF	**C**ompression **F**unction
DES	**D**ata **E**ncryption **S**tandard
DL	**D**ouble (Block) **L**ength
DM	**D**avies-**M**eyer
HF	**H**ash **F**unction
HMAC	**H**ash-based **MAC**
iff	**if** and only **if**
MAC	**M**essage **A**uthentication **C**ode
MD	**M**erkle-**D**amgård
MMO	**M**atyas-**M**eyer-**O**seas
NMAC	**N**ested **MAC**
MP	**M**iyaguchi-**P**reneel
SL	**S**ingle (Block) **L**ength

List of Symbols

$\|\|$	concatenation of bit strings, *e.g.*, $A \in \{0,1\}^a, B \in \{0,1\}^b$, then $A\|\|B$ denotes the concatenated string $A\|\|B \in \{0,1\}^{a+b}$
$\circ$	in formulas: Composition of functions, in figures: bit-by-bit complement
$\overline{A}$	bit-by-bit complement of the binary string A
$\mathrm{K} \overset{\$}{\leftarrow} \{0,1\}^k$	K is selected uniformly at random from the set $\{0,1\}^k$
$A \boxplus B$	modular addition of A and B
$A \oplus B$	exclusive-or (XOR) of A and B
$\|A\|$	bit length of variable A
N	2^n
N'	$2^n - q$
N''	$2^n - 2q$

List of Publications

The lists are ordered by the date of publication.

Lecture Notes in Computer Science

1. **Ewan Fleischmann**, Michael Gorski, and Stefan Lucks: *On the Security of Tandem- DM.* Orr Dunkelman (Editor): Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven (Belgium), Lecture Notes in Computer Science 5665, pages 84 - 103, Springer, 2009. [60]

2. **Ewan Fleischmann**, Christian Forler, Michael Gorski, and Stefan Lucks: *Twister - A Framework for Secure and Fast Hash Functions.* Bao *et al.* (Editors): Information Security Practice and Experience, 5th International Conference, ISPEC 2009, Xi'an (China), Lecture Notes in Computer Science 5451, pages 257 - 273, Springer, 2009. [46]

3. **Ewan Fleischmann**, Michael Gorski, and Stefan Lucks: *Memoryless Related-Key Boomerang Attack on 39-Round SHACAL-2.* Bao *et al.* (Editors): Information Security Practice and Experience, 5th International Conference, ISPEC 2009, Xi'an (China), Lecture Notes in Computer Science 5451, pages 310 - 323, Springer, 2009. [58]

4. **Ewan Fleischmann**, Michael Gorski, and Stefan Lucks: *Memoryless Related-Key Boomerang Attack on the Full Tiger Blockcipher.* Bao *et al.* (Editors): Information Security Practice and Experience, 5th International Conference, ISPEC 2009, Xi'an (China), Lecture Notes in Computer Science 5451, pages 298 - 309, Springer, 2009. [59]

5. **Ewan Fleischmann**, Michael Gorski, and Stefan Lucks: *Attacking 9 and 10 Rounds of AES-256.* C. Boyd and J. Gonzàlez Nieto (Editors), Information Security and Privacy, 14th Australasian Conference, ACISP 2009, Brisbane (Australia), Lecture Notes in Computer Science 5594, pages 60 - 72, Springer, 2009. [57]

6. **Ewan Fleischmann**, Michael Gorski, and Stefan Lucks: *Security of Cyclic Double Block Length Hash Functions.* M. G. Parker (Editor), 12th IMA International Conference, Cryptography and Coding 2009, Cirencester (UK), Lecture Notes in Computer Science 5921, pages 153 - 175, Springer, 2009. [61]

7. Orr Dunkelmann, **Ewan Fleischmann**, Michael Gorski, and Stefan Lucks: *Related- Key Rectangle Attack of the Full 80-Round HAS-160 Encryption Mode.* In R. Roy and N. Sendrier (Editors), Progress in Cryptology - INDOCRYPT 2009, 10th International Conference on Cryptology in India, New Delhi (India), Lecture Notes in Computer Science 5922, pages 157 - 168, Springer, 2009. [39]

