Design and Analysis of Multi-Block-Length Hash Functions

THÈSE N^{O} 5333 (2012)

PRÉSENTÉE LE 29 JUIN 2012

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE DE CRYPTOLOGIE ALGORITHMIQUE

PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Onur ÖZEN

acceptée sur proposition du jury:

Prof. E. Telatar, président du jury
Prof. A. Lenstra, directeur de thèse
Prof. W. Meier, rapporteur
Dr M. Stam, rapporteur
Prof. S. Vaudenay, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2012

To my beloved grandmother

SATÍYE ÖZEN

Acknowledgements

First and foremost I would like to express my deep and sincere gratitude to my advisor Arjen K. Lenstra for giving me an invaluable opportunity to pursue a Ph.D. at LACAL. It has always been an honor for me to be a part of his group ; I also would like to thank him for his limitless moral support and close friendship that led to unforgettable moments during my Ph.D work at EPFL. Finally, and more importantly, I am thankful to him for giving me vision to understand life better and of course for sharing with me his collection of classical music masterpieces everyday.

Special thanks go to Martijn Stam who basically served as my mentor during the last four years ; without his amazing ideas and intelligence it would have been impossible for me to tackle the problems I faced with. He is my co-author for all the scientific work presented in this thesis : I am particularly thankful to him for his patience and joyful friendship ; and for trying his best to keep our social lives more enjoyable.

I give many thanks to Dimitar Jetchev who, after joining LACAL, spent most of his time with me to help solve the problems I had been working on. His friendship and support always gave me more confidence and enabled me to relax. I am also grateful to Tom Shrimpton for proposing the project I was involved in and for serving as the principal investigator for quite some time.

I would like to thank Joppe Bos with whom I shared an office and very enjoyable moments. In addition to our close friendship, we wrote two articles together that, although not included in this thesis, were published in two good international conferences. I also would like to thank Shahram Khazaei, my dear friend and ex-office mate, who helped me a lot to get used to Switzerland and EPFL.

It was a great pleasure to have spent valuable moments with previous and current LACAL folks : Patrick Amon, Maxime Augier, Alina Dudeanu, Nicolas Gama, Marcelo E. Kaihara, Alexandre Karlov, Thorsten Kleinjung, Andrea Miele, Seyyd H. Mir Jalili, Dag Arne Osvik and Juraj Šarinay. I am very thankful to Monique Amhof who runs all the important practical administrative duties in our laboratory and, of course, to Holly B. Cogliati for her invaluable support on writing this dissertation.

Thanks to the very smart people I worked with, we produced several papers that were published in different international conferences ; for certain reasons, however, these papers are not included in this thesis. Nevertheless, I would like to express my gratitude to my co-authors who collaborated with me on these projects : Jean-Philippe Aumasson, Çağdaş Çalık, Jean-Pierre Hubaux, Çelebi Kocair, Willi Meier, Raphael C.-W. Phan, Cihangir Tezcan and Kerem Varıcı.

Many thanks to my ex-flatmate Roland Mages and all of my friends at EPFL, in particular to the Turkish community who always supported me in my very stressful moments. I am also thankful to the financial support from the Swiss National Science Foundation, 200021-122162.

Last but not least, I would like to express my heart-felt thanks to my wife, Nihal, for sharing this journey with me and supporting me with her limitless love. I am thankful to my parents for giving me vision, support and confidence and to every single member of my family for believing in me during this period.

Abstract

Cryptographic hash functions are used in many cryptographic applications, and the design of provably secure hash functions (relative to various security notions) is an active area of research. Most of the currently existing hash functions use the Merkle–Damgård paradigm, where by appropriate iteration the hash function inherits its collision and preimage resistance from the underlying compression function. Compression functions can either be constructed from scratch or be built using well-known cryptographic primitives such as a blockcipher. One classic type of primitive-based compression functions is *single-block-length* : It contains designs that have an output size matching the output length n of the underlying primitive. The single-block-length setting is well-understood. Yet even for the optimally secure constructions, the (time) complexity of collision- and preimage-finding attacks is at most $2^{n/2}$, respectively 2^n ; when $n = 128$ (e.g., Advanced Encryption Standard) the resulting bounds have been deemed unacceptable for current practice. As a remedy, *multi-block-length* primitive-based compression functions, which output more than n bits, have been proposed. This output expansion is typically achieved by calling the primitive multiple times and then combining the resulting primitive outputs in some clever way. In this thesis, we study the collision and preimage resistance of certain types of multi-call multi-block-length primitive-based compression (and the corresponding Merkle–Damgård iterated hash) functions : Our contribution is three-fold.

First, we provide a novel framework for blockcipher-based compression functions that compress $3n$ bits to $2n$ bits and that use two calls to a $2n$-bit key blockcipher with block-length n. We restrict ourselves to two parallel calls and analyze the sufficient conditions to obtain close-to-optimal collision resistance, either in the compression function or in the Merkle–Damgård iteration.

Second, we present a new compression function $\mathsf{h}\colon \{0,1\}^{3n} \to \{0,1\}^{2n}$; it uses two parallel calls to an ideal primitive (public random function) from $2n$ to n bits. This is similar to MDC-2 or the recently proposed MJH by Lee and Stam (CT-RSA'11). However, unlike these constructions, already in the compression function we achieve that an adversary limited (asymptotically in n) to $\mathcal{O}(2^{2n(1-\delta)/3})$ queries (for any $\delta > 0$) has a disappearing advantage to find collisions. This is the first construction of this type offering collision resistance beyond $2^{n/2}$ queries.

Our final contribution is the (re)analysis of the preimage and collision resistance of the Knudsen–Preneel compression functions in the setting of public random functions. Knudsen–Preneel compression functions utilize an $[r,k,d]$ linear error-correcting code over $\mathbb{F}_{2^e}$ (for $e > 1$) to build a compression function from underlying blockciphers operating in the Davies–Meyer mode. Knudsen and Preneel show, in the complexity-theoretic setting, that finding collisions takes time at least $2^{(d-1)n/2}$. Preimage resistance, however, is conjectured to be the square of the collision resistance. Our results show that both the collision resistance proof and the preimage resistance conjecture of Knudsen and Preneel are incorrect : With the exception of two of the proposed parameters, the Knudsen–Preneel compression functions do not achieve the security level they were designed for.

Keywords : Hash function, compression function, blockcipher, security proof, cryptanalysis.

Résumé

Les fonctions de hachage cryptographiques sont utilisées dans de nombreuses applications cryptographiques, et la conception de fonctions de hachage dont la sécurité est prouvable (relativement à diverses notions de sécurité) est un domaine actif de la recherche. La plupart des fonctions de hachage existantes utilisent le paradigme de Merkle–Damgård, où la fonction de hachage hérite, par une itération appropriée, des propriétés de résistance à la collision et à la préimage d'une fonction de compression sous-jacente. Ces fonctions de compression sont soit construites entièrement pour l'occasion, soit dérivées d'une autre primitive cryptographique bien maîtrisée, comme un chiffrement par blocs. Un type classique de fonctions de compression basées sur une telle primitive est une fonction dont la taille de sortie correspond à la taille de sortie de la primitive. La sécurité de ce type de fonctions est bien comprise ; pourtant, même dans le cas d'une construction à la sécurité optimale, la complexité en temps des attaques de collision et de préimage sont, au plus, respectivement de $2^{n/2}$ et de 2^n. Lorsque n vaut 128 (ce qui est le cas pour AES, l'Advanced Encryption Standard) les bornes correspondantes ont été jugées inacceptables pour un usage pratique actuel. Pour y remédier, des fonctions de compression basées sur une primitive, mais dont la taille de sortie est un multiple de la taille de sortie de la primitive, ont été proposées. L'expansion de la sortie est typiquement obtenue en appelant la primitive plusieurs fois, et en combinant de manière intelligente les sorties respectives. Dans cette thèse, nous étudions la résistance à la collision et à la préimage de certains types de fonctions de compression faisant plusieurs appels à la primitive sous-jacente et dont la taille de sortie est un multiple de la taille de sortie de la primitive. Nous étudions aussi les fonctions de hachage obtenues lorsque la fonction de compression est utilisée dans le paradigme de Merkle–Damgård. Notre contribution est triple.

Premièrement, nous fournissons un nouveau cadre pour des fonctions de compression basées sur un chiffrement par blocs, qui compressent $3n$ bits vers $2n$ bits en utilisant deux appels à un chiffrement par blocs ayant une longueur de clef de $2n$ bits et une longueur de bloc de n bits. Nous nous restreignons à deux appels parallèles, et analysons les conditions suffisantes pour obtenir une résistance à la collision proche de l'optimum, soit directement dans la fonction de compression, soit lorsque la fonction est utilisée dans le procédé de Merkle–Damgård.

Deuxièmement, nous présentons une nouvelle fonction de compression $\mathrm{h}\colon \{0,1\}^{3n} \to \{0,1\}^{2n}$, utilisant deux appels parallèles à une primitive idéale (une fonction aléatoire publique) de $2n$ vers n bits. Cette construction est similaire à MDC-2 ou à la récente proposition MJH de Lee et Stam (CT-RSA'11). Cependant, contrairement à ces constructions, nous obtenons qu'un adversaire limité à $\mathcal{O}(2^{2n(1-\delta)/3})$ requêtes (asymptotiquement en n, pour tout $\delta > 0$) a un avantage négligeable de trouver des collisions (pour la fonction de compression). La construction que nous proposons est la première de ce type offrant une résistance à la collision au delà de $2^{n/2}$ requêtes.

Notre contribution finale est la (ré)analyse de la résistance à la préimage et à la collision des fonctions de compression de Knudsen-Preneel dans le contexte de fonctions aléatoires publiques. Les fonctions

de compression de Knudsen-Preneel utilisent un code correcteur d'erreurs linéaire de paramètres $[r,k,d]$ sur $\mathbb{F}_{2^e}$ (pour $e>1$) pour construire une fonction de compression sur un chiffrement par blocs sous-jacent fonctionnant suivant le mode de Davies–Meyer. Knudsen et Preneel montrent, dans un contexte de complexité théorique, que trouver des collisions demande un temps d'au moins $2^{(d-1)n/2}$. La résistance à la préimage, cependant, est simplement conjecturée être le carré de la résistance à la collision. Nos résultats montrent que la preuve de résistance à la collision, tout comme la conjecture de résistance à la préimage, de Knudsen et Preneel, sont incorrectes : Á la possible exception de deux ensembles de paramètres sur tous ceux proposés, les fonctions de compression de Knudsen-Preneel n'atteignent pas le niveau de sécurité pour lequel elles ont été conçues.
Mots-clés : Fonctions de hachage, Fonctions de compression, chiffrement par blocs, preuve de la sécurité, cryptanalyse.

Contents

List of Figures

List of Tables

1 Introduction

The title of this thesis contains the term 'hash function' [1]. It is regarded as one of the key components of modern cryptology; many also refer to it as 'the salt and pepper' of cryptology used today. In this chapter, by casting a light on how cryptology itself developed as a science which has its roots from an 'ancient art', we will learn how cryptographic hashing became so vital, hence why a thesis should be written on it. Beginning with the very early methods and the basic notions, we present (very informally) the major developments of cryptology and describe how cryptographic hash functions have taken an important role in this field. The readers familiar with these basics might well skip this chapter and start reading the thesis from Chapter 2 onwards.

1.1 Cryptology

1.1.1 A Brief Historical Tour

Finding (efficient) ways of sending messages where only the intended recipients (and the sender itself) are able to read the original message is dealt with by one of the two major branches of cryptology, called *cryptography*. Cryptography has been known and used for a long time, initially used mainly for military purposes to prevent the enemies [2] from reading secret information. The tool—developed by the methods of cryptography—allowing such a property is called a *cipher*, a term that formerly meant 'zero' in the Latin world and that has its roots in Arabic.

One of the oldest ciphers known today is due to the Romans, in particular to Julius Caesar (*Caesar cipher*) who *encrypted* messages (*plaintext*) to communicate with his generals: Each letter in the message to be encrypted is replaced by a letter three positions down the alphabet. For instance, 'a' would be replaced by 'd', 'b' would become 'e', etc. In order to *decrypt* the encrypted message (*ciphertext*), it is sufficient to know the above secret knowledge, or the so-called *secret-key*, which in this case is this simple substitution over the alphabet.

According to historical notes, there existed no known method at that time for obtaining the original message without knowing the secret-key, until the discovery of *frequency analysis* in the Arab world

1. By hash functions we mean cryptographic hash functions; throughout, these two terms will be used interchangeably to refer to the same concept.

2. The word enemy stems from the military-based applications; in the cryptographic literature, the term used to refer to an opponent is *adversary* or *eavesdropper*.

(ninth century [85]). *Frequency analysis* is a method allowing an opponent to decrypt ciphertexts and obtain the secret-key without initially knowing it. This effort is generally referred to as *breaking* or *attacking* the cipher; yet today it has a broader meaning. The branch of cryptology dealing with developing methods for 'assessing the security' of the cryptographic primitives is called *cryptanalysis*, which, along with cryptography, constitutes the science of cryptology.

After the fall of the Caesar cipher, triggered by the advances in cryptanalysis, more advanced methods were employed, such as the *Vigenère cipher* [85] or *le chiffre indéchiffrable*. Here the shared secret knowledge is gathered through the Vigenère square, also known as the *tabula recta*, that can be used for both encrypting the plaintexts (*encryption*) and decrypting the ciphertexts (*decryption*). The Vigenère cipher was successfully cryptanalyzed after a few centuries and various other schemes were proposed by cryptographers.

In the course of time, cryptography and cryptanalysis turned out to be two fields not only affecting each other significantly but also having direct consequences in the real world. For instance, as exploited during the Second World War by the Allies, obtaining the *Enigma* machine and its successful cryptanalysis allowed them to decrypt the confidential and sensitive information of the German military. Consequently, the Allied forces gained a significant strategic advantage, which undoubtedly changed the course of the war. Indeed, as emphasized by Winston Churchill to Britain's King George VI: "It was thanks to Ultra[3] that we won the war" [48].

The use of the secret component of Enigma is no different than that of the Caesar or Vigenère cipher, except for its (much more) complicated structure. That is, the secret knowledge now is the device itself, along with the initial states that are set among the intended recipients before the communication starts. Note that in all these three examples, the security of the cryptographic system relies on the very same secret knowledge shared among the intended users, whereas an adversary is assumed not to have access to it. This type of cryptographic system, still very widely accepted, belongs to the field called *symmetric-key cryptography*, symmetric in the sense that the (secretly) communicating parties do have the same key. This had been the only way of secret communication until very recently; the role of cryptology in modern world has been boosted by the invention of *public-key cryptography* [54].

However, even well before (in the late 19th century) public-key cryptography was discovered, there was a remarkable effort to reformulate the term 'security' in symmetric-key cryptography by using axiomatic terms and looking at the problem from a modern point of view. This term, also known as Kerckhoffs' Principle [85], is one of the striking ideas from earlier years of cryptology shedding some light on the evolution of this field. Generally and informally speaking, Kerckhoffs' Principle states that a cryptographic system should be secure even if everything, except for the secret-key, about the encryption and decryption process is publicly known. This concept has been widely accepted by cryptographers, compared to what is known as *security through obscurity*.

Still for most cryptographers, modern cryptology was born by the pioneering work [182] of Shannon, who introduced the mathematical structure of secrecy systems using his then-new theory of communications and information [181]. The major conclusion of Shannon is that, given unlimited computing power, the amount of information that can be sent with perfect secrecy is at most the information available in the secret-key. Hence, a malicious adversary with unlimited computational

3. Ultra is the codename for the information gained from the cryptanalysis of Enigma.

power can (in theory) obtain part of the secret message if more information is transmitted. Obviously, Shannon's information-theoretic cryptography does not reflect very well the cryptography used in practice, as adversaries in real life are computationally bounded (which allows for more messages to be sent). Still, it provides a good measure as a first step towards understanding the structural properties of a cryptosystem. Even more importantly, Shannon's work serves as one of the first scientific references to modern cryptology.

For more on the history and evolution of cryptology, we refer to two well-known and remarkable books: *The Codebreakers* by Kahn [85] and, more recent, *The Code Book* by Singh [188].

1.1.2 Modern Cryptology

Until the 1960s, cryptology had been used only by the military and secret services in order to keep confidential information secret. Beginning in the 1970s, due to the fast development of computers and communication in general, the need for cryptology became evident for the private sector as well. This need, along with the involvement of the research community, transformed its role from an ancient art to a branch of science.