8. **Ewan Fleischmann**, Michael Gorski, Jan-Hendrik Hühne, and Stefan Lucks: *Key Recovery Attack on full GOST Blockcipher*

with Negligible Time and Memory. Proceedings of the Second Western European Workshop on Research in Cryptology, WE-WoRC 2009, Graz (Austria), to appear in Lecture Notes in Computer Science. [55]

9. **Ewan Fleischmann**, Michael Gorski, and Stefan Lucks: *Some observations on Indifferentiability.* Information Security and Privacy, 15th Australasian Conference, ACISP 2010, Sydney (Australia), Lecture Notes in Computer Science 6168, pages 117 - 134, Springer 2010. [62]

10. **Ewan Fleischmann**, Christian Forler, Michael Gorski, and Stefan Lucks: *Collision Resistant Double-Length Hashing.* The 4th International Conference on Provable Security, ProvSec 2010, Malacca (Malaysia), Lecture Notes in Compute Science 6402, pages 102 - 118, Springer, 2010. [47]

11. **Ewan Fleischmann**, Christian Forler, Michael Gorski, and Stefan Lucks: *New Boomerang Attacks on ARIA.* Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad (India), Lecture Notes in Computer Science 6498, pages 163 - 175, Springer, 2010. [48]

12. **Ewan Fleischmann**, Christian Forler, and Stefan Lucks: *Gamma-MAC[H,P] - A new universal MAC scheme.* Proceedings of the Third Western European Workshop on Research in Cryptology, WEWoRC 2011, Weimar (Germany), Lecture Notes in Computer Science 7242, pages 83-98, Springer 2012, [50]

13. Frederik Armknecht, **Ewan Fleischmann**, Matthias Krause, Jooyoung Lee, Martijn Stam, and John P. Steinberger: *The Preimage Security of Double-Block-Length Compression Functions.* Dong Hoon Lee, Xiaoyun Wang (Eds.): Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul (South Korea), Lecture Notes in Computer Science 7073, pages 233 - 251, Springer, 2011. [3]

14. **Ewan Fleischmann**, Christian Forler, Michael Gorski, and Stefan Lucks: *McOE – A Family of Almost Foolproof On-line Authenticated Encryption Schemes.* Fast Software Encryption, 19th International Workshop, FSE 2012, Washington DC (United States of America), Lecture Notes in Computer Science 7549, pages 196-215, Springer, 2012. [53]

15. **Ewan Fleischmann**, Christian Forler, and Stefan Lucks: *On the Collision Security of* MDC-4. Progress in Cryptology - AFRICACRYPT 2012 - 5th International Conference on Cryptology in Africa, Ifrane (Morocco), Lecture Notes in Computer Science 7374, pages 252-269, Springer, 2012. [52]

16. **Ewan Fleischmann**, Christian Forler, Stefan Lucks, and Jakob Wenzel: *Weimar-DM – A Secure Double Length Compression Function.* Information Security and Privacy, 17th Australasian Conference, ACISP 2012, Wollongong (Australia), Lecture Notes in Computer Science 7372, pages 152-165, Springer, 2012 [54]

International Publications in Journals

1. **Ewan Fleischmann**, Christian Forler, Michael Gorski, and Stefan Lucks: *Twister$_\pi$ - a framework for secure and fast hash functions.* International Journal of Applied Cryptography (IJACT) Volume 2 Number 1, pages 68 - 81, Inderscience, 2010. [49]

Further International Publications

1. **Ewan Fleischmann**, Christian Forler, and Michael Gorski: *The Twister Hash Function Family*, Submission for the NIST SHA-3 Competition. Presented at NIST first hash function workshop, Leuven, 2009. Available at `http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/submissions_rnd1.html`. [56]

2. **Ewan Fleischmann**, Christian Forler, and Michael Gorski: *Classification of SHA-3 Candidates*, IACR Cryptology ePrint, 2008/511. [45]