Cryptology as a science, developed in two major directions: the traditional symmetric-key model and, relatively new, public-key (or the asymmetric-key) setting. The former is just the modern view of the good old symmetric-key cryptology also used by Caesar, which nowadays contains much more intelligent and involved techniques. Whereas public-key cryptology is based on principles that are entirely different from those underlying the symmetric-key model. Today, both techniques are at the core of modern cryptology.

Advances in Symmetric-Key Cryptology

One of the important developments in cryptology is that the US standards body, then named NBS (National Bureau of Standards), deemed necessary a symmetric-key encryption standard—the US FIPS (Federal Information Processing Standard)—for encrypting unclassified yet sensitive information. To this end, in 1973, they made a call for proposals for a cipher that would meet their requirements. Interestingly, after the initial investigation, none of the proposed designs were found acceptable based on their criteria. Then, in 1974, they made a second call that turned out to be more fruitful: A submission from IBM (developed by Feistel in the early 1970s), which was heavily influenced by an earlier cipher called *Lucifer* [60], was found suitable. The standard was filed in 1976 and the cipher was called DES (Data Encryption Standard) [138]. The IBM effort led by Feistel and the deployment of DES accelerated the role of cryptography in the civilian world, in particular in the research arena. Indeed, the community then had an explicitly defined cipher to work on, which had never been the case before.

DES is currently referred to as a *blockcipher*[4]; it informally means (see Section 2.3 for a more formal treatment) that the plaintexts are partitioned into blocks of strings of fixed-length and each block is encrypted using a secret-key where each key specifies a fixed permutation (over the domain of input blocks). A single execution of DES encrypts a 64-bit plaintext using a 56-bit key. In order to encrypt

4. Blockciphers are not the only symmetric-key cryptographic systems; streamciphers can also be given in this category. We do not consider the latter in this thesis. One can refer to [2] for the developments in the field of streamciphers.

arbitrary-length plaintexts, it is necessary to use a mode of operation (along with a proper padding scheme and multiple DES executions) where we refer to one of the classic books on cryptology (e.g., [122]) for available blockcipher modes of operation.

Parallel to these advances in cryptography, cryptanalysts made significant efforts after the design of DES to analyze its security. *Differential cryptanalysis* by Biham and Shamir [22, 23] and Matsui's *linear cryptanalysis* [114, 115] can be given as the two most powerful techniques in this direction. These two major tools are still at the heart of more involved cryptanalysis techniques that are known today and are used against the symmetric-key cryptographic primitives. Nevertheless, DES and its triple-key variant TDEA [134] had been used extensively until the late 1990s (TDEA is still being used) as the developed cryptanalytic techniques were mostly theoretical, i.e., they did not cause "a practical threat" against DES at the time due to their computational requirements.

A more practical threat against DES came with the developments in technology and the use of the Internet for cryptanalytic purposes. Specifically, towards the end of the 1990s, using reasonable financial resources and time, DES was shown to be susceptible to *exhaustive key search* [1], which is the most basic and straightforward method to obtain the secret-key; it simply functions by trying every possible key until the correct one is found. The attack was described as follows [1]: "Tens of thousands of computers, all across the U.S. and Canada, linked together via the Internet in an unprecedented cooperative supercomputing effort to decrypt a message encoded with the government-endorsed Data Encryption Standard (DES)". The effort turned out to be productive and successfully terminated in four months showing that the short key-length of DES is not suitable anymore for sensitive applications.

Triggered by these advances in cryptanalysis, as an alternative to DES, cryptographers proposed many blockciphers in the 1990s. One of the common characteristics of these designs is that they were proposed to address yet another issue, namely the efficiency of DES: At the time DES was designed, the major applications were mainly supported by special purpose hardware and consequently DES was primarily targeted taking into account the efficiency in hardware. Although later shown to be sufficiently fast in software [148] on many platforms, DES was not very flexible for many modern architectures. These two problems, i.e., short key-length and the efficiency issues, led to the call for the second symmetric-key cryptographic standard by the US NIST (National Institute of Standards and Technology). In early 1997, NIST made its announcement for AES (Advanced Encryption Standard) to select "an unclassified, publicly disclosed encryption algorithm capable of protecting sensitive government information well into the next century" [137].

There were several differences between the DES and AES selection processes; the most evident of which is that this time NIST made a call for blockcipher submissions supporting 128-bit plaintext blocks, as well as three key-length flavors: 128-, 192- and 256-bit. Moreover, the calls seemed to be made public without exclusive involvement from non-civilian organizations. As a result, fifteen submissions were received from all over the world—in 1999 five of them were announced as finalists—and the blockcipher *Rijndael* [51] was selected as AES in 2001 [139].

Rijndael was designed by two Belgian cryptographers, Daemen and Rijmen, and it was clearly the most software-efficient (and reasonably fast and compact in hardware) candidate among the five finalists, considering many platforms. Even regarding security, the arguments provided by the designers (and the extensive efforts during the selection process) were also found convincing at

the time. But, the real story is a bit different, at least from a security point of view, which has been well-known by most of the people working in this area: Cryptanalytic methods never get worse, on the contrary, they improve quite rapidly. Therefore, although convincing at the time, everyone was aware of the unavoidable fact: AES may not be able to provide resistance against 'unknown' and yet to be improved cryptanalytic methods.

Unsurprisingly, in the following years, the community did not forget this fact. AES was first shown to be susceptible to so-called *side-channel attacks* [16, 199], which benefit from the information leaked by the physical implementation of a cryptographic algorithm. Fortunately, the methods [16, 199] themselves suggest countermeasures to prevent such weaknesses. Yet, it was the first indication that the current standard might have some security flaws. The second indication came recently with the result of Biryukov and Khovratovich [24]: a cryptanalysis of AES variants of 192- and 256-bit secret-keys using a sophisticated method derived from *differential cryptanalysis*. Their result is (as of today) far from being practical [5] due to the high complexity of their technique; as a result, AES is still considered to be reasonably secure, at least for practitioners. Yet these advances show a clear indication that AES has structural weaknesses as well.

After experiencing these two standardization processes and the subsequent cryptanalytic developments, it is questionable that AES is the blockcipher the century has been waiting for. Hence, a new call for proposals from NIST for the next generation blockcipher is to be expected, not very soon perhaps but at least in a couple of decades.

A New Era: The Rise of the Public-key Cryptology

Although symmetric-key cryptography provides, to some extent, a solution to ensuring secrecy, it has a clear disadvantage that has been known for quite some time: the key distribution. Indeed, in order to establish a secure communication via symmetric-key cryptography, we need to ensure, that the intended recipients do have the required shared secret knowledge, well before the communication starts. Now, imagine that there exist thousands or millions of users trying to communicate with each other using symmetric-key cryptography. If everyone uses the very same key, then it is easy to keep track of the secret-keys. Yet, once the shared key is revealed, then the whole system gets broken. If, on the contrary, there exist several keys among the users, then it becomes a very difficult and non-trivial task to manage and distribute the keys. Hence, it would be desirable to distribute the keys or at least agree on them in a more efficient manner.

Public-key cryptography addresses these issues and provides remediations using unconventional yet practical approaches. The method was first described in the seminal work of Diffie and Hellman [54] who suggested the famous *Diffie–Hellman key exchange* inspired by Merkle's work on public-key distribution [124]. Diffie and Hellman's ingenious idea is that each user has a different (and independently generated) pair of private- and public-keys where the private-key is kept secret by each user, whereas the (somehow related) public-key is known by everybody (which ideally cannot be used to derive the private-key). The sender simply uses the public-key of the recipient to encrypt a message

5. We note that the aforementioned analysis done by Biryukov and Khovratovich [24] works under the related-key model; a model where there exist multiple keys and the attacker is assumed to have some extra knowledge about the keys. This model, for many of the researchers, is considered to be unpractical. Nevertheless, it is still possible to attack AES using single keys (again with unpractical computational requirements), see the recent result of Bogdanov et al. [31] on all versions of AES.

and (only) the intended recipient can read the message by using its private-key. Clearly, the users do not have to share any secret information. To start the communication, we only need to search for the public-key of the corresponding recipient (assuming that the public-key belongs to the intended user).

The key ingredient of the *Diffie–Hellman key exchange* is the so-called *one-way function*; these are functions that are easy to compute (at least in practice) for any given input but hard to invert given the image of a random input. For public-key encryption, however, we would need a *trapdoor one-way function*, a function still hard to invert, except for those with additional information. The concept of trapdoor one-way functions was introduced by Diffie and Hellman [54] in their famous work; yet they were not able to illustrate an example of it nor a public-key encryption scheme. In 1978, Rivest, Shamir and Adleman [167] published the first public-key cryptosystem known as RSA. Their major contribution is the explicit example of a trapdoor one-way function that relies on the intractability of computing roots modulo a composite number constructed as the product of two large prime numbers. In this case, the factorization of the modulus allows for easy extraction of the roots and thus provides the trapdoor. The security of the RSA cryptosystem system is based on the 'presumed' difficulty of factoring large integers.

One of the nicest applications of public-key cryptography is a new application called *digital signatures*, a concept providing authentication like traditional handwritten signatures; the major difference being that ideally constructed digital signatures are even more difficult to forge than the handwritten type. Digital signatures were introduced by Diffie and Hellman by noting that only the private-key owner can invert the trapdoor one-way function, and therefore, prove that the message indeed originates from him. As in the case of public-key encryption, finding a concrete digital signature scheme was an open problem until the invention of RSA.

Following the first encryption and digital signature scheme RSA, there have been many public-key cryptosystems proposed starting from the late 1970s, and the difficulty of developing such techniques was realized soon after. Specifically, proving that a one-way function is efficient to compute is easy; yet showing that it is hard to invert is, in general, difficult. That is why most of the public-key cryptosystems known even today are based on the difficulty assumptions of well-known mathematical problems, e.g., integer factorization. If a cryptanalytic breakthrough is achieved that can affect the underlying hardness assumption, then the system would be considered broken, or we would need to modify the scheme to increase the security limit. Unfortunately, this, in general, results in a much less efficient construction. Indeed, the revolutionary advances in the integer factorization methods [109] led RSA to increase the key-length for practical applications, which resulted in a less efficient scheme operating on thousands of bits instead of a few hundred.

In order to remedy this issue, alternative techniques have been proposed; the most promising of which is based on *elliptic curves*, known also as *elliptic curve cryptography* (ECC) [98, 126]. This novel technique relies on a different hardness assumption: the so-called *discrete-logarithm problem* in cyclic subgroups of elliptic curves. The main advantage of ECC over RSA is that much smaller keys are required for the same level of security. As a result, we can significantly decrease the storage requirements (e.g., the public- and private-key pair) by using an ECC-based cryptosystem and enjoy more practical implementations. From security point of view, however, not much progress has been achieved, unlike integer factorization (except for the parallelized version [147] of Pollard's work [158]).

Although reasonably efficient, all of the public-key cryptosystems known to date are slower than the conventional symmetric-key primitives, e.g., AES. Therefore, in practice, to encrypt large amounts of data symmetric-key ciphers are often preferred over public-key algorithms. Yet, key agreement and authentication are still handled via public-key methods. For this reason, like in symmetric-key cryptography, NIST conducts standardization efforts for public-key cryptosystems as well. For instance, in 1994, the first US FIPS was filed for a *digital signature standard* —called DSA (see the latest version in [142])—which is a variant of ElGamal Signature Scheme [59]. Today, DSA supports an elliptic curve extension for DSA named ECDSA [142].

Finally it is worth noting that one of the most beautiful results of public-key cryptography is that it opens up many possibilities and new ideas for cryptology. Given its flexibility and power, it seems that public-key cryptology will remain at the core of cryptologic research, as well as real-life security applications, for quite some time.

1.1.3 Cryptography Today

The fundamental goal of cryptography that we understand today is to address (at least) the following four notions [122] in a theoretically sound and practically efficient way:

1. *Confidentiality:* It provides solutions for the protection and the secrecy of the data by allowing access only to authorized recipients.

2. *Authenticity:* This notion is a service related to identification, which allows us to identify entities trying to communicate with each other.

3. *Data integrity:* It deals with the detection of the manipulated data done by unauthorized parties.

4. *Non-repudiation:* It is a service for the confirmation and denial of the previous actions done by the entities.

All these notions appear quite often in daily life where cryptography is present, without even being noticed. For instance, banking applications via ATM machines or those using the Internet for e-banking services have become very common. Our identity cards, passports, mobile and smart phones, they all serve as cryptographic modules in order to facilitate our basic needs in a secure way. In the near future, many countries will start moving towards e-voting and almost all bureaucratic applications will be handled via e-government tools.

The research community today is trying to find effective cryptographic solutions for almost all of these applications, each requiring different constraints to be satisfied. The main goal here is to achieve a satisfactory trade-off between security and efficiency for cryptosystems that are used in many different platforms, ranging from tiny smart cards to high-end workstations. The high-level contribution of this thesis should be seen in this direction as well: In addition to achieving several provable arguments from a security point of view, we consider relevant, practical constructions that might well be deployed in real-life applications (see Section 2.4 for a more detailed, technical description of the contribution of this work).

1.2 The Role of Cryptographic Hash Functions

This subsection provides an informal introduction to cryptographic hash functions—a concept initially introduced by Diffie and Hellman [54] so as to make digital signatures more efficient—and illustrates their roles in today's cryptography in an intuitive way. Moreover, we will look at the most famous constructions to date and briefly recapitulate the basic advances in this field. A more detailed discussion is in Chapter 2.

To be able to discuss cryptographic hash functions, it is worth introducing the term 'algorithm'. Probably the best way is to consult Knuth [95] for the definition. An *algorithm* can be seen as a process, a method or a recipe with a finite set of rules containing a sequence of operations in order to solve a well-defined problem. Yet, an algorithm should also satisfy several other properties. Firstly, it should have some *inputs* and *outputs* where the outputs should be determined (given inputs) after a finite number of steps (*finiteness*). Moreover, it should be *definite* and *effective* in the sense that in each step it should be perfectly clear what step to perform and how.

A *hash function* can be defined as a deterministic and efficient algorithm mapping an input of arbitrary length into an output of fixed length (input length being generally much bigger than the output length). The output can be called the *hash value*, *fingerprint* or simply the *digest* of the corresponding message. Not all hash functions are cryptographic; indeed, *cryptographic hash functions* are expected to satisfy certain 'cryptographic' properties that vary depending on the application. In the very informal sense, these properties are (second) preimage resistance and collision resistance. The former is a notion related to the one-wayness; for cryptographic hash functions it should be hard to find an input for any given specific output (without knowing any corresponding inputs). Whereas collision resistance requires the difficulty of finding two distinct messages with the same digest. A more formal treatment for these security notions is carried out in Section 2.1.2.

1.2.1 Applications

Historically, in their first applications (as a one-way function), hash functions were used for masking or compressing passwords [143]; because we would not want to take the risk of exposing all passwords (or even a single password) kept in a file, once the file itself is revealed. Instead, we would hash the passwords and keep the results; hence whenever a user wants to be authenticated, only the hashed passwords are compared. Obviously, constructing a password from the corresponding hash value should be infeasible, which is guaranteed by the preimage resistance. Password hashing is still an important application of cryptographic hash functions.

Nevertheless, the best-known application of cryptographic hash functions is in the context of digital signatures. In a very general sense, digital signatures contain a message along with its signature, where the signature proves that the message was indeed signed by the authorized user. The main idea here for using a cryptographic hash function is to reduce the arbitrary length of the message to be signed to a certain amount so that the digest corresponds to a well-structured bit-string, say a group element. Now instead, the message digest is signed and attached to the original message, which can then be verified by checking the signature, as well as the hash value corresponding to the message. Note that here we assume that the description of the cryptographic hash function is public,

which is a valid assumption. From a security point of view, however, the least requirement for the hash function to be collision resistant; otherwise it would be easy to generate two distinct messages with the same signature by simply finding collisions for the hash function.

Another important feature of cryptographic hashing in the signature schemes is that it is often used to hide the 'algebraic' structure inherited in the digital signature algorithm. For instance, the product of two signed messages is equal to the signature of the product of the respective messages in the RSA signatures when hashing is omitted: Such an unfortunate property can be used to create a valid signature for any given message without even knowing the secret-key of any authorized user. The use of a 'good' hash function, however, is expected to provide extra scrambling, and thus prevents such a weakness from being exploited.

So, what is the key property? Many existing security proofs (e.g., [13, 63, 156]) use the so-called *random oracle model*; a model (formalized by Bellare and Rogaway [13]) that turns out to be very useful to argue about security yet, at the same time, impossible to realize in real life. A *random oracle* is a (random) function (typically from the set of arbitrary-length strings to the set of fixed-length strings) that is accessible to all parties and that responds (from the output space) uniformly and independently from all other outputs to every fresh query. Moreover, it consistently gives the same answer to a previously asked query. In practice, cryptographic hash functions are used to model random oracles [6]. Hence, a good cryptographic hash function is expected, in the ideal case, to 'behave' like a random oracle.

Digital signatures provide a satisfactory solution to the problem of authenticity and non-repudiation (when used with a time-stamp) and cryptographic hashing stays at the core of any currently used and standardized digital signature scheme. In the same direction, one of the most important and widely accepted applications of hashing is the *message authentication code* (often MAC) that is also used to authenticate a message; yet this time in a symmetric-key manner (i.e., using only a secret-key). The currently deployed NIST standard HMAC [140] is constructed using a cryptographic hash function.

Finally, cryptographic hashing is also used to provide data integrity, yet another notion that cryptography is trying to address nowadays. Indeed, hash functions are quite useful in determining whether any alteration has been made to a message. For example, two parties trying to exchange a file can ensure that the file has not been tampered with by simply checking the hash of the corresponding file (this also holds for checking the integrity of a software package).

These applications show that cryptographic hashing is crucial for a wide spectrum of cryptographic applications ranging from symmetric-key to public-key cryptography. Obviously, we do not cover here all the applications of cryptographic hash functions nor the future possibilities. A good reference for this direction is [122], which contains a wide variety of practical and theoretical directions in this field.

6. In [42] (see also [117]), it is shown that there exist encryption and signature schemes that are provable secure in the random oracle model yet they turn out to be insecure when instantiated by any concrete cryptographic hash function. Similar "uninstantiability" results are presented in [11, 43, 144]. Although the encryption and signature schemes presented in [42] are "unnatural" and a similar statement does not hold for the well-known schemes that appear in the literature, we should note that the security proved in the random oracle model does not necessarily imply security in practical applications.

1.2.2 A Quick Glance at Cryptographic Hash Function Research

To the best of our knowledge, the first concrete hash function construction was given by Rabin who used DES [162]: an iterated blockcipher-based construction inspiring several other similar hash functions in the early 1980s (see [122, 159, 177]). These constructions can be regarded as generic blockcipher-based designs; generic in the sense that any blockcipher can be used to instantiate them. The schemes considered in this thesis can also be regarded as designs in this direction. Specifically, they are based on some primitives e.g., blockciphers (for a more detailed discussion on primitive-based constructions, we refer to Chapter 2). All of the hash functions known today are designed as such because of their efficiency and provable properties.

Towards the end of the 1980s, Rabin's iterative method was improved independently by Merkle [125] and Damgård [52], who both show that hash functions can enjoy formal security proofs and profit from compact implementations. The main (high-level) ingredient of the Merkle–Damgård mode of operation is the use of a small fixed-input-length *compression function* (inspired by Rabin) that is used to compress (iteratively) independent blocks of messages at a time. Merkle and Damgård provide as a bonus "a proof" showing that the hash function created via their method turns out to be at least as collision resistant as the compression function (see Chapter 2). This result is useful not only for theoretical reasons but also from the point of practitioners, as it allows them to reduce the task of obtaining a good hash function to designing a good (i.e., efficient and secure) compression function.

Almost all the designs known today follow the Merkle–Damgård paradigm and the research focus is mainly on the design and analysis of compression functions. In this same line, one of the first concrete constructions is considered to be the MD-family (MD stands for "message digest"), designed by Rivest, which influenced many other follow-up designs. The earliest publicly available MD-members date back to MD2 [165] (1988) and MD4 [86] (1990) followed by the most famous (and strengthened) successor [7]: MD5 [166] (1991). The striking feature of the MD-family of hash functions is that they are all known to be extremely fast on modern computers (in particular on 32-bit platforms), mainly due to their highly efficient compression functions; they outperform DES (MD5 is around ten times faster than DES [18, 148]). Impressed by this fact, and to increase the short digest size of MD5 (128-bit), NIST issued two standards—SHA-0 [135] and SHA-1 [136]—in the early 1990s by modifying MD5 a bit in order to meet the requirements of a secure and an efficient 160-bit hash function.

Unfortunately, for SHA members, the story was not that different from the advances in symmetric-key cryptology: Beginning in the 1990s, MD-members, as well as SHA-0 were shown to have weak components [30, 46, 55]. Yet, the proposed attacks could be used to break only the underlying compression functions (note that the attack on compression functions are not guaranteed to carry over to the hash function); extending the attack to the full blown hash function was not straightforward.

In 2004, Wang et al. [202, 203] presented the first (non-trivial) collision-finding methods for both MD5 and SHA-1 (see also [44, 84, 84, 113, 194, 195, 204]): Their attacks—mainly exploiting the weaknesses of the compression functions of MD5 and SHA-1—are based on clever use of the previously mentioned *differential cryptanalysis*. Interestingly, beginning in 2004, the generic Merkle–Damgård construction was shown to have some unknown and trivial security flaws as well (see Section 2.2.2).

7. The latest member of the MD-family is MD6 [169], submitted to the SHA-3 competition.

As a result, NIST decided to launch a new competition, called "the SHA-3 competition [145]", in order to meet the requirements of a modern hash function. Although NIST had already started moving from SHA-1 to SHA-2 [141], the internal features of SHA-2 are not very different from its predecessor; hence similar weaknesses are expected to be inherited by SHA-2. Besides, SHA-2 does not support the expected flexibility and compactness in terms of efficiency on multiple platforms. So, it is a widely held opinion that SHA-3 is indeed necessary (in spite of no known security weaknesses of SHA-2).

SHA-3 selection is still in progress; the new hash function standard will be decided in 2012 among the five finalists (Blake [10], Grøstl [67], Keccak [19], JH [206] and Skein [62]) that were declared in late 2010. The research community will therefore be busy in the following years analyzing SHA-3 candidates and understanding their primitives better. Due to these developments, cryptographic hashing has become a more active area of research; in particular, new practical hash function designs with sound security claims are now the basis of this renewed interest. The results presented in this thesis can also be seen as developments in this direction.

1.3 About This Dissertation

The problems we discuss in this thesis are also based on the latest advances in cryptographic hash function research. More concretely, we try to find effective solutions to a question that has been known for quite some time and which we further describe here (see Chapter 2 for an extensive background). Assume we are given one or more 'secure' [8] blockciphers (or simply a function): How can we construct a secure compression or hash function out of it? The basis for such a question is simple: Blockciphers have been 'the primitives' since the early years of modern cryptology and they have been used in many applications. Therefore, it would be convenient to make use of them to create a hash function as well, because one blockcipher would be sufficient to obtain an encryption scheme and a hash function simultaneously. Moreover, it would be nice to transfer the trust in the blockcipher, as well as its performance, to the hash function built out of it.

The wisdom of blockcipher-based hashing is still valid today. Indeed, the current cryptographic hash function standard SHA-2 and some of the SHA-3 candidates are, or can be regarded as, blockcipher-based designs. This is mainly because of the (positive) theoretical results on blockcipher-based hashing achieved in the last two decades (see Chapter 2 for the related background) although the major underlying assumption above is still far from being realistic (i.e., how can you make sure a blockcipher is 'secure'?). Nevertheless, such an approach can still be used to develop a good indication about the structural properties of a hash function and to provide a first step towards *arguing* for its security, and of course its performance.

To this end, we study the collision and preimage resistance of certain practical, relevant hash function constructions (created via the Merkle–Damgård iteration) with compression functions that make multiple calls (multi-call) to a well-known cryptographic primitive (in particular a blockcipher) and produce digests of length more than (multi-block-length) the output of the underlying primitive [9] (e.g., the block-size). Our contribution, from a very high-level, can be divided into two categories:

8. See the discussion in Section 2.3.1 for the precise assumptions on the underlying blockciphers used in this thesis.
9. We are not the first to work on these type of problems; there has been an extensive effort since the late 1980s. See Chapter 2 for related background.

(i) the design of new hash functions that are supported by provable properties; and (ii) the security assessment and more conceptual understanding of already existing constructions. We give a more detailed, technical description of our contribution in Section 2.4, after we have developed necessary background on cryptographic hashing.

1.3.1 Publications

During my doctoral studies at EPFL, I have been involved in different research projects that, thanks to the very talented and smart people I worked with, resulted in several publications in international cryptology conferences. For several reasons, however, not all of my publications are included in this thesis; nevertheless, to show my gratitude to my co-authors, I listed them all here.

The following four papers are contained in the subsequent chapters of this thesis.

- [150] Onur Özen, Martijn Stam: Another Glance at Double-Length Hashing: In: Parker, M.G. (ed.) Cryptography and Coding, 12th IMA International Conference, Cryptography and Coding 2009, Cirencester, UK, December 15-17, 2009. Lecture Notes in Computer Science, vol. 5921, pp. 176-201. Springer, Heidelberg (2009).
- [149] Onur Özen, Thomas Shrimpton, Martijn Stam: Attacking the Knudsen–Preneel Compression Functions: In: Hong, S., Iwata, T. (eds.) Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010. Lecture Notes in Computer Science, vol. 6147, pp. 94-115. Springer, Heidelberg (2010).
- [151] Onur Özen, Martijn Stam: Collision Attacks against the Knudsen–Preneel Compression Functions: In: Abe, M. (ed.) Advances in Cryptology, ASIACRYPT 2010, 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Lecture Notes in Computer Science, vol. 6477, pp. 76-93. Springer, Heidelberg (2010).
- [81] Dimitar Jetchev, Onur Özen, Martijn Stam: Collisions are not Incidental: A Compression Function Exploiting Discrete Geometry: To appear in the Proceedings of the 9th Theory of Cryptography Conference, TCC 2012.

The paper [149], written along with Martijn Stam and Thomas Shrimpton, was awarded “best paper” by the program committee of FSE 2010 and invited as a submission to the Journal of Cryptology.

The following paper, written with Joppe W. Bos and Jean-Pierre Hubaux, was published as an outcome of the course “Security and Cooperation in Wireless Sensor Networks” given by Prof. Hubaux at EPFL in the fall of 2008.

- [34] Joppe W. Bos, Onur Özen, Jean-Pierre Hubaux: Analysis and Optimization of Cryptographically Generated Addresses. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) Information Security, 12th International Conference, ISC 2009, Pisa, Italy, September 7-9, 2009. Lecture Notes in Computer Science, vol. 5735, pp. 17–32. Springer, Heidelberg (2009).

After I joined EPFL, I continued some of the work that I had started before my doctoral studies. As a result, I was involved in certain cryptanalysis efforts (with various co-authors) that were published in two conference proceedings:

- [152] Onur Özen, Kerem Varıcı, Cihangir Tezcan, Çelebi Kocair: Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT. In: Boyd, C., Nieto, J.M.G. (eds.) Information Security and Privacy, 14th Australasian Conference, ACISP 2009, Brisbane, Australia, July 1-3, 2009, Lecture Notes in Computer Science, vol. 5594, pp. 90-107. Springer, Heidelberg (2009).

- [9] Jean-Philippe Aumasson, Çağdaş Çalık, Willi Meier, Onur Özen, Raphael C.-W. Phan, Kerem Varıcı: Improved Cryptanalysis of Skein. In: Matsui, M. (ed.) Advances in Cryptology, ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Lecture Notes in Computer Science, vol. 5912, pp. 542-559. Springer, Heidelberg (2009).

Finally, to some extent related to the work presented here, I contributed to the following paper that appeared in CHES 2011 where we investigate the performance of the well-known primitive-based hash functions on a modern architecture supporting the AES instruction set.

- [35] Joppe W. Bos, Onur Özen, Martijn Stam: Efficient Hashing Using the AES Instruction Set. In: Preneel,B., Takagi,T. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2011, 13th International Workshop, Nara, Japan, September 28-October 1, 2011. Lecture Notes in Computer Science, vol. 6917, pp. 507-522. Springer, Heidelberg (2011).

1.3.2 Organization

The thesis is organized as follows. In Chapter 2, we review the cryptographic hash function literature and introduce necessary formalizations by mainly focusing on the major interests of this work. In Chapter 3, we address the tools required in the subsequent chapters. In Chapters 4, 5 and 6 we present the main contributions of this thesis. Specifically, Chapter 4 and 5 are based on and extend the papers [150] and [81], respectively. In Chapter 6, we discuss the joint-version of the two papers [149] and [151]. Finally, we conclude the thesis in Chapter 7. We provide auxiliary material in the Appendices.

2 Cryptographic Hash Functions

In this chapter, we discuss the recent trends and developments in the research on cryptographic hash functions, along with the detailed, technical contributions of this work. Contrary to the informal introduction given in Chapter 1 where we mainly discussed their importance and current applications, here we dive more into the technical details. We first formally define cryptographic hash functions and related security properties. Then, we look at the existing constructions and their security levels. Our major focus is on the *primitive-based* constructions obtained in an iterated fashion, where the primitive itself is either a blockcipher or simply a function drawn uniformly at random from the set of all blockciphers or functions (with specified domain and range, domain being larger than the range), respectively. One of the other primitive-based compression functions, permutation-based constructions, is briefly recapitulated in Appendix B.

The chapter is organized as follows: In Section 2.1 we start with the model we follow and the basics of cryptographic hash functions. In Section 2.2 we introduce iterated hash functions, in particular the celebrated Merkle–Damgård domain extension (and similar other domain extenders) along with its security properties. In the following section (Section 2.3) we discuss blockcipher-based hash functions, the major background for this work. We conclude the chapter with the detailed contributions of this thesis (Section 2.4).

2.1 Preliminaries

2.1.1 Basic Notions

Informally (more formal treatment will follow shortly), a cryptographic hash function can be defined as a function $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^n$, for some fixed integer $n > 0$, mapping strings of arbitrary length (for the moment we assume a message space of $\{0,1\}^*$, the set of strings of arbitrary length) into fixed length strings that satisfy several security properties. Traditionally, cryptographic hash functions are expected to satisfy three security notions [122]:

1. Collision resistance: It is computationally infeasible to find two inputs $x' \neq x$ (from the domain of H) such that $\mathsf{H}(x) = \mathsf{H}(x')$.
2. Preimage resistance: For any pre-specified (and valid) output $y \in \{0,1\}^n$ (i.e.,there exists at least one $x \in \{0,1\}^*$ such that $\mathsf{H}(x) = y$) for which a corresponding input is not known, it is computationally infeasible to find an x in the domain of H such that $\mathsf{H}(x) = y$.

3. Second-preimage resistance: For any specified input $x \in \{0,1\}^*$, it is computationally infeasible to find yet another $x' \in \{0,1\}^*$ (i.e., with $x' \neq x$) that satisfies $\mathsf{H}(x) = \mathsf{H}(x')$.

The ambiguity (as pointed out below) of the traditional formulation of a hash function and the following notions of security were perceived soon after their appearance in the literature and motivated researchers to find a more rigorous and mathematically sound treatment (e.g., the work of Rogaway and Shrimpton [171], as well as [7, 164, 175]). The main problem with the above definition is that it does not allow for a rigorous formalization for collision resistance (similar problems exist for (second)-preimage resistance as well). Rogaway [175] calls this *foundations-of-hashing dilemma*: By the pigeonhole principle, collisions do exist for sure (as long as the message space is large enough) and they are simply hard-coded in the function specification. The dilemma stems from the fact that we, human beings, cannot write down the collision-finding algorithm very easily and might not know any colliding pairs. Consequently, it is difficult to formally write down the security definitions. In order to remedy this, we treat a hash function H as a family of functions, also known as *keyed-hash functions*, where the term 'key' here is not used to refer to a secret component. In contrast, it is made public, chosen from a finite set and used for randomization purposes to enable rigorous mathematical treatment. The following definition makes the term more concrete.

Definition 2.1.1. A hash function (family) is a function $\mathsf{H} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$, for a positive integer n, where $\mathcal{K}$ is a finite non-empty set and each $K \in \mathcal{K}$ defines a function $\mathsf{H}_K(\cdot) = \mathsf{H}(K, \cdot)$; here H can be thought of as a family of hash functions $\mathsf{H} = \{\mathsf{H}_K \mid K \in \mathcal{K}\}$, each key (or index) $K \in \mathcal{K}$ specifying one. The number n is called the hash length or digest size of H.