3. Matthias Krause, Frederik Armknecht, and **Ewan Fleischmann**: *Preimage Resistance Beyond the Birthday Bound: Double-Length Hashing Revisited*, IACR Cryptology ePrint, 2010/519. [99]

4. **Ewan Fleischmann**, Christian Forler, Stefan Lucks, and Jakob Wenzel: *McOE – A Family of Almost Foolproof On-line Authenticated Encryption Schemes*, full version of [53], IACR Cryptology ePrint, 2011/644. [51]

Bibliography

[1] Elena Andreeva, Gregory Neven, Bart Preneel, and Thomas Shrimpton. Seven-Property-Preserving Iterated Hashing: ROX. In Kurosawa [101], pages 130–146.

[2] Elena Andreeva and Bart Preneel. A Three-Property-Secure Hash Function. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 228–244. Springer, 2008.

[3] Frederik Armknecht, Ewan Fleischmann, Matthias Krause, Jooyoung Lee, Martijn Stam, and John P. Steinberger. The Preimage Security of Double-Block-Length Compression Functions. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 233–251. Springer, 2011.

[4] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A Family of Fast Syndrome Based Cryptographic Hash Functions. In Ed Dawson and Serge Vaudenay, editors, *Mycrypt*, volume 3715 of *Lecture Notes in Computer Science*, pages 64–83. Springer, 2005.

[5] Feng Bao, Hui Li, and Guilin Wang, editors. *Information Security Practice and Experience, 5th International Conference, ISPEC 2009, Xi'an, China, April 13-15, 2009, Proceedings*, volume 5451 of *Lecture Notes in Computer Science*. Springer, 2009.

[6] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. Online Ciphers and the Hash-CBC Construction. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 292–309. Springer, 2001.

[7] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. Online Ciphers and the Hash-CBC Construction. Cryptology ePrint Archive, Report 2007/197; full version of [6], 2007. `http://eprint.iacr.org/`.

[8] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. In Cachin and Camenisch [26], pages 171–188.

[9] Mihir Bellare and Tadayoshi Kohno. Hash Function Balance and Its Impact on Birthday Attacks. In Cachin and Camenisch [26], pages 401–418.

[10] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[11] Mihir Bellare and Thomas Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In Lai and Chen [102], pages 299–314.

[12] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[13] Mihir Bellare and Phillip Rogaway. Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000.

[14] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX Mode of Operation. In Roy and Meier [155], pages 389–407.

[15] Kamel Bentahar, Dan Page, Markku-Juhani O. Saarinen, Joseph H. Silverman, and Nigel Smart. LASH. In *NIST: The Second Cryptographic Hash Workshop*. Online, August 2006.

[16] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In Smart [159], pages 181–197.

[17] Eli Biham and Orr Dunkelman. The SHAvite-3 Hash Function. Submission to NIST (Round 2), 2009.

[18] John Black. The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function. In Robshaw [143], pages 328–340.

[19] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Yung [169], pages 320–335.

[20] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In *MOBICOM*, pages 180–189, 2001.

[21] Joppe W. Bos, Onur Özen, and Martijn Stam. Efficient Hashing Using the AES Instruction Set. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 507–522. Springer, 2011.

[22] B. Brachtl, D. Coppersmith, M. M. Hyden, C. H. Meyer, S. M. Matyas, J. Oseas, S. Pilpel, and M. Schilling. Data authentication using modification detection codes based on a public one way encryption function. U.S. Patent No. 4,908,861, March 13, 1990.

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

[24] Lawrence Brown, Josef Pieprzyk, and Jennifer Seberry. LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications. In Jennifer Seberry and Josef Pieprzyk, editors, *AUSCRYPT*, volume 453 of *Lecture Notes in Computer Science*, pages 229–236. Springer, 1990.

[25] Enrico Buonanno, Jonathan Katz, and Moti Yung. Incremental Unforgeable Encryption. In Matsui [116], pages 109–124.

[26] Christian Cachin and Jan Camenisch, editors. *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*. Springer, 2004.