As hash functions are used as input to other algorithms (e.g., digital signature algorithms) we can see them as algorithms. Thus, after refining the above definition, we arrive at the following formalization for a *keyed-hashed function*: A *keyed-hash function* consists of a pair of algorithms $\mathcal{K}$ and H where $\mathcal{K}$ is a probabilistic algorithm taking no input and producing the key K (from a finite non-empty set), whereas H is a deterministic (and efficient) algorithm that takes K and a string $X \in \{0,1\}^*$ as input and produces a digest of some fixed length (say n). Here, we assume the existence of a message space $\mathcal{X}$ where $X \in \mathcal{X} \subseteq \{0,1\}^*$; otherwise we simply assign $\mathsf{H}_K(X)$ as undefined for any $X \notin \mathcal{X}$. In this work, we do not consider such $X \notin \mathcal{X}$. In the following, whenever we write $X \in \{0,1\}^*$, we mean the existence of a message space $\mathcal{X}$ where $X \in \mathcal{X} \subseteq \{0,1\}^*$.

In order to proceed with the reformulated security definitions, we recall the concept *adversary*. An adversary is an algorithm that aims to break a well-defined security notion using some (bounded) resources. In this section, we set as a convention that the only resource used by the adversary is its runtime; for other hash function constructions, in particular the constructions based on some primitives, we will change this restriction. We define (under some RAM model of computation) $t_{\mathsf{H},\mathcal{X}}$ to be the minimum, over all programs P_{H} that compute H, of the length of P_{H} plus the worst-case running time of P_{H} over all inputs (K, X) where $K \in \mathcal{K}$ and $X \in \mathcal{X}$; plus the the minimum, over all programs P_K that sample the key from $\mathcal{K}$, of the time to compute the key plus the size of P_K.

In general, the performance of an adversary $\mathcal{A}$ against breaking a security notion xxx is measured by its advantage $\mathsf{Adv}^{\mathrm{xxx}}_{\mathsf{H}}(\mathcal{A})$ which is quantified by a probability (using a specific source of randomness). Hence, it is a real number in $[0,1]$; if the probability is zero, then it means that $\mathcal{A}$ has done a bad job and has no advantage at all in achieving xxx, whereas if it is one, $\mathcal{A}$ always succeeds. By $\mathsf{Adv}^{\mathrm{xxx}}_{\mathsf{H}}(t)$ we denote the maximum advantage over all adversaries that try to break the property xxx by using at most time t (for any bounded t).

2.1.2 Security Notions

Throughout, when $\mathscr{X}$ is a set, by $x \xleftarrow{\$} \mathscr{X}$ we mean the uniform random selection of an element x from $\mathscr{X}$. We let $y \leftarrow \mathscr{A}(x)$ be the assignment of y as an output of a deterministic algorithm $\mathscr{A}$ taking x as an input. Similarly, we denote by $y \xleftarrow{\$} \mathscr{A}(x)$ when $\mathscr{A}$ is probabilistic. These are the only notations required for our formalizations. As security notions, we discuss only (second) preimage and collision resistance; these are the main targets in this thesis [1].

Collision Resistance

The following definition provides the necessary formalization for collision resistance of a (keyed) hash function.

Definition 2.1.2. Let $\mathscr{A}$ be a collision-finding adversary against $\mathsf{H} : \mathscr{K} \times \{0,1\}^* \to \{0,1\}^n$. We define the collision-finding advantage of $\mathscr{A}$ as

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(\mathscr{A}) = \Pr\left[K \xleftarrow{\$} \mathscr{K}, (X, X') \xleftarrow{\$} \mathscr{A}(K) : X \neq X' \wedge \mathsf{H}_K(X) = \mathsf{H}_K(X')\right].$$

We call $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(t)$ the maximum collision-finding advantage over all adversaries that use at most time t. The hash function H is called (t, ϵ) coll-secure if no adversary using at most time t has an advantage more than ϵ.

Definition 2.1.2 makes sense only for the keyed-hash functions. However, the hash functions used in practice, e.g., SHA-2 are key-less. So, how do we argue collision resistance for the key-less setting? One way is to see the constant or the initialization vector (see [141] for the exact specification) used in SHA-2 as the key. Yet, as pointed out by Rogaway [175], it was never indicated by NIST that they were chosen to be so. Moreover, it is not clear whether they were picked randomly or not. In order to solve this discrepancy and define collision resistance in the key-less setting, Rogaway formalized his *human ignorance model*; for the details on the human ignorance model and the security for the key-less setting in general, we refer to [4, 175]. In this work, we either use keyed-hash functions or primitive-based constructions (Section 2.3) for which we provide corresponding definitions.

Preimage and Second-Preimage Resistance

Concerning (second) preimage resistance, the ambiguity of the informal definition still remains for the unkeyed definition. For instance, it is not clear how the output point $y \in \{0,1\}^n$ to be inverted is given; is it chosen uniformly at random or following another distribution? Similarly, how is the input $x \in \{0,1\}^*$ given to the adversary in the case of second-preimage resistance? These questions, along with the novel keyed-hash function definition, are the basis for several different preimage resistance notions. Here, we formally define only the most relevant one for our purposes: epre (everywhere preimage resistance). It states that, for whatever hash value is selected, it is computationally hard to find a preimage for it.

1. In this work, we do not consider a more general security concept, indifferentiability [117], introduced by Maurer et al. as an extension to the indistinguishability framework of Maurer [119]. We refer to [49] for the adaptation of indifferentiability in the context of cryptographic hash functions, in particular iterative hash functions, where it was shown how to argue that a hash function construction 'behaves' like a random oracle (an idealized object that is mimicked by a hash function).

Definition 2.1.3. Let $\mathsf{H} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$ be a hash function and $\mathscr{A}$ be an everywhere preimage-finding adversary against H. We define $\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{H}}(\mathscr{A})$ to be the everywhere preimage-finding advantage of $\mathscr{A}$:

$$\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{H}}(\mathscr{A}) = \max_{Y \in \{0,1\}^n} \left\{ \Pr\left[K \xleftarrow{\$} \mathcal{K}, X' \xleftarrow{\$} \mathscr{A}(K) : \mathsf{H}_K(X') = Y \right] \right\} .$$

We call $\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{H}}(t)$ the maximum everywhere preimage-finding advantage over all adversaries that use at most time t. The hash function H is called (t,ϵ) epre-secure if no adversary using time at most t has advantage more than ϵ.

Intuitively, the above definition states that any point in the range is hard to invert; in a way, epre measures the hardness of inverting a *fixed* challenge by using a *random key*. In addition to everywhere preimage resistance, Rogaway and Shrimpton [171] introduced two more notions related to preimage resistance: pre (preimage resistance) and apre (always preimage resistance). In pre, the experiment is carried out with a *random* key and *random* challenge, whereas for apre the key is *fixed* yet the challenge is *random*:

$$\mathsf{Adv}^{\mathsf{pre[m]}}_{\mathsf{H}}(\mathscr{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K}, X \xleftarrow{\$} \{0,1\}^m, Y \leftarrow \mathsf{H}_K(X), X' \xleftarrow{\$} \mathscr{A}(K,Y) : \mathsf{H}_K(X') = Y \right] ,$$

$$\mathsf{Adv}^{\mathsf{apre[m]}}_{\mathsf{H}}(\mathscr{A}) = \max_{K \in \mathcal{K}} \left\{ \Pr\left[X \xleftarrow{\$} \{0,1\}^m, Y \leftarrow \mathsf{H}_K(X), X' \xleftarrow{\$} \mathscr{A}(Y) : \mathsf{H}_K(X') = Y \right] \right\} .$$

The always preimage resistance notion is particularly important for the practical hash functions (e.g., SHA-2) as it captures that a hash function is preimage resistant regardless of the choice of the key (i.e., it is preimage resistant for all members of the hash function family). In a way, apre can be seen as the natural extension to Rogaway's human-ignorance approach. Similar to the preimage resistance, Rogaway and Shrimpton framework also define notions for second-preimage resistance: sec (random key, random challenge), esec (random key, fixed challenge) and asec (fixed key, random challenge) [2]:

$$\mathsf{Adv}^{\mathsf{sec[m]}}_{\mathsf{H}}(\mathscr{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K}, X \xleftarrow{\$} \{0,1\}^m, X' \xleftarrow{\$} \mathscr{A}(K,X) : X \neq X' \wedge \mathsf{H}_K(X) = \mathsf{H}_K(X') \right] ,$$

$$\mathsf{Adv}^{\mathsf{esec[m]}}_{\mathsf{H}}(\mathscr{A}) = \max_{X \in \{0,1\}^m} \left\{ \Pr\left[K \xleftarrow{\$} \mathcal{K}, X' \xleftarrow{\$} \mathscr{A}(K) : X \neq X' \wedge \mathsf{H}_K(X) = \mathsf{H}_K(X') \right] \right\} ,$$

$$\mathsf{Adv}^{\mathsf{asec[m]}}_{\mathsf{H}}(\mathscr{A}) = \max_{K \in \mathcal{K}} \left\{ \Pr\left[X \xleftarrow{\$} \{0,1\}^m, X' \xleftarrow{\$} \mathscr{A}(X) : X \neq X' \wedge \mathsf{H}_K(X) = \mathsf{H}_K(X') \right] \right\} .$$

We note that the notion esec is equivalent to the universal one-way hash functions of Naor and Yung [133], as well as the target collision resistance of Bellare and Rogaway [15].

Implications and Separations

One of the important results of the Rogaway–Shrimpton framework is the study of implications (and separations) between the newly introduced security notions. Informally, the security notion xxx implies yyy means that if H is secure in the xxx sense then it is also secure in terms of yyy. The separation, however, means that a security notion xxx can be achieved without being secure in the yyy sense (see Definitions 2.1.4 and 2.1.5 for a more formal treatment). Following this model, Rogaway and Shrimpton provide a picture (shown in Figure 2.1) illustrating all the implications and

2. Recently, Andreeva and Stam pointed out some extensions on the above mentioned preimage resistance notions, we refer to [7] for more on these additional concepts.

separations regarding seven security notions (i.e., coll, pre, epre, apre, sec, esec, asec). Extensions to the implication and separation results of Rogaway and Shrimpton are proposed in [7, 164].

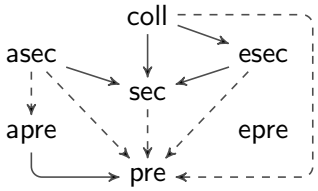

Figure 2.1 – Implication and separation (see Definitions 2.1.4 and 2.1.5) results among seven notions introduced by Rogaway and Shrimpton. Solid arrows show conventional implications, whereas dashed arrows represent provisional implications (see Definition 2.1.4); the strength of the implication is dependent on the size of the domain and range (see [171] for the details). No arrows represent separation. We note that each security notion implies itself (trivially) although it is not shown in the figure explicitly.

Let us discuss more formally what we mean by an implication and separation. Fix $\mathcal{K}$ and n; moreover, consider a message space of $\mathcal{X}$ (rather than $\{0,1\}^*$) where $\{0,1\}^m \subseteq \mathcal{X}$ for a fixed positive integer m. Therefore, the family of hash functions in question is defined as $\mathsf{H} : \mathcal{K} \times \mathcal{X} \to \{0,1\}^n$.

Definition 2.1.4 (Implications [171])**.** We say that xxx implies (conventionally) yyy if for any bounded t, there exists an absolute constant c such that

$$\mathsf{Adv}^{\mathsf{yyy}}_{\mathsf{H}}(t) \leq c\mathsf{Adv}^{\mathsf{xxx}}_{\mathsf{H}}(t')$$

for all hash functions $\mathsf{H} : \mathcal{K} \times \mathcal{X} \to \{0,1\}^n$, where $t' = t + ct_{\mathsf{H},\mathcal{X}}$. If

$$\mathsf{Adv}^{\mathsf{yyy}}_{\mathsf{H}}(t) \leq c\mathsf{Adv}^{\mathsf{xxx}}_{\mathsf{H}}(t') + \epsilon\,,$$

for some $\epsilon > 0$, we say that xxx implies (provisionally) yyy to ϵ.

In the provisional implications given by Rogaway and Shrimpton the value of ϵ is 2^{n-m} (see above for the definition of m and n). Thus, the implication result is tied to the amount of compressing. For *normal* hash functions, m is chosen to be sufficiently large; hence ϵ is very close to zero.

Definition 2.1.5 (Separations [171])**.** We say that xxx is separated (conventionally) from yyy to ϵ if for any $\mathsf{H} : \mathcal{K} \times \mathcal{X} \to \{0,1\}^n$ and (bounded) t, there exists an $\mathsf{H}' : \mathcal{K} \times \mathcal{X} \to \{0,1\}^n$ and an absolute constant c such that

$$\mathsf{Adv}^{\mathsf{xxx}}_{\mathsf{H}'}(t) \leq c\mathsf{Adv}^{\mathsf{xxx}}_{\mathsf{H}}(t') + \epsilon$$

and yet $\mathsf{Adv}^{\mathsf{yyy}}_{\mathsf{H}'}(t') = 1$, where $t' = t + ct_{\mathsf{H},\mathcal{X}}$. If there exists an $\mathsf{H} : \mathcal{K} \times \mathcal{X} \to \{0,1\}^n$ such that

$$\mathsf{Adv}^{\mathsf{xxx}}_{\mathsf{H}}(t) \leq \epsilon\,,$$

for all t and yet $\mathsf{Adv}^{\mathsf{yyy}}_{\mathsf{H}}(t') \leq 1$, for $t' = ct_{\mathsf{H},\mathcal{X}}$, then xxx is separated (unconditionally) from yyy to ϵ.

In Figure 2.1, separation is indicated whenever one of the separations given above holds. We refer to [7] for a more recent and generalized version of Figure 2.1.

2.1.3 Generic Cryptanalytic Methods

Based on the security notions given above, we now investigate the generic cryptanalytic methods to obtain a (second) preimage and collision with a non-negligible probability[3]. These methods are generic in the sense that, regardless of the construction, they always work; it is impossible to prevent them. Resistance against such methods is parametrized by the digest size n. To illustrate, for a hash function, the generic (second) preimage-finding method is expected to require 2^n hash function evaluations. Collisions, however, can be found with high probability after an effort of $2^{n/2}$ (see below for the corresponding (second) preimage- and collision-finding algorithms). Constructions shown to be resistant up to the stated bounds are deemed optimal in terms of the corresponding security notions.

Finding Collisions

The generic method to find collisions is the so-called birthday attack (based on the birthday paradox), first pointed out by Yuval [208] in the context of hash functions. Under the assumption that all the outputs are equally likely[4], the birthday paradox states (for a randomly chosen key) that the expected number of required messages to find a collision is $N = \sqrt{\pi/2} \cdot 2^{n/2}$ (see Appendix A for the justification). Hence, the (optimal) collision resistance is tied to the square root of the cardinality of the range of the hash function.

The standard birthday attack (Algorithm 2.1.6) uses this simple idea: A collision-finding adversary $\mathscr{A}$ picks $N = \sqrt{\pi/2} \cdot 2^{n/2}$ input points randomly (from a finite subset $\mathscr{X}$ of $\{0,1\}^*$), and applies H_K to all of them. If two distinct inputs create the same output, then the collision is found. By the birthday paradox, we know that, with a high probability, a collision is hidden somewhere among the outputs; we need to simply search for the collision. But, how? One method is to store the hash outputs in a hash table; hence in constant time we can add new output points and check for collisions. The computational complexity of this method (asymptotically in n) is $\mathscr{O}(2^{n/2})$ time and $\mathscr{O}(2^{n/2})$ memory. It seems that the time-complexity of the above collision-finding method is somewhat optimal; yet the memory requirements are too much; this can be reduced using Pollard's rho method [157,158], which benefits from Floyd's algorithm [96] (see also [146]) for cycle-detection. Pollard's rho method can be parallelized, as shown in [147], which is one of the most practical techniques in today's cryptanalytic applications.

Algorithm 2.1.6 ((Standard) Birthday Attack)**.**

Input: $\mathsf{H} : \mathscr{K} \times \mathscr{X} \to \{0,1\}^n$ *and* $K \xleftarrow{\$} \mathscr{K}$ *(where* $|\mathscr{X}| \geq \sqrt{\pi/2} \cdot 2^{n/2}$*).*

Output: $X, X' \in \mathscr{X}$ *such that* $X \neq X'$ *and* $\mathsf{H}_K(X) = \mathsf{H}_K(X')$.