[27] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[28] Donghoon Chang, Sangjin Lee, Mridul Nandi, and Moti Yung. Indifferentiable Security Analysis of Popular Hash Functions with Prefix-Free Padding. In Lai and Chen [102], pages 283–298.

[29] Scott Contini, Arjen K. Lenstra, and Ron Steinfeld. VSH, an Efficient and Provable Collision-Resistant Hash Function. In Vaudenay [164], pages 165–182.

[30] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In Shoup [158], pages 430–448.

[31] IBM Corp. FIPS140-2 Security Policy for IBM CryptoLite in C (CLiC). October 2003. Available at `http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp356.pdf`.

[32] Intel Corporation. AES-NI Sample Library v1.2. `http://software.intel.com/en-us/articles/download-the-intel-aesni-sample-library/`, October 2010.

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

[34] Ivan Damgård. A Design Principle for Hash Functions. In Brassard [23], pages 416–427.

[35] Bert den Boer and Antoon Bosselaers. Collisions for the Compressin Function of MD5. In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 1993.

[36] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the Generic Insecurity of the Full Domain Hash. In Shoup [158], pages 449–466.

[37] Yevgeniy Dodis, Thomas Ristenpart, and Thomas Shrimpton. Salvaging Merkle-Damgård for Practical Applications. In *EUROCRYPT*, pages 371–388, 2009.

[38] Orr Dunkelman, editor. *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*. Springer, 2009.

[39] Orr Dunkelman, Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Related-Key Rectangle Attack of the Full HAS-160 Encryption Mode. In Bimal K. Roy and Nicolas Sendrier, editors, *INDOCRYPT*, volume 5922 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 2009.

[40] Morris Dworkin. *Special Publication 800-38C: Recommendation for blockcipher modes of operation: the CCM mode for authentication and confidentiality.* National Institute of Standards and Technology, U.S. Department of Commerce, May 2005.

[41] Eberhard Freitag and Rolf Busam. Complex Analysis, Springer; 1st edition, 2005.

[42] Shimon Even and Yishay Mansour. A Construction of a Cipher From a Single Pseudorandom Permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT*, volume 739 of *Lecture Notes in Computer Science*, pages 210–224. Springer, 1991.

[43] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein Hash Funciton Family. Available online at `http://www.skein-hash.info/sites/default/files/skein1.2.pdf`, February 2010.

[44] Marc Fischlin and Anja Lehmann. Multi-property Preserving Combiners for Hash Functions. In Ran Canetti, editor, *TCC*,

volume 4948 of *Lecture Notes in Computer Science*, pages 375–392. Springer, 2008.

[45] Ewan Fleischmann, Christian Forler, and Michael Gorski. Classification of the SHA-3 Candidates. Cryptology ePrint Archive, Report 2008/511, 2008. `http://eprint.iacr.org/`.

[46] Ewan Fleischmann, Christian Forler, Michael Gorski, and Stefan Lucks. Twister- A Framework for Secure and Fast Hash Functions. In Bao et al. [5], pages 257–273.

[47] Ewan Fleischmann, Christian Forler, Michael Gorski, and Stefan Lucks. Collision Resistant Double-Length Hashing. In Swee-Huay Heng and Kaoru Kurosawa, editors, *ProvSec*, volume 6402 of *Lecture Notes in Computer Science*, pages 102–118. Springer, 2010.

[48] Ewan Fleischmann, Christian Forler, Michael Gorski, and Stefan Lucks. New Boomerang Attacks on ARIA. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT*, volume 6498 of *Lecture Notes in Computer Science*, pages 163–175. Springer, 2010.

[49] Ewan Fleischmann, Christian Forler, Michael Gorski, and Stefan Lucks. TWISTER$_\pi$ - a framework for secure and fast hash functions. *IJACT*, 2(1):68–81, 2010.

[50] Ewan Fleischmann, Christian Forler, and Stefan Lucks. Γ-MAC[H, P] - A New Universal MAC Scheme. In Frederik Armknecht and Stefan Lucks, editors, *WEWoRC*, volume 7242 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2011.

[51] Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A Foolproof On-Line Authenticated Encryption Scheme. *IACR Cryptology ePrint Archive*, 2011:644, 2011.

[52] Ewan Fleischmann, Christian Forler, and Stefan Lucks. The Collision Security of MDC-4. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *AFRICACRYPT*, volume 7374 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2012.

[53] Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In Anne Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 196–215. Springer, 2012.

[54] Ewan Fleischmann, Christian Forler, Stefan Lucks, and Jakob Wenzel. Weimar-DM: A Highly Secure Double-Length Compression Function. In Willy Susilo, Yi Mu, and Jennifer Seberry, editors, *ACISP*, volume 7372 of *Lecture Notes in Computer Science*, pages 152–165. Springer, 2012.

[55] Ewan Fleischmann, Michael Gorski, Jan-Hendrik Hühne, and Stefan Lucks. Key Recovery Attack on full GOST Blockcipher with Negligible Time and Memory. Proceedings of the Second Western European Workshop on Research in Cryptology, WEWoRC 2009, to appear in Lecture Notes in Computer Science.

[56] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. The Twister Hash Function Family, Available online at `http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/submissions_rnd1.html`. Submission to NIST (Round 1), 2008.

[57] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Attacking 9 and 10 Rounds of AES-256. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, volume 5594 of *Lecture Notes in Computer Science*, pages 60–72. Springer, 2009.

[58] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Memoryless Related-Key Boomerang Attack on 39-Round SHACAL-2. In Bao et al. [5], pages 310–323.

[59] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Memoryless Related-Key Boomerang Attack on the Full Tiger Block-cipher. In Bao et al. [5], pages 298–309.

[60] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. On the Security of Tandem-DM. In Dunkelman [38], pages 84–103.

[61] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Security of Cyclic Double Block Length Hash Functions. In Parker [134], pages 153–175.

[62] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Some Observations on Indifferentiability. In Ron Steinfeld and Philip Hawkes, editors, *ACISP*, volume 6168 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 2010.

[63] Pierre-Alain Fouque, Gwenaëlle Martinet, Frédéric Valette, and Sébastien Zimmer. On the Security of the CCM Encryption Mode and of a Slight Variant. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS*, volume 5037 of *Lecture Notes in Computer Science*, pages 411–428, 2008.

[64] Praveen Gauravaram, William Millan, and Lauren May. CRUSH: A New Cryptographic Hash Function using Iterated Halving Technique. In Ed Dawson and Wolfgang Klemm, editors, *Cryptographic Algorithms and their Uses*, pages 28–39. Queensland University of Technology, 2004.

[65] Henri Gilbert and Helena Handschuh, editors. *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*. Springer, 2005.

[66] Brian Gladman. Brian Gladman's AES Implementation, 19th June 2006. Available online at `http://gladman.plushost.co.uk/oldsite/AES/index.php`.

[67] Virgil D. Gligor and Pompiliu Donescu. Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes. In Matsui [116], pages 92–108.

[68] Shafi Goldwasser and Yael Tauman Kalai. On the (In)security of the Fiat-Shamir Paradigm. In *FOCS*, pages 102–113. IEEE Computer Society, 2003.

[69] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[70] H. Dobbertin. The status of MD5 after a recent attack, 1996.

[71] Bart Preneel Hans Dobbertin, Anton Bosselaers. RIPEMD (RACE integrity primitives evaluation message digest), 1996.

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

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

[74] Shoichi Hirose. Some Plausible Constructions of Double-Block-Length Hash Functions. In Robshaw [143], pages 210–225.

[75] Walter Hohl, Xuejia Lai, Thomas Meier, and Christian Waldvogel. Security of Iterated Hash Functions Based on Block Ciphers. In Stinson [163], pages 379–390.

[76] George Hotz. Console Hacking 2010 - PS3 Epic Fail. 27th Chaos Communications Congress, Available online at `http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf`, 2010.

[77] ISO/IEC. *ISO DIS 10118-2: Information technology - Security techniques - Hash-functions, Part 2: Hash-functions using an n-bit blockcipher algorithm. First released in 1992*, 2000.