For $i = 1, \ldots, N = \sqrt{\pi/2} \cdot 2^{n/2}$ *do*

a. Pick $X_i \xleftarrow{\$} \mathscr{X}$*; Calculate* $\mathsf{H}_K(X_i) = Y_i$*;*

b. If $\exists\ j < i$ *such that* $Y_i = Y_j$ *but* $X_i \neq X_j$*, then* $X \leftarrow X_i$*,* $X' \leftarrow X_j$*. Return* X *and* X'*.*

Return fail.

3. We call a function $\nu : \mathbb{N} \to [0,1]$ non-negligible if for any $k > 0$, it holds that $\nu(x) \geq 1/n^k$ for infinitely many x.

4. This assumption, in fact, is not true in full generality for hash functions: It actually assumes that the number of preimages of each hash function output is the *same*. More specifically, it assumes that the hash function is *regular*. Nevertheless, it is not an invalid argument as it turns out, for irregular hash functions, that the birthday attack succeeds apparently even faster [14].

Finding (Second) Preimages

An obvious method to look for (second) preimages is the following: Let $\mathscr{A}$ be an everywhere preimage-finding adversary trying to invert a fixed challenge $Y \in \{0,1\}^n$ for a randomly chosen key $K \in \mathscr{K}$. The adversary $\mathscr{A}$ simply picks randomly the inputs X from a finite subset $\mathscr{X}$ of $\{0,1\}^*$) and checks the condition $\mathsf{H}_K(X) = Y$. As $Y \in \{0,1\}^n$, a particular $X \in \mathscr{X}$ hashes to Y with a probability of 2^{-n}. So, after about 2^n different trials (hash function evaluations), we expect to end up with a preimage. We note that a similar method (with the same amount of resources) works for finding second preimages. Algorithm 2.1.7 can be parallelized and it requires negligible memory.

Algorithm 2.1.7 (Generic (Second) Preimage Attack)**.**

Input: $\mathsf{H} : \mathscr{K} \times \mathscr{X} \to \{0,1\}^n$, $K \xleftarrow{\$} \mathscr{K}$ *and the range point* $Y \in \{0,1\}^n$ *to be inverted (where* $|\mathscr{X}| \geq 2^n$*).*

Output: $X \in \mathscr{X}$ *such that* $\mathsf{H}_K(X) = Y$.

For $i = 1, \ldots, 2^n$ *do*

a. Pick $X \xleftarrow{\$} \mathscr{X}$ *and calculate* $\mathsf{H}_K(X)$*;*

b. If $\mathsf{H}_K(X) = Y$*, return* X.

Return fail.

2.2 Iterated Hash Functions

The way to construct a VIL (variable input length) hash function H that uses a smaller FIL (fixed input length) primitive h, called *compression function* as an underlying component, is called a *domain extension* or a *mode of operation*. All of the hash function designs known today use domain extenders; in particular, the compression function is called in a cascaded manner. We call these constructions *iterated hash functions*. Informally, iterative hash functions work as follows. First a (carefully) chosen padding scheme pad is used to make the message length become a multiple of a positive integer, say m. Then the message is divided into m-bit blocks. Later, starting from an initialization vector (IV), we iteratively call the compression function $\mathsf{h} : \mathscr{K} \times \{0,1\}^s \times \{0,1\}^m \to \{0,1\}^s$ (for positive integers m and s) to compress a block of messages at a time by using the output of the previous compression function evaluation as one of the inputs to the fresh call. Finally, an optional output transformation g is used to reduce the final output size to the desired digest size (see Figure 2.2 for illustration). Definition 2.2.1 provides a more formal treatment.

Definition 2.2.1. Let $\mathscr{K}$ be a finite, nonempty set and let $\mathscr{Z} = \{0,1\}^n, \mathscr{V} = \{0,1\}^s, \mathscr{M} = \{0,1\}^m$ for positive integers m, n and s. We call $\mathsf{H}^{\mathsf{h},\mathsf{g}} : \mathscr{K} \times \mathscr{V} \times \{0,1\}^* \to \mathscr{Z}$ a (generalized) iterative hash function with underlying compression function $\mathsf{h} : \mathscr{K} \times \mathscr{V} \times \mathscr{M} \to \mathscr{V}$ and (any) function $\mathsf{g} : \mathscr{V} \to \mathscr{Z}$ if the digest $Z \in \mathscr{Z}$ is computed, for an input $X \in \{0,1\}^*$, $K \in \mathscr{K}$, positive integer ℓ and initialization vector $\mathsf{IV} \in \mathscr{V}$, as follows

1. $v_0 \leftarrow \mathsf{IV}$, $\mathsf{M} \leftarrow \mathsf{pad}(X)$;
2. Divide M into ℓ blocks as $\mathsf{M} = (M_1, \ldots, M_\ell)$, where $M_i \in \mathscr{M}$ for $i = 1, \ldots, \ell$;
3. For $i = 1, \ldots, \ell$ compute $v_i = \mathsf{h}(K, v_{i-1}, M_i)$;
4. Output $Z = \mathsf{g}(v_\ell)$.

Here $\mathsf{pad} : \{0,1\}^* \to \mathscr{M}^+$ is a fixed (injective) padding function.

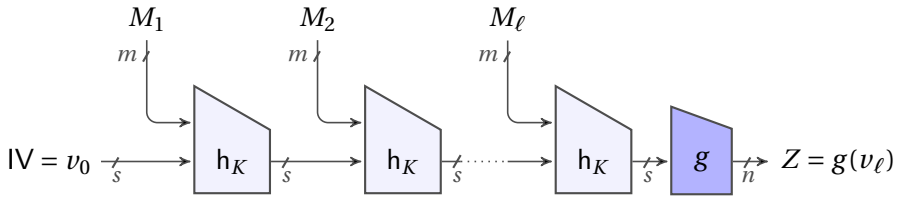

Figure 2.2 – The iterative hash function is illustrated with the compression function h_K (i.e., h with $K \xleftarrow{\$} \mathcal{K}$) for $\mathcal{M} = \{0,1\}^m, \mathcal{V} = \{0,1\}^s$ and $\mathcal{Z} = \{0,1\}^n$ (see Definition 2.2.1).

The main reason for iterative hashing is intuitive: Smaller independent message blocks can be compressed at each compression function evaluation without a need to store and process the entire message. On the one hand, this allows for compact implementations due to smaller primitives; on the other hand, the compression functions have to be processed iteratively and cannot be parallelized, which is sometimes considered as a disadvantage.

One of the major goals of the research efforts on iterative hash functions is to relate the security of the (iterative) hash function with that of the underlying compression function. To formally define the security of a compression function, however, we need to extend our security definitions (coll and epre) to the keyed-compression function setting. Definition 2.2.2 and 2.2.3 serve for this purpose.

Definition 2.2.2. Let $\mathscr{A}$ be a collision-finding adversary against the compression function $\mathsf{h} : \mathcal{K} \times \mathcal{V} \times \mathcal{M} \to \mathcal{V}$. Then the collision-finding advantage of $\mathscr{A}$ is defined as

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(\mathscr{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K}, \big((V,M),(V',M')\big) \xleftarrow{\$} \mathscr{A}(K) : (V,M) \neq (V',M') \wedge \mathsf{h}(K,V,M) = \mathsf{h}(K,V',M')\right].$$

We call $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(t)$ the maximum collision-finding advantage over all adversaries that use at most time t.

Definition 2.2.3. Let $\mathsf{h} : \mathcal{K} \times \mathcal{V} \times \mathcal{M} \to \mathcal{V}$ be a compression function and $\mathscr{A}$ be an everywhere preimage-finding adversary against h. We define $\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{H}}(\mathscr{A})$ to be the everywhere preimage-finding advantage of $\mathscr{A}$:

$$\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(\mathscr{A}) = \max_{Y \in \{0,1\}^n} \left\{\Pr\left[K \xleftarrow{\$} \mathcal{K}, (V',M') \xleftarrow{\$} \mathscr{A}(K) : \mathsf{h}(K,V',M') = Y\right]\right\}.$$

We call $\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(t)$ the maximum everywhere preimage-finding advantage over all adversaries that use at most time t.

Similarly, for both notions, we say that the compression function h is called (t,ϵ) xxx-secure (for $\mathsf{xxx} \in \{\mathsf{coll}, \mathsf{epre}\}$) if no adversary using at most time t has advantage more than ϵ. Finally, we note that the generic attacks presented in Section 2.1.3 work against the compression functions as well.

2.2.1 Merkle–Damgård Domain Extension

In [125] and [52], Merkle and Damgård introduce independently an iterated hash function construction (using g as the identity map) for which they show a revolutionary result: The hash function inherits its collision resistance from the underlying compression function (see Theorem 2.2.5 for the

precise statement). More specifically, once it is assumed that the compression function is optimally collision resistant, then so is the corresponding iterative hash function (note that the reverse statement is not guaranteed to carry over). This result is extremely important from the designers' point of view; indeed, they do not need to make the effort to design the full-blown hash function from scratch and can simply focus on designing (secure) compression functions. Then the Merkle–Damgård domain extension takes care of the rest.

The main difference between Merkle's [125] and Damgård's [52] construction is that they proposed two different padding schemes, without affecting their major collision resistance preservation result. For concreteness, let us recall both techniques.

1. **Merkle's Padding Rule [125]:** In Merkle's padding rule, the message space is restricted to $\{0,1\}^{2^b-1}$, for some fixed positive integer b, rather than the set of arbitrary length messages. In practice, b is chosen to be sufficiently large, e.g., $b = 64$ as in SHA-2, and restricting the message space to $\{0,1\}^{2^{64}-1}$ is not a real problem. Merkle's idea is to append the binary encoding of the message length to the last message block in order to let his proof work; his padding algorithm works as follows. Let $\mathcal{M} = \{0,1\}^m$ and $\mathsf{len}_b(X)$ represent the b-bit binary representation of $|X|$. Then
$$\mathsf{pad}_{\mathsf{merkle}}(X) = X \,||\, 10^d \,||\, \mathsf{len}_b(X)\ ,$$
where d is the least positive integer such that $|X| + 1 + d + \mathsf{len}_b(X)$ is a multiple of m.
2. **Damgård's Padding Rule [52]:** Contrary to Merkle's padding rule, Damgård's padding scheme can handle messages of arbitrary length; on the negative side, it requires more padded bits for long messages, which may lead to less efficient constructions. Let $\mathcal{M} = \{0,1\}^m$ and $X \,||\, 0^d = X_1 \,||\ldots||\, X_\ell$ for some positive integer ℓ, where $|X| + d$ is a multiple of $m-1$ (d being the least positive integer that satisfies this property) and $X_i \in \{0,1\}^{m-1}$ (for $i \in \{1,...,\ell\}$). Then
$$\mathsf{pad}_{\mathsf{damgård}}(X) = 0 \,||\, X_1 \,||\, 1 \,||\, X_2 \,||\ldots||\, 1 \,||\, X_\ell \,||\, \mathsf{len}_m(d).$$

Due to efficiency reasons, in practice, most of the well-known cryptographic hash functions (e.g., SHA-2) use Merkle's padding scheme; this version is often called Merkle–Damgård with length strengthening or strengthened Merkle–Damgård [102] (sMD) as referred by Lai and Massey. The padding rules of SHA-3 candidates are also similar to Merkle's padding rule.

In [130], Nandi described the key property that a padding scheme needs to possess in order for an iterated hash function to preserve collision resistance: suffix-freeness. A suffix-free padding scheme ensures, for any binary string s, that $s||\mathsf{pad}(X) \neq \mathsf{pad}(X')$ for all $X \neq X'$. It is easy to check that both the padding schemes of Merkle and Damgård are suffix-free, thus they do not violate Nandi's requirement. The following definition introduces the Merkle–Damgård iterative hash function construction MD defined by any (injective) suffix-free padding; unless stated otherwise, we set MD as the mode of operation used throughout the thesis.

Definition 2.2.4. MD is an iterative hash function $\mathsf{H}^{\mathsf{h}}_{\mathsf{MD}} : \mathcal{K} \times \mathcal{V} \times \{0,1\}^* \to \mathcal{V}$ with compression function $\mathsf{h} : \mathcal{K} \times \mathcal{V} \times \mathcal{M} \to \mathcal{V}$, g the identity function and pad a specific (injective) suffix-free padding (e.g., Merkle's or Damgård's padding rules).

The following result is the reformulation and generalization of the preservation result provided by Merkle [125] and Damgård [52].

Theorem 2.2.5. *Let* $\mathsf{H}^{\mathsf{h}}_{MD}$ *be the Merkle–Damgård (*MD*) iterated hash function defined by the compression function* $\mathsf{h} : \mathcal{K} \times \mathcal{V} \times \mathcal{M} \to \mathcal{V}$*. Then*

1. *If* h *is* (t', ϵ') coll*-secure, then* $\mathsf{H}^{\mathsf{h}}_{MD}$ *is* (t, ϵ) coll*-secure for* $t = t' - 2(\ell - 1)\, t_{\mathsf{h}, \mathcal{V} \times \mathcal{M}}$ *and* $\epsilon = \epsilon'$ *.*
2. *If* h *is* (t', ϵ') epre*-secure, then* $\mathsf{H}^{\mathsf{h}}_{MD}$ *is* (t, ϵ) epre*-secure for* $t = t' - \ell\, t_{\mathsf{h}, \mathcal{V} \times \mathcal{M}}$ *and* $\epsilon = \epsilon'$ *,*

where ℓ *is the maximum length of the messages, in blocks (say* $\ell = 2^{64}$*) and* $t_{\mathsf{h}, \mathcal{V} \times \mathcal{M}}$ *is defined as* $t_{\mathsf{H}, \mathcal{X}}$*.*

Proof. Let $K \xleftarrow{\$} \mathcal{K}$ be given and $\mathsf{pad}_{\mathsf{sf}}$ be any (injective) suffix-free padding. Assume, without loss of generality, that $\mathcal{M} = \{0,1\}^m$ for a positive integer m. First, we prove the collision resistance preservation claim; a simpler proof for (everywhere) preimage resistance preservation follows later. The proof follows by constructing, given a collision-finding adversary $\mathscr{A}_{\mathsf{H}}$ for the iterated hash function H, a collision-finding adversary $\mathscr{A}_{\mathsf{h}}$ for the compression function with the same advantage. Let $\mathscr{A}_{\mathsf{h}}$ run $\mathscr{A}_{\mathsf{H}}$ and collect a colliding pair of messages (X, X'); i.e., $\mathsf{H}_K(X) = \mathsf{H}_K(X')$ and $X \neq X'$. Let

$$\mathsf{pad}_{\mathsf{sf}}(X) = \mathsf{M} = (M_1, \ldots, M_\ell) \quad \text{and} \quad \mathsf{pad}_{\mathsf{sf}}(X') = \mathsf{M}' = (M'_1, \ldots, M'_{\ell'}) \,.$$

Therefore, $\mathsf{H}_K(\mathsf{X}) = v_\ell = v'_{\ell'} = \mathsf{H}_K(\mathsf{X}')$. Now assume, without loss of generality, that $\ell \leq \ell'$ and that

$$\mathsf{M}' = \mathsf{M}'_{\mathsf{L}} || \mathsf{M}'_{\mathsf{R}} \quad \text{for} \quad \mathsf{M}'_{\mathsf{L}} \in \left(\{0,1\}^m\right)^{\ell' - \ell} \quad \text{and} \quad \mathsf{M}'_{\mathsf{R}} \in \left(\{0,1\}^m\right)^{\ell} \,.$$

By suffix-freeness, we are ensured that $\mathsf{M} \neq \mathsf{M}'_{\mathsf{R}}$. Indeed, otherwise we would obtain

$$\mathsf{M}' = \mathsf{pad}_{\mathsf{sf}}(X') = \mathsf{M}'_{\mathsf{L}} || \mathsf{M}'_{\mathsf{R}} \;=\; \mathsf{M}'_{\mathsf{L}} || \mathsf{pad}_{\mathsf{sf}}(X) = \mathsf{M}'_{\mathsf{L}} || \mathsf{M} \,,$$

which would violate suffix-freeness. Moreover, the collision for H results in

$$\begin{aligned} v_\ell &= \mathsf{h}(K, \mathsf{h}(K, \ldots \mathsf{h}(K, \mathsf{h}(K, v_0, M_1), M_2) \ldots, M_{\ell-1}), M_\ell) \\ &= \mathsf{h}\left(K, \mathsf{h}(K, \ldots \mathsf{h}(K, \mathsf{h}(K, v'_{\ell'-\ell}, \mathsf{M}'_{\mathsf{R},1}), \mathsf{M}'_{\mathsf{R},2}) \ldots, \mathsf{M}'_{\mathsf{R},\ell-1}), \mathsf{M}'_{\mathsf{R},\ell}\right) = v'_{\ell'} \,, \end{aligned}$$

where $\mathsf{M}'_{\mathsf{R}} = \left(\mathsf{M}'_{\mathsf{R},1} || \ldots || \mathsf{M}'_{\mathsf{R},\ell}\right)$ and $\mathsf{M}'_{\mathsf{R},\mathsf{j}} \in \{0,1\}^m$ for $j = 1, \ldots, \ell$. We claim that, given $v_\ell = v'_{\ell'}$ and

$$(v_0, M_1, \ldots, M_\ell) \neq \left(v'_{\ell'-\ell}, \mathsf{M}'_{\mathsf{R},1}, \ldots, \mathsf{M}'_{\mathsf{R},\ell}\right) \,,$$

there must exist an i for $0 \leq i \leq \ell - 1$ such that

$$(v_i, M_{i+1}) \neq \left(v'_{\ell'-\ell+i}, \mathsf{M}'_{\mathsf{R},\mathsf{i}+1}\right) \quad \text{but} \quad \mathsf{h}(K, v_i, M_{i+1}) = \mathsf{h}\left(K, v'_{\ell'-\ell+i}, \mathsf{M}'_{\mathsf{R},\mathsf{i}+1}\right) \,,$$

which is nothing but the definition of a collision for the compression function. To prove our claim, we simply assume the contrary: For all $1 \leq i < \ell$,

$$v_{i+1} = v'_{\ell'-\ell+i+1} \;\Rightarrow\; (v_i, M_{i+1}) = \left(v'_{\ell'-\ell+i}, \mathsf{M}'_{\mathsf{R},\mathsf{i}}\right) \,.$$

Due to the definition of $\mathsf{H}^{\mathsf{h}}_{\mathsf{MD}}$, however, this assumption would imply

$$(v_0, M_1, \ldots, M_\ell) = \left(v'_{\ell'-\ell}, \mathsf{M}'_{\mathsf{R},1}, \ldots, \mathsf{M}'_{\mathsf{R},\ell}\right) \,,$$

which is a contradiction. Hence, a collision for the compression function h is found and we obtain

that $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(\mathscr{A}_{\mathsf{h}}) = \mathsf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(\mathscr{A}_{\mathsf{H}})$. Moreover, $\mathscr{A}_{\mathsf{h}}$ halts the collision in time at most $t + 2(\ell' - 1)\, t_{\mathsf{h},\mathcal{V}\times\mathcal{M}}$.

Now, given an everywhere preimage-finding adversary $\mathscr{A}_{\mathsf{H}}$ for the hash function H, consider the following adversary $\mathscr{A}_{\mathsf{h}}$ against the compression function. As in the collision resistance preservation proof, $\mathscr{A}_{\mathsf{h}}$ runs $\mathscr{A}_{\mathsf{H}}$ and obtains a preimage X for the target value Y; i.e., $\mathsf{H}_K(X) = Y$. Using the same notation as above we have $\mathsf{H}_K(X) = Y = v_\ell$ and, looking at the last compression function evaluation, $\mathsf{h}(K, v_{\ell-1}, M_\ell) = v_\ell$ for $\mathsf{pad}_{\mathsf{sf}}(X) = \mathsf{M} = (M_1, \ldots, M_\ell)$. The adversary $\mathscr{A}_{\mathsf{H}}$ simply outputs $(v_{\ell-1}, M_\ell)$ as the preimage; note that its advantage is the same as $\mathscr{A}_{\mathsf{h}}$. In addition, the preimage is found in time $t + \ell\, t_{\mathsf{h},\mathcal{V}\times\mathcal{M}}$: t for running $\mathscr{A}_{\mathsf{h}}$ and the rest for calculating $v_{\ell-1}$ (from IV), as well as the target. □

We note that MD does not preserve the remaining five security notions (apre, pre, asec, esec, sec) mentioned previously [5, 15]: The case for esec is shown in [5, 15], whereas the remaining cases are shown in [5]. Finally we remark an alternative collision resistance result for an iterated hash function: If we assume that the padding function is only injective and the underlying compression function is both collision and everywhere preimage resistant, then the overall iterated hash function (again using the identity function as the output transformation g) becomes collision resistant [5].

2.2.2 Generic Cryptanalytic Methods Against Strengthened Merkle–Damgård

In this section, we briefly recall the generic cryptanalytic methods presented in the last few years against the strengthened Merkle–Damgård domain extension. The methods presented here are generic in the sense that, regardless of the choice of the compression function, they can be applied to any hash function constructed via sMD. It should be noted however that some of these methods are not directly related to the security notions discussed previously. Instead, they are used to argue the security of sMD against other properties that a hash function is expected to satisfy, e.g., indifferentiability (from a random oracle).

One of the important results in this direction is due to Joux [82]: He presents the so-called *multi-collision attack* to effectively find r distinct messages which hash to the very same value (r*-collision*). For a random function, finding an r*-collision* requires to hash roughly $2^{(r-1)n/r}$ messages for a digest size of n-bits [82]. Joux shows, however, that for sMD finding an r*-collision* is almost as hard as finding a 2*-collision* (up to a logarithmic factor), regardless of the value of r. This result was one of the strongest indications that sMD is not *very* secure. Joux uses his technique to show that concatenating the results of several hash functions (at least one of which is constructed via sMD) to get a higher length hash function does not yield a secure construction, which had been an open problem then.

In [5], Andreeva et al. show that sMD does not preserve pre and sec, i.e., we do not necessarily obtain a preimage and second-preimage resistant hash function even though the underlying compression function is perfect in term of these notions. Yet, they do not manage to find an attack to illustrate the concrete weakness. The first indication in this direction comes with the works of Dean [53] and Kelsey and Schneier [88]. In [88], it is shown that finding a second-preimage for a long message requires much less than 2^n work (for an n-bit digest); it is the first concrete attack showing such a strong result against sMD.

5. A proof for such a statement follows the same principle as in the proof of Theorem 2.2.5; given a collision-finding adversary for the hash function, we construct the one for the underlying compression function. The only difference is that we end up either with a collision for the compression function or a preimage for the initialization vector IV along the way.

Another work in this direction is the work of Kelsey and Kohno [87] who show yet a different weakness of sMD. They call their method *herding attack*, utilized to violate the notion named *chosen-target forced-prefix* (CTFP) preimage resistance. Informally, CTFP preimage resistance ensures the infeasibility of producing a message M that, when combined with the forced-prefix P, hashes to a chosen-target Z (i.e., for any fixed key K, $\mathsf{H}_K(\mathrm{P}||\mathrm{M}) = Z$). Ideally, we would expect a good hash function to have CTFP resistance close to preimage resistance; yet for sMD it seems that CTFP preimage resistance is somewhere in between (optimal) collision resistance and preimage resistance.

Finally, we note the so-called *extension attacks* against sMD, a method that affects MAC (message authentication codes) applications of hash functions. A common way to construct a MAC by using a cryptographic hash function is to hash $K'||X$ where K' is a secret-key (hence different from K used in keyed-hash functions) and X is the message to be authenticated. It is almost immediate that this construction is secure when the underlying hash function is modeled as a random oracle [49]. However, when sMD is used, we can *extend* the message X with any arbitrary message X' and obtain the MAC of $X||X'$ without knowing the secret-key K' [49]. The extension attack is considered to be the most important weakness of sMD. Indeed, NIST also explicitly demands (from the designers) resistance against extension attacks from the very beginning of the SHA-3 competition. Several techniques to thwart extension attacks are proposed in [49].

2.2.3 Other Iterated Domain Extenders

In order to thwart some of the above mentioned weaknesses, several alternative methods to sMD are proposed in the literature. Here we list some of the iterated constructions, without giving the details [6] (for the exact specifications and security preservations, we refer to corresponding references, as well as [3, 5]): prefix-free Merkle–Damgård [49] (pfMD), enveloped Merkle–Damgård [12] (EMD), Merkle–Damgård with permutation [77] (MDP), linear hash [15] (LH), linear XOR [15] (XLH), Shoup's hash [184] (SH), random oracle XOR [3] (ROX), the backwards chaining mode [6] (BCM), HAIFA [21], Dither hash [168] (DH), Randomized hash [72] (RMX), Double-pipe hash [111] (LDP) and Zipper hash [110] (ZH).

Table 2.1 (from [3]) contains a summary of preservation results (see e.g., [3, 12, 56, 108] for the preservation results for other security notions). We note that the only domain extension algorithm that preserves all the seven security notions given earlier is ROX [3]. The major difference of ROX from the other schemes is that it requires certain calls to two random oracles (one for padding, one for the iteration) during execution, which can be seen as a disadvantage for practical applications although it provides strong security guarantees.

2.3 Compression Functions Based on Blockciphers

Ever since the initial design of cryptographic hash functions, one of the most popular and best-known methods to create a hash function revolves around blockciphers. The idea of blockcipher-based hashing dates back to Rabin [162] who suggested to hash a message using DES (inside a compression function with a mode of operation similar to MD, without imposing suffix-free padding). Other blockcipher-based designs soon followed, the best-known ones being Davies–Meyer [122], Matyas–

6. We note however that Definition 2.2.1 does not apply to all of the listed designs.

Scheme	coll	epre	pre	apre	sec	esec	asec
BCM	✓	×	×	×	✓	×	×
DH	✓	×	×	×	×	×	×
EMD	✓	×	×	×	×	×	×
HAIFA	✓	×	×	×	×	×	×
LDP	✓	×	×	×	×	×	×
LH	×	✓	×	×	×	×	×
MD	✓	✓	×	×	×	×	×
MDP	✓	×	×	×	×	×	×
pfMD	×	×	×	×	×	×	×
RMX	✓	×	×	×	×	×	×
ROX	✓	✓	✓	✓	✓	✓	✓
SH	✓	✓	×	×	×	✓	×
XLH	✓	✓	×	×	×	✓	×
ZH	?	×	×	×	×	×	×

Table 2.1 – A summary of the preservation results of some iterated hash functions. The symbol "✓" shows that the preservation holds, whereas "×" means that it is not preserved. We also show by "×" if the corresponding definition is not applicable, e.g., the construction does not accept keys. The symbol "?" demonstrates that no result is known.

Meyer–Oseas [177] and Miyaguchi–Preneel [159].

The recipe for blockcipher-based hashing is simple and clear: First a compression function is created using the blockcipher and subsequently a full-blown hash function is obtained using a domain extender, e.g., sMD. The compression functions of the MD-family, SHA-family and even some of the SHA-3 candidates can, either explicitly or implicitly, be regarded as blockcipher-based designs.

The incentive for blockcipher-based hashing is intuitive and a bit historical: Blockciphers have been *the* primitives since the early years of modern cryptography and a blockcipher operating on n-bit blocks and a κ-bit key can already be regarded as a compression function from $n+\kappa \to n$ bits, compressing κ bits at a time. Moreover, using a blockcipher to create a hash function is particularly effective for resource constrained environments as we only need to implement one blockcipher to obtain an encryption scheme and a hash function simultaneously.

In this section, following the recent advances in blockcipher-based hashing, we formally define the blockcipher-based hash and compression functions and state related security properties. Moreover, we briefly recapitulate the latest results in this field along with the remaining open problems. As usual, we begin first with the definitions.

2.3.1 The Model

Let $\mathrm{Block}(\kappa, n)$ denote the set of all blockciphers having a κ-bit key and operating on n-bit blocks. In other words, $\mathrm{Block}(\kappa, n)$ [7] is the set of all maps $E\colon \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$, such that for any key $K \in \{0,1\}^\kappa$, $E_K(\cdot) = E(K,\cdot)$ is a permutation on the set $\{0,1\}^n$. For a blockcipher E, we denote its

7. For each κ-bit key, there exist $2^n!$ permutations as the block-length is n; in addition, as we can assign any permutation to a given key, there are $(2^n!)^{2^\kappa}$ possible blockciphers.

inverse by D, so for all $K \in \{0,1\}^\kappa$ and $X \in \{0,1\}^n$ we have that $D_K(E_K(X)) = X$. When $E \xleftarrow{\$} \text{Block}(\kappa, n)$ is chosen uniformly at random, we call it an *ideal cipher*. By choosing a random element of $\text{Block}(\kappa, n)$, denoted $E \xleftarrow{\$} \text{Block}(\kappa, n)$, we mean that for each $K \in \{0,1\}^\kappa$ we choose a permutation $E_K(\cdot)$ over $\{0,1\}^n$ uniformly at random.

We refer to a compression function h^E (or more generally, $\mathsf{h}^{E^1,\ldots,E^r}$ when there are r independently sampled blockciphers $E^1, \ldots, E^r \xleftarrow{\$} \text{Block}(\kappa, n)$) as *blockcipher-based* if one or multiple blockciphers are utilized to instantiate it. More formally, a blockcipher-based compression function is a map $\mathsf{h}^E : \text{Block}(\kappa, n) \times \mathcal{V} \times \mathcal{M} \to \mathcal{V}$ where access to an element $E \in \text{Block}(\kappa, n)$ is black box. That is, h^E must be given by a program that, given $M \in \mathcal{M}$ and $V \in \mathcal{V}$, computes $\mathsf{h}^E(V, M)$ using an E oracle. Similarly, the iterated hash function $\mathsf{H}^{\mathsf{h}^E}$, constructed via h^E, is called blockcipher-based and can be defined in a similar fashion. Note that here the random choice of the blockcipher takes the role of the key in our standard keyed-hash function definition; thus we again concern ourselves with a family of hash functions. Throughout the thesis, we often omit the superscript E if the context suffices to determine that the compression and hash functions in question are blockcipher-based.

Security Notions

Similar to Section 2.1, we define an adversary as an algorithm; this time, however, adversaries have oracle access to E and D (or more generally, to $E^1, \ldots, E^r, D^1, \ldots, D^r$ if there are r different blockciphers). In the following, we write oracles as superscripts. As security notions, we limit ourselves again to (everywhere) preimage resistance and collision resistance. In the following, we concern ourselves with information-theoretic adversaries only, meaning that the sole resource of interest for the adversary is the number of queries made to their oracles. Without loss of generality, adversaries are assumed not to repeat queries [8] nor to query an oracle outside of its specified domain. These adversaries are considered computationally unbounded and this model has been called the *Shannon model* or, as we prefer, the *ideal cipher model*.

The ideal cipher model has been used extensively in the literature [8, 26, 27, 76, 107, 191, 192] for arguing security of blockcipher-based hash and compression functions. The reason for using such a model is a natural one [28]: The common assumption for blockciphers, in the standard model, is that they are pseudo-random permutations [9]. However in some scenarios, this assumption is not sufficient to get useful security results [187]. Therefore, an alternative method is required to model blockciphers in order to achieve provability.

We remark that, as in the random oracle model, there is an "uninstantiability" result by Black [28] who provides a blockcipher-based hash function that is secure in the ideal cipher model yet it turns out to be insecure when instantiated with a real-world blockcipher. Here again the example given by Black is quite unnatural; nevertheless, it proves that the security provided in the ideal cipher model does not necessarily imply security in practical applications. As pointed out in [28], "no scheme has thus far been proven secure in the random oracle (or ideal cipher model) and then broken once instantiated, unless this was the goal from the start."

8. By this we mean that an adversary never makes a query to the blockcipher E that results in a query-response triple (K, X, Y) (with $E_K(X) = Y$) which already appears in the query list. For instance an adversary is not allowed to make a decryption query (K, Y) if a corresponding encryption query (K, X), which satisfies $E_K(X) = Y$, has already been made.

9. Informally, this assumption asserts that a blockcipher that operates on n-bit blocks (under a secret randomly-chosen key) is computationally indistinguishable from a randomly-chosen n-bit permutation.

For (everywhere) preimage and collision resistance, adversarial success can be determined based on the query history $\mathcal{Q}$ only, which we formalize using the yield set (Definition 2.3.1). We partition $\mathcal{Q}$ in $\mathcal{Q}[1]\ldots\mathcal{Q}[r]$ depending on which of the r primitives was called and, although technically elements of $\mathcal{Q}$ are triples (primitive, query, response), we assume that the context suffices to determine which of the r primitives was used (occasionally, we abuse notation by writing $X \in \mathcal{Q}$).