[78] ISO/IEC. *19772:2009, Information technology – Security techniques – Authenticatd Encryption*, 2009.

[79] Tetsu Iwata. New Blockcipher Modes of Operation with Beyond the Birthday Bound Security. In Robshaw [143], pages 310–327.

[80] Tetsu Iwata. Authenticated Encryption Mode for Beyond the Birthday Bound Security. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2008.

[81] Tetsu Iwata and Kan Yasuda. BTM: A Single-Key, Inverse-Cipher-Free Mode for Deterministic Authenticated Encryption. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 313–330. Springer, 2009.

[82] Tetsu Iwata and Kan Yasuda. HBS: A Single-Key Mode of Operation for Deterministic Authenticated Encryption. In Dunkelman [38], pages 394–415.

[83] John P Steinberger. The Collision Intractability of MDC-2 in the Ideal Cipher Model. 2006. `http://eprint.iacr.org/`.

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

[85] Charanjit S. Jutla. Encryption Modes with Almost Free Message Integrity. *J. Cryptology*, 21(4):547–578, 2008.

[86] Jonathan Katz and Moti Yung. Unforgeable Encryption and Chosen Ciphertext Secure Modes of Operation. In *FSE*, pages 284–299, 2000.

[87] John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In Vaudenay [164], pages 183–200.

[88] John Kelsey and Bruce Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^{n} Work. In Cramer [33], pages 474–490.

[89] Aggelos Kiayias, editor. *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*. Springer, 2011.

[90] Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 252–267. Springer, 1996.

[91] Lars R. Knudsen. Hash Functions and SHA-3. Invited Talk at FSE 2008, available at `http://fse2008.epfl.ch/docs/slides/day_1_sess_2/Knudsen-FSE2008.pdf`, 2008.

[92] Lars R. Knudsen and Xuejia Lai. New Attacks on all Double Block Length Hash Functions of Hash Rate 1, including the Parallel-DM. In Alfredo De Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 410–418. Springer, 1994.

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

[94] Lars R. Knudsen, Florian Mendel, Christian Rechberger, and Søren S. Thomsen. Cryptanalysis of MDC-2. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2009.

[95] Lars R. Knudsen and Frédéric Muller. Some Attacks Against a Double Length Hash Proposal. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 462–473. Springer, 2005.

[96] Neal Koblitz and Alfred Menezes. Another Look at ”Provable Security”. II. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 148–175. Springer, 2006.

[97] Tadayoshi Kohno. Attacking and Repairing the WinZip Encryption Scheme. In *ACM Conference on Computer and Communications Security*, pages 72–81, 2004.

[98] Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A High-Performance Conventional Authenticated Encryption Mode. In Roy and Meier [155], pages 408–426.

[99] Matthias Krause, Frederik Armknecht, and Ewan Fleischmann. Preimage Resistance Beyond the Birthday Bound: Double-Length Hashing Revisited. Cryptology ePrint Archive, Report 2010/519, 2010. `http://eprint.iacr.org/`.

[100] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In Antoine Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.

[101] Kaoru Kurosawa, editor. *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*. Springer, 2007.

[102] Xuejia Lai and Kefei Chen, editors. *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*. Springer, 2006.

[103] Xuejia Lai and James L. Massey. Hash Function Based on Block Ciphers. In Rainer A. Rueppel, editor, *EUROCRYPT*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 1992.

[104] Jooyoung Lee and Daesung Kwon. The Security of Abreast-DM in the Ideal Cipher Model. Cryptology ePrint Archive, Report 2009/225, 2009. `http://eprint.iacr.org/`.

[105] Jooyoung Lee and Daesung Kwon. The Security of Abreast-DM in the Ideal Cipher Model. *IACR Cryptology ePrint Archive*, 2009:225, 2009.

[106] Jooyoung Lee and Martijn Stam. MJH: A Faster Alternative to MDC-2. In Kiayias [89], pages 213–236.

[107] Jooyoung Lee, Martijn Stam, and John P. Steinberger. The Collision Security of Tandem-DM in the Ideal Cipher Model. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 561–577. Springer, 2011.