Definition 2.3.1 (Adversarial yield)**.** Let $\mathsf{h} : \mathrm{Block}(\kappa, n) \times \mathcal{V} \times \mathcal{M} \to \mathcal{V}$ be a blockcipher-based compression function and let $\mathcal{Q}$ be a set of queries with answers to the underlying blockcipher (E and D), then the yield set $\mathrm{yieldset}^{\mathsf{h}}(\mathcal{Q})$ is the set of all triples (V, M, Z) such that $Z = \mathsf{h}^E(V, M)$ and all queries to evaluate the compression function at (V, M) are in $\mathcal{Q}$. We refer to the cardinality of $\mathrm{yieldset}^{\mathsf{h}}(\mathcal{Q})$ as the *yield* and denote it by $\mathrm{yield}^{\mathsf{h}}(\mathcal{Q})$. Additionally, we define

$$\mathrm{yield}^{\mathsf{h}}(q) = \max_{\mathcal{Q}} \left\{ \mathrm{yield}^{\mathsf{h}}(\mathcal{Q}) \right\} ,$$

where $|\mathcal{Q}| \leq q$, q being the maximum number of queries made to E plus D. (Note that as $\mathcal{Q}$ incorporates the primitives' answers, the maximum implicitly includes a maximization over the choice of the underlying blockcipher.) We similarly define all of these notions (i.e., $\mathrm{yieldset}^{\mathsf{H}}(\mathcal{Q})$ and $\mathrm{yield}^{\mathsf{H}}(\mathcal{Q})$) for the hash function H (and omit the superscript when it is clear from the context) and generalize them for multiple blockcipher calls (for r calls we have $|\mathcal{Q}| \leq rq$).

Now, we move on to the formal definitions of the security notions we consider. In the following, without loss of generality, all the definitions are given assuming that the underlying primitive is a single blockcipher. Generalization to the multiple-call versions and other primitives is immediate.

Definition 2.3.2. Let $\mathsf{h} : \mathrm{Block}(\kappa, n) \times \mathcal{V} \times \mathcal{M} \to \mathcal{V}$ and $\mathsf{H}^{\mathsf{h}}_{\mathsf{MD}} : \mathrm{Block}(\kappa, n) \times \{0,1\}^* \to \mathcal{V}$ be a blockcipher-based compression and hash function constructed via MD, respectively. For a given $\mathcal{Q}$ and $Z \in \mathcal{V}$, define

$$\mathrm{coll}^{\mathsf{h}}(\mathcal{Q}) \equiv \exists_{Z,(V,M)\neq(V',M')} (V, M, Z), (V', M', Z) \in \mathrm{yieldset}^{\mathsf{h}}(\mathcal{Q}) \quad \text{and}$$

$$\mathrm{coll}^{\mathsf{H}}(\mathcal{Q}) \equiv \exists_{Z,M\neq M'} (M, Z), (M', Z) \in \mathrm{yieldset}^{\mathsf{H}}(\mathcal{Q}) .$$

The collision-finding advantage of an adversary $\mathscr{A}$ is defined as

$$\mathrm{Adv}^{\mathrm{coll}}_{\mathsf{X}}(\mathscr{A}) = \Pr\left[E \xleftarrow{\$} \mathrm{Block}(\kappa, n), \mathcal{Q} \xleftarrow{\$} \mathscr{A}^{E,D} : \mathrm{coll}^{\mathsf{X}}(\mathcal{Q}) \right] ,$$

where $\mathsf{X} \in \{\mathsf{h}, \mathsf{H}\}$. Similarly, define $\mathrm{Adv}^{\mathrm{coll}}_{\mathsf{X}}(q) = \max_{\mathscr{A}} \left\{ \mathrm{Adv}^{\mathrm{coll}}_{\mathsf{X}}(\mathscr{A}) \right\}$, where the maximum is taken over all adversaries $\mathscr{A}$ making at most q queries in total (to E plus D).

Definition 2.3.3. Let $\mathsf{h} : \mathrm{Block}(\kappa, n) \times \mathcal{V} \times \mathcal{M} \to \mathcal{V}$ and $\mathsf{H}^{\mathsf{h}}_{\mathsf{MD}} : \mathrm{Block}(\kappa, n) \times \{0,1\}^* \to \mathcal{V}$ be a blockcipher-based compression and hash function constructed via MD, respectively. For a given $\mathcal{Q}$ and $Z \in \mathcal{V}$, define

$$\mathrm{epre}^{\mathsf{h}}_Z(\mathcal{Q}) \equiv \exists_{(V',M')} (V', M', Z) \in \mathrm{yieldset}^{\mathsf{h}}(\mathcal{Q}) \quad \text{and} \quad \mathrm{epre}^{\mathsf{H}}_Z(\mathcal{Q}) \equiv \exists_{M'} (M', Z) \in \mathrm{yieldset}^{\mathsf{H}}(\mathcal{Q}) .$$

The everywhere preimage-finding advantage of an adversary A is defined as

$$\mathrm{Adv}^{\mathrm{epre}}_{\mathsf{X}}(\mathscr{A}) = \max_{Z\in\mathcal{V}} \left\{ \Pr\left[E \xleftarrow{\$} \mathrm{Block}(\kappa, n), \mathcal{Q} \xleftarrow{\$} \mathscr{A}^{E,D} : \mathrm{epre}^{\mathsf{X}}_Z(\mathcal{Q}) \right] \right\} ,$$

where $\mathsf{X} \in \{\mathsf{h}, \mathsf{H}\}$. Similarly, define $\mathrm{Adv}^{\mathrm{epre}}_{\mathsf{X}}(q) = \max_{\mathscr{A}} \left\{ \mathrm{Adv}^{\mathrm{epre}}_{\mathsf{X}}(\mathscr{A}) \right\}$, where the maximum is taken over all adversaries $\mathscr{A}$ making at most q queries in total (to E plus D).

The following result, the proof of which can be given as in Theorem 2.2.5, is a reformulation of the preservation result we have already discussed for MD.

Theorem 2.3.4. *Let* $\mathsf{h} : \mathrm{Block}(\kappa, n) \times \mathcal{V} \times \mathcal{M} \to \mathcal{V}$ *and* $\mathsf{H}^{\mathsf{h}}_{MD} : \mathrm{Block}(\kappa, n) \times \{0,1\}^* \to \mathcal{V}$ *be a blockcipher-based compression and hash function constructed via MD, respectively. Then*

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(q) \leq \mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q) \quad \textit{and} \quad \mathsf{Adv}^{\mathsf{epre}}_{\mathsf{H}}(q) \leq \mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q)\,.$$

As previously noted, we remark that under the assumption that the padding function is only injective, the collision and everywhere resistance of the underlying compression function is sufficient to obtain a collision resistant iterative hash function.

2.3.2 Generalization to Other Primitives and Stam's Conjecture

We note that blockciphers are not the only primitives for constructing a hash function. Indeed, there exist two more well-known primitives used in the literature for designing compression functions: public [10] random functions (PuRFs) and random permutations. For positive integers c and n, let $\mathrm{Func}(cn, n)$ [11] denote the set of all maps $\{0,1\}^{cn} \to \{0,1\}^n$ and let $f \xleftarrow{\$} \mathrm{Func}(cn, n)$ denote that f is sampled uniformly from $\mathrm{Func}(cn, n)$. We call f a *public random function* (PuRF) and we refer to a compression function making oracle calls to f as *PuRF-based*. Similarly, let $\mathrm{Perm}(n)$ denote the set of all permutations of $\{0,1\}^n$ and let $\pi \xleftarrow{\$} \mathrm{Perm}(n)$ denote that π is sampled uniformly at random from all elements in $\mathrm{Perm}(n)$. We call the compression function making oracle calls to π *permutation-based* (as in the blockcipher-based case, an adversary has oracle access to both π and π^{-1}). Throughout the thesis, whenever we refer to a primitive-based compression function, we mean that the primitive is either a blockcipher or a PuRF or a permutation. A generalization of the security definitions (Definitions 2.3.1, 2.3.2 and 2.3.3) given previously to the arbitrary primitive-based setting is immediate.

Now assume we are constructing a primitive-based compression function. It would be desirable to know how many calls to the underlying primitives we need to make for which level of security, in particular collision resistance, as it would give some sort of a confidence in the design. In Crypto'08, Stam [190] studies this security/efficiency trade-off problem (see also the work of Rogaway and Steinberger [173, 174]) by using the yield as a main tool and arrives at the following conjecture. This conjecture is widely accepted and used to deduce certain possibility/impossibility results.

Conjecture 2.3.5 (Stam's Conjecture)**.** *Let* $\mathsf{h} : \{0,1\}^{m+s} \to \{0,1\}^s$ *be a primitive-based compression function that compresses* m *bits at a time and makes* r *calls to the underlying primitives. Assume that the primitives accept* cn*-bit inputs where* c *is an arbitrary positive real number (*$c \in \{1,2,3\}$ *is of particular practical interest); the output length is not important. Then collisions for* h *can be found (with probability at least one half) with*

$$q = \lceil 2^{(crn-m)/(r+1)} \rceil + 1$$

queries to each of the primitives.

10. Public in the sense that all parties including the adversary has access to these primitives.

11. We note that the output length does not necessarily need to divide the input length; because conventional cryptographic primitives obey this rule, we consider this case in this thesis.

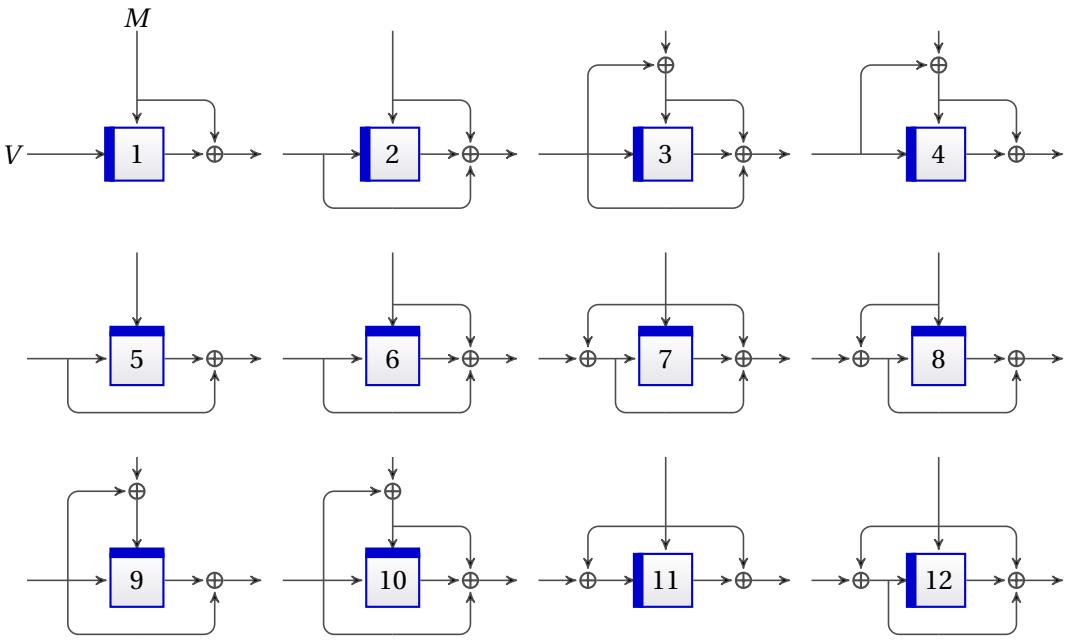

Figure 2.3 – PGV compression functions $\mathsf{h}^E(V,M)$ with almost optimal collision and (everywhere) preimage resistance. The box illustrates the blockcipher and all the wires carry n bits. The key of the blockcipher is fed into the dark side of the box, whereas the plaintext comes as an input to the other edge shown with an incoming arc. The state value V enters from the left and the message block M is fed from the top (see the PGV compression function 1 for the example of the illustration). The output exits from the right. Matyas–Meyer–Oseas, Miyaguchi–Preneel and Davies–Meyer compression functions are shown in 1, 2 and 5, respectively.

Part of the proof of the conjecture (i.e., the case where $(2m-n(r-1))/(r+1) \geq \log_2(17)$), up to a small constant, is given in [193].

2.3.3 Single-Call Compression Functions

Of the many methods, the most common approach for creating a blockcipher-based compression function is to use a single call to a blockcipher operating on n-bit blocks with κ-bit keys, which results in a compression function from $n+\kappa$ bits to n bits. The previously mentioned methods of Rabin, Davies–Meyer, Matyas–Meyer–Oseas and Miyaguchi–Preneel can be given in this class. This category is often referred to as rate-1 single-block-length. Here the rate is a measure of efficiency, defined as the ratio of the number of message blocks being hashed over the number of blockcipher calls (thus higher rates are more efficient). Single-block-length, however, indicates that the output length of the compression function is the same as the block-length of the underlying blockcipher.

One of the first systematic analyses of single-call single-block-length compression functions was done by Preneel, Govaerts and Vandewalle [160], henceforth "PGV", who considered the compression functions h^E of the form, for $E \in \mathrm{Block}(n,n)$, $\mathsf{h}^E(V,M) = E_K(X) \oplus t$, where $K, X, t \in \{\bar{c}, V, M, M \oplus V\}$ for an arbitrary constant $\bar{c} \in \{0,1\}^n$. Their analysis was attack oriented: They found efficient preimage- and collision-finding algorithms for both the compression function and the hash function obtained

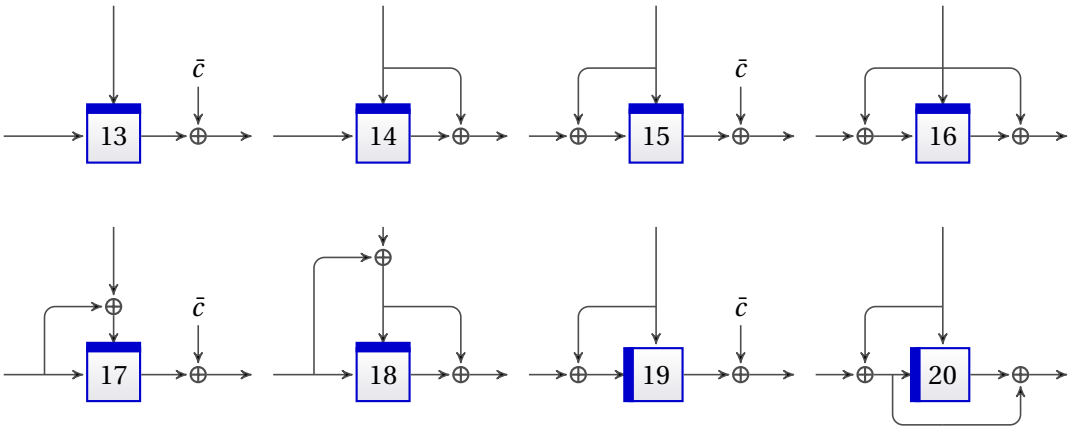

Figure 2.4 – Eight PGV compression functions that turn out to be almost optimally collision resistant when used via MD. Notation as in Figure 2.3 and $\bar{c}$ is an arbitrary constant in $\{0,1\}^n$. Rabin's scheme (with $\bar{c} = 0$) is shown in 13.

via sMD. Consequently, 12 schemes (see Figure 2.3 for an illustration [12] of the 12 constructions) out of possible 64 were deemed optimally collision and preimage resistant. Later, Black, Rogaway and Shrimpton [26] (and afterwards Duo and Li [58] and Stam [191]) proved that it is indeed the case. Furthermore, Black, Rogaway and Shrimpton were able to show that eight more constructions (see Figure 2.4) turn out to be optimally collision resistant when iterated (with an injective padding), even though collisions and preimages are trivial to find for their compression functions. Preimage resistance of these eight schemes are suboptimal (see Theorem 2.3.7 for the concrete bounds).

Here, we follow a more generalized approach, suggested initially by Stam [191] (see also [27]) to analyze the single-call blockcipher-based compression functions (also capturing the analysis done by Black, Rogaway and Shrimpton). We first define the single-call compression functions (see Figure 2.5).