[108] Anja Lehmann and Stefano Tessaro. A Modular Design for Hash Functions: Towards Making the Mix-Compress-Mix Approach Practical. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 364–381. Springer, 2009.

[109] Gaëtan Leurent, Charles Bouillaguet, and Pierre-Alain Fouque. SIMD Is a Message Digest. Submission to NIST (Round 2), 2009.

[110] Moses Liskov. Constructing an Ideal Hash Function from Weak Ideal Compression Functions. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 358–375. Springer, 2006.

[111] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable Block Ciphers. In Yung [169], pages 31–46.

[112] Stefan Lucks. Ciphers Secure against Related-Key Attacks. In Roy and Meier [155], pages 359–370.

[113] Stefan Lucks. Two-Pass Authenticated Encryption Faster Than Generic Composition. In Gilbert and Handschuh [65], pages 284–298.

[114] Stefan Lucks. A Collision-Resistant Rate-1 Double-Block-Length Hash Function. In Eli Biham, Helena Handschuh, Stefan Lucks, and Vincent Rijmen, editors, *Symmetric Cryptography*, volume 07021 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[115] M. Rabin. Digitalized Signatures: In R. DeMillo, D. Dobkin, A. Jones and R.Lipton, editors, Foundations of Secure Computation, Academic Press, pages 155-168, 1978.

[116] Mitsuru Matsui, editor. *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*. Springer, 2002.

[117] S. Matyas, C. Meyer, and J. Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985.

[118] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.

[119] David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[120] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

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

[122] C.H. Meyer and M. Schilling. Secure program load with manipulation detection code. In *SECURICOM'88*, pages 111–130, France, 1988. Paris.

[123] S. Miyaguchi, K. Ohta, and M. Iwata. 128-bit hash function (N-Hash). In *SECURICOM '90*, pages 123–137, 1990.

[124] Mridul Nandi, Wonil Lee, Kouichi Sakurai, and Sangjin Lee. Security Analysis of a 2/3-Rate Double Length Compression Function in the Black-Box Model. In Gilbert and Handschuh [65], pages 243–254.

[125] National Bureau of Standards. *FIPS Publication 46-1: Data Encryption Standard*, January 1988.

[126] Niels Ferguson and Stefan Lucks and Bruce Schneier and Doug Whiting and Mihir Bellare and Tadayoshi Kohno and Jon Callas and Jesse Walker. Skein source code and test vectors. `http://www.skein-hash.info/downloads`.

[127] Jesper Buus Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In Yung [169], pages 111–126.

[128] NIST National Institute of Standards and Technology. SHA-3 Cryptographic Hash Algorithm Competition (2007-2012). More information available at `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`.

[129] NIST National Institute of Standards and Technology. FIPS 180-1: Secure Hash Standard. April 1995. See `http://csrc.nist.gov`.

[130] NIST National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard. August 2002. See `http://csrc.nist.gov`.

[131] NIST National Institute of Standards and Technology. NIST Special Publication 800-57 Recommendation for Key Management - Part 1: General (Revised), March, 2007. Abailalbe at `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf`.

[132] NIST National Institute of Standards and Technology. Tentative Timeline of the Development of New Hash Functions. `http://csrc.nist.gov/pki/HashWorkshop/timeline.html`.

[133] Onur Özen and Martijn Stam. Another Glance at Double-Length Hashing. In Parker [134], pages 176–201.

[134] Matthew G. Parker, editor. *Cryptography and Coding, 12th IMA International Conference, Cryptography and Coding 2009, Cirencester, UK, December 15-17, 2009. Proceedings*, volume 5921 of *Lecture Notes in Computer Science*. Springer, 2009.

[135] Kenneth G. Paterson and Gaven J. Watson. Plaintext-Dependent Decryption: A Formal Security Treatment of SSH-CTR. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 345–361. Springer, 2010.

[136] B. Preneel. *Analysis and Design of Cryptographic Hash Functions.* Thesis (Ph.D.), Katholieke Universiteit Leuven, Leuven, Belgium, January 1993.

[137] Bart Preneel, Antoon Bosselaers, Rene Govaerts, and Joos Vandewalle. Collision-free hashfunctions based on blockcipher algorithms. In *Proceedings 1989 International Carnahan Conference on Security Technology (Oct 3–5 1989: Zurich, Switzerland)*, pages 203–210, pub-IEEE:adr, 1989. IEEE Computer Society Press. IEEE catalog number 89CH2774-8.

[138] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In Stinson [163], pages 368–378.

[139] Vincent Rijmen and Paulo S. L. M. Barreto. The WHIRLPOOL Hash Function. World-Wide Web document, 2001.

[140] Thomas Ristenpart and Thomas Shrimpton. How to Build a Hash Function from Any Collision-Resistant Function. In Kurosawa [101], pages 147–163.

[141] R. L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm.* Internet Activities Board, April 1992.

[142] Ronald L. Rivest. The MD4 Message Digest Algorithm. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 303–311. Springer, 1990.

[143] Matthew J. B. Robshaw, editor. *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March*

15-17, 2006, Revised Selected Papers, volume 4047 of *Lecture Notes in Computer Science*. Springer, 2006.

[144] Phillip Rogaway. Authenticated-Encryption with Associated-Data. In *ACM Conference on Computer and Communications Security*, pages 98–107, 2002.

[145] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[146] Phillip Rogaway. Nonce-Based Symmetric Encryption. In Roy and Meier [155], pages 348–359.

[147] Phillip Rogaway. Formalizing Human Ignorance. In Phong Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.

[148] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205, 2001.

[149] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Roy and Meier [155], pages 371–388.

[150] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In Vaudenay [164], pages 373–390.

[151] Phillip Rogaway and Thomas Shrimpton. Deterministic Authenticated-Encryption: A Provable-Security Treatment of the Key-Wrap Problem. Cryptology ePrint Archive, Report 2006/221; full version of [150], 2006. `http://eprint.iacr.org/`.

[152] Phillip Rogaway and John P. Steinberger. Constructing Cryptographic Hash Functions from Fixed-Key Blockciphers. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 433–450. Springer, 2008.

[153] Phillip Rogaway and John P. Steinberger. Security/Efficiency Tradeoffs for Permutation-Based Hashing. In Smart [159], pages 220–236.

[154] Phillip Rogaway and Haibin Zhang. Online Ciphers from Tweakable Blockciphers. In Kiayias [89], pages 237–249.

[155] Bimal K. Roy and Willi Meier, editors. *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*. Springer, 2004.

[156] Todd Sabin. Vulnerability in Windows NT's SYSKEY encryption. *BindView Security Advisory*, 1999. Availalbe at `http://marc.info/?l=ntbugtraq&m=94537191024690&w=4`.

[157] Satoh, Haga, and Kurosawa. Towards Secure and Fast Hash Functions. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, 1999.

[158] Victor Shoup, editor. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.

[159] Nigel P. Smart, editor. *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*. Springer, 2008.

[160] Martijn Stam. Blockcipher-Based Hashing Revisited. In Dunkelman [38], pages 67–83.

[161] John P Steinberger. The Collision Intractability of MDC-2 in the Ideal Cipher Model. Cryptology ePrint Archive, Report 2006/294, 2006. `http://eprint.iacr.org/`.

[162] John P. Steinberger. The Collision Intractability of MDC-2 in the Ideal-Cipher Model. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 34–51. Springer, 2007.

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

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

[165] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer [33], pages 1–18.

[166] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Shoup [158], pages 17–36.

[167] Robert S. Winternitz. A Secure One-Way Hash Function Built from DES. In *IEEE Symposium on Security and Privacy*, pages 88–90, 1984.

[168] Hongjun Wu. The Misuse of RC4 in Microsoft Word and Excel. Cryptology ePrint Archive, Report 2005/007, 2005. `http://eprint.iacr.org/`.

[169] Moti Yung, editor. *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*. Springer, 2002.

Index