Definition 2.3.6. Let $E \in \mathrm{Block}(\kappa, n)$. We call a blockcipher-based compression function $\mathsf{h}^E : \{0,1\}^s \times \{0,1\}^m \to \{0,1\}^s$ single-call if, given input (V, M), the output is computed as follows:

1. Prepare key and plaintext: $(K, X) \leftarrow C^{\mathsf{pre}}(V, M)$;
2. Make the call: $Y \leftarrow E_K(X)$;
3. Output the digest: $Z \leftarrow C^{\mathsf{post}}(V, M, Y)$,

where we call $C^{\mathsf{pre}} : \{0,1\}^s \times \{0,1\}^m \to \{0,1\}^\kappa \times \{0,1\}^n$ a preprocessing function, whereas $C^{\mathsf{post}} : \{0,1\}^s \times \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^s$ is called a postprocessing function. In particular, if $s = n$, then we call h^E single-call single-block-length (SBL). If $s = 2n$, h^E is called single-call double-block-length (DBL).

Note that PGV compression functions are a special case of single-call blockcipher-based compression functions for $s = \kappa = n$. The following theorem (also capturing the result of Black, Rogaway and Shrimpton on PGV schemes) is proved in [27, 191]. In Theorem 2.3.7, we also introduce the so-called Type-I and Type-II single-call SBL compression functions [27, 191]. Each type is defined as a set of conditions on the pre and postprocessing functions (and an additional function that is defined via C^{pre} and C^{post}). To illustrate, PGV compression functions shown in Figure 2.3 belong to Type-I,

12. The figure is inspired by the one given in [27].

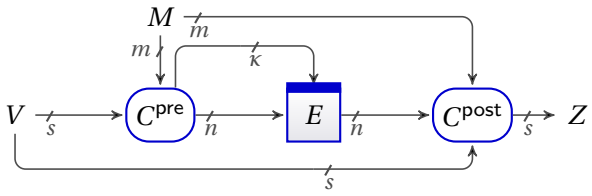

Figure 2.5 – Single-call blockcipher-based compression functions (see Definition 2.3.6).

whereas those illustrated in Figure 2.4 are of Type-II. Theorem 2.3.7 states concrete upper bounds of both types for collision and everywhere preimage advantages defined in Definitions 2.3.2 and 2.3.3, respectively.

Theorem 2.3.7. *Let* h^E *be a single-call blockcipher-based compression function with* $n = s$ *and* $\kappa = m$. *Define* $C^{\mathsf{aux}}(K,X,Y) = C^{\mathsf{post}}(C^{-\mathsf{pre}}(K,X),Y)$ *to be an auxiliary postprocessing function. If (Type-I single-call SBL compression functions, see Figure 2.3)*

1. *the preprocessing function* C^{pre} *is bijective,*
2. *for all* V,M, *the postprocessing function* $C^{\mathsf{post}}(V,M,\cdot)$ *is bijective,*
3. *for all* K,Y, *the auxiliary postprocessing function* $C^{\mathsf{aux}}(K,\cdot,Y)$ *is bijective,*

then

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q) \le \frac{q(q+1)}{2^n} \quad \textit{and} \quad \mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q) \le \frac{q}{2^{n-1}}$$

for any $q \ge 1$. *If, however, (Type-II single-call compression functions, see Figure 2.4)* 1., 2. *hold (but not* 3.*) together with*

3'. $C^{-\mathsf{pre}}(K,\cdot)$, *restricted to* V, *is bijective for all* K,

then for any $q > 1$,

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(q) \le \frac{q(q+1)}{2^n} \quad \textit{and} \quad \mathsf{Adv}^{\mathsf{epre}}_{\mathsf{H}}(q) \le \frac{q(q+1)}{2^n} .$$

Hence, Theorem 2.3.7 provides sufficient conditions for C^{pre} and C^{post} to get (almost) optimal collision resistance for the compression and (the iterated) hash function using a single-call single-block-length blockcipher-based compression function. Moreover, for preimage resistance, optimality can be achieved using Type-I single-call blockcipher-based compression functions.

Stam [191] (see also [27]) derives the same result as Black, Rogaway and Shrimpton for PGV compression functions [13] using simple counting arguments along with Theorem 2.3.7; consequently, he significantly simplifies the analysis done by Black, Rogaway and Shrimpton. Moreover, he provides similar sufficient conditions for single-call blockcipher-based compression functions of the following form:

1. *Chopped compression functions*: These single-call blockcipher-based compression functions correspond to the case where $s < n$ and $m + \kappa = n + s$.
2. *Overloaded compression functions*: These correspond to the case where $s + m > n + \kappa$; i.e., when the preprocessing function C^{pre} is compressing.

13. To illustrate, consider the Davies–Meyer compression function depicted in Figure 2.3 with number 5. The preprocessing function is defined by $C^{\mathsf{pre}}(V,M) = (M,V) = (K,X)$, whereas the postprocessing is given by $C^{\mathsf{post}}(V,M,Y) = V \oplus Y$. We can also express C^{aux} as $C^{\mathsf{aux}}(K,X,Y) = X \oplus Y$. Note that the Davies–Meyer is a Type-I single-call SBL compression function defined in Theorem 2.3.7 and the corresponding statements hold.

3. *Supercharged compression functions*: This class corresponds to the case where the postprocessing function C^{post} expands: $s+m=n+\kappa$ and $s \geq n$. One of the nicest outcomes of this class is single-call double-block-length compression function constructions (see Section 2.3.4 for more on this).

For the exact security bounds for these classes of compression functions we refer to [191].

2.3.4 Double-Block-Length Compression Functions

Single-block-length blockcipher-based compression functions, i.e., the constructions whose output length matches with the block-length of the blockcipher, historically face one major problem: Existing blockciphers seldom have sufficiently large block-lengths and in order to meet the basic security requirements (e.g., that of NIST for SHA-3) we would need a blockcipher operating on more than 256 bits. This rules out most existing blockciphers, including AES which operates on 128 bits blocks only.

To counter this discrepancy in required security levels, so-called double-block-length (DBL) compression functions (and corresponding hash functions) were introduced: These are compression functions with $2n$-bit output and are based on a blockcipher with only n-bit blocks. Thus, we can hope to achieve, for instance, collision resistance up to roughly 2^n blockcipher evaluations. DBL compression functions can also be useful for wide-pipe iterative hash functions [111].

DBL hash functions come in various guises, depending on the number of blockcipher calls made per compression function and the (bit)length of the key (to the blockcipher). The three most important variants are (for a blockcipher that operates on n-bit blocks):

1. single-call to a $2n$-bit key blockcipher;
2. double-call to a $2n$-bit key blockcipher; and
3. double-call to an n-bit key blockcipher.

Stam's supercharged compression functions take care of the first group above; yet for the remaining cases we need to extend the framework suggested by Stam. Note however that here we consider the constructions that make parallel calls to the underlying blockciphers; it is sufficient for our purposes and actually more interesting from a practical point of view due to the potential increase in efficiency.

Definition 2.3.8. Let $E^1, E^2 \in \mathrm{Block}(\kappa, n)$. We call a blockcipher-based compression function (see Figure 2.6) $\mathsf{h}^{E^1,E^2} : \{0,1\}^s \times \{0,1\}^m \to \{0,1\}^s$ double-call if, given input (V, M), the output is computed as follows:

1. Prepare key and plaintext: $(k_1, x_1) \leftarrow C_1^{\mathsf{pre}}(M,V), (k_2, x_2) \leftarrow C_2^{\mathsf{pre}}(M,V)$;
2. Make the calls: $y_1 \leftarrow E^1_{k_1}(x_1), y_2 \leftarrow E^2_{k_2}(x_2)$;
3. Output the digest: $Z \leftarrow C^{\mathsf{post}}(M,V,y_1,y_2)$,

where we call $C^{\mathsf{pre}} = (C_1^{\mathsf{pre}}, C_2^{\mathsf{pre}})$ for $C_1^{\mathsf{pre}}, C_2^{\mathsf{pre}} : \{0,1\}^s \times \{0,1\}^m \to \{0,1\}^\kappa \times \{0,1\}^n$ a preprocessing function and $C^{\mathsf{post}} : \{0,1\}^s \times \{0,1\}^m \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^s$ a postprocessing function. In particular, if $s = 2n$, then we call h^{E^1,E^2} a double-call DBL blockcipher-based.

Here we note that the underlying blockciphers do not necessarily have to be different; i.e., it is allowed that $E^1 = E^2$.

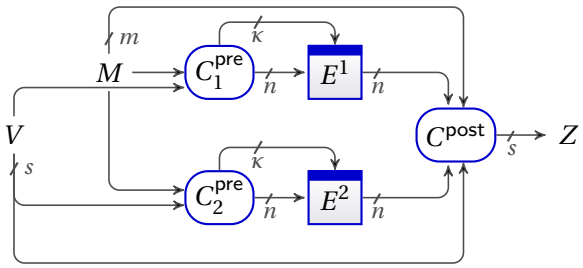

Figure 2.6 – Double-call blockcipher-based compression functions (see Definition 2.3.8).

Single-Call DBL Hash Functions from Double-Key Blockciphers

Here we consider the possibility of building a DBL blockcipher-based compression function with single-call to blockcipher with $2n$-bit key and n-bit blocks; i.e., single-call blockcipher-based compression functions with $s = \kappa = 2n$ and $m = n$ (Definition 2.3.6). Promising results are previously given by Lucks [112], who gives a scheme secure only in the iteration, and Stam [191], who gives a general framework for security of the (supercharged) compression functions. However, his framework does not include the earlier work by Lucks; an extension to Stam's framework for secure hash functions (that use weak supercharged single-call DBL compression functions) is given in [150]. In [191] Stam also gives a concrete proposal (also known as quadratic-polynomial-based design: QPB-DBL) for the DBL case with almost optimal collision resistance.

Although the efficiency of blockcipher-based compression functions is usually measured by the rate, this metric does not always give accurate efficiency estimates. In particular, both Lucks' and Stam's constructions require two $\mathbb{F}_{2^n}$ finite field multiplications. In practical situations, these field multiplications might actually cost more than a blockcipher call. For this reason, single-call DBL hash functions are not widely accepted in practice and DBL constructions, with more than one call to the underlying blockcipher, attracted much more attention.

Double-Call DBL Hash Functions from Double-Key Blockciphers

These constructions correspond to the case where $s = \kappa = 2n$ and $m = n$ (Definition 2.3.8). The two classic hash functions of this type are Tandem Davies–Meyer (Tandem-DM) and Abreast Davies–Meyer (Abreast-DM) [102]. Both schemes employ a single blockcipher called twice, where for Tandem-DM the calls are made in sequence (hence not captured by Definition 2.3.8) and for Abreast-DM they are made in parallel (see Figure 4.4 on page 79 for an illustration). Although both are widely believed to be (close to) optimally collision resistant, for a long time no security proof of this was known for either construction. Only recently Lee et al. [107] give a proof of collision resistance up to (almost) the birthday bound for Tandem-DM; a proof for the collision resistance up to (almost) the birthday bound for Abreast-DM is given in [64, 104]. For both schemes proving preimage resistance beyond the birthday bound had been an open problem until the recent work of Armknecht et al. [8].

There are some other DBL hash functions based on two calls to a double-key blockcipher that enjoy collision resistance up to the birthday bound. Hirose [75] proposes several conditions to achieve optimal security for the linear instances of pre and postprocessing functions for rate-1/2 in this category. In a way, these schemes can be regarded for this class to be what the PGV schemes are

for the classic, single-block-length setting. In his constructions, two independent blockciphers are called, which is sometimes considered as a disadvantage. Later, taking inspiration from earlier work by Nandi [128], Hirose [76] presents a scheme with optimal collision resistance using only one blockcipher (see Figure 4.5 on page 80 for the illustration).

Satoh et al. [178] mount several collision and preimage attacks on the compression functions of double-call hash functions from double-key blockciphers where only one blockcipher is used. They also provide a set of conditions for the case where the message length is equal to the digest length, which may be optimally collision resistant; he then left its investigation for future work. This work has been pursued further by Hattori et al. [73]. They first investigate a case uncovered by the analysis of Satoh et al., then show that the compression functions, which were not attacked by Satoh et al., are at most as secure as those of single-block-length hash functions.

Other DBL Hash Functions

There are two classic DBL blockcipher-based compression functions with $\kappa = m = n$ (Definition 2.3.8), namely MDC-2 and MDC-4 [37], where MDC-2 makes two calls and MDC-4 four calls to compress a single message block [14]. The compression function of MDC-2 is known to be collision resistant only up to $2^{n/2}$ queries (thus offering no improvement over single-block-length constructions), but the iterated hash function was widely believed to provide near-optimal collision resistance. Steinberger [192] gives the first non-trivial result in the ideal cipher model for the collision resistance of MDC-2 by showing that an adversary asking fewer than $2^{3n/5}$ queries has only a negligible chance of finding a collision in the iteration, e.g., when used with sMD. Knudsen et al. [90] give a collision attack of time-complexity approximately $2^n/n$, nibbling a logarithmic factor off the optimal birthday bound (they also give a preimage attack on MDC-2 of time-complexity 2^n). Thus, there remains a large gap with respect to the collision resistance security of MDC-2. For a more complicated (yet less efficient) scheme such as MDC-4 even less is known.

Recently, an alternative DBL construction to MDC-2—called MJH—was proposed by Lee and Stam [105], which is inspired by the compression function of JH [206] (one of the SHA-3 finalists). As in the case of MDC-2, MJH uses two calls to a single-key blockcipher and the compression function itself does not provide security beyond what a single-block-length compression function can offer. Yet, in the iteration, Lee and Stam show that we need (asymptotically in n) $\Omega(2^{2n/3-\log n})$ queries to the underlying blockcipher to find a collision with high probability. Lee and Stam's work is the first design on this class of compression functions that beat the collision resistance lower bound of MDC-2. Constructing a double-call DBL compression function with $\kappa = m = n$ (Definition 2.3.8) with collision resistance bound beyond $2^{n/2}$ queries has been an open problem; similarly, for the same class, beating the lower bound of MJH in the iteration is still open.

Another classic work on the double-call DBL hash functions with $s = m = 2n$ and $\kappa = n$ (Definition 2.3.8) is by Knudsen et al. [94] who show that for all $\mathbb{F}_2$-'block'-linear schemes collisions can be found (with high probability) in time roughly $2^{3n/4}$ and preimages in time close to 2^n. Knudsen et al.'s work is a clear indication for an impossibility result of creating a highly efficient double-call DBL hash function.

14. MDC-2 and MDC-4 are originally designed for use with DES; the generalization allowing key-size equal to the block size is immediate.

Finally, we note the work of Peyrin et al. [154] who mimic the approach of PGV done on single-call single-block-length blockcipher-based compression functions to the case for DBL compression functions with varying blockcipher calls [15]. Consequently, they determine, under a very general attack-based approach (only considering time-complexity upper bounds), necessary conditions to have a secure compression function. They conclude that we need at least five calls to the (single- or double-key) blockciphers in order to thwart some generic attacks and they propose some constructions that satisfy their criteria. In a later work, Peyrin and Seurin [180] follow a more proof-centric approach and derive (suboptimal) query-complexity lower and upper bounds for their proposals (improving the bounds given in [180] and showing the security in the iteration are still open problems).

2.3.5 Extensions: Knudsen–Preneel Compression Functions

In Section 2.3.4, we saw how to achieve blockcipher-based compression functions that output more than the block-length of the underlying blockcipher. The output expansion is typically achieved by calling the blockcipher(s) multiple times and combining the resulting blockcipher outputs in some clever way. In almost all these classic examples, the standard approach in designing wider-output compression functions has been to fix a target output size (and often a number of blockcipher calls) and then to try to build a compression function that is optimally collision-resistant for that size.

In three papers [91–93], Knudsen and Preneel adopt a different approach, specifically to let the output size and (relatedly) the number of blockcipher calls vary as needed in order to guarantee a particular security target. Given r independent ideal compression functions (or PuRFs) $f^1,\ldots,f^r$, each mapping cn bits to n bits, they create a new 'bigger' compression function outputting rn bits. The functions $f^1,\ldots,f^r$ are run in parallel, and each of their inputs is some linear combination of the blocks of message and chaining variable that are to be processed. The rn-bit output of their construction is the concatenation of the outputs of these parallel calls. Knudsen and Preneel also propose to instantiate the underlying ideal compression functions with a blockcipher run in the Davies–Meyer mode and to iterate the compression function to obtain a blockcipher-based hash function (when iterated, we could compress the final state to some desired length depending on the security target).

The elegance of the KP construction is in *how* the inputs to $f^1,\ldots,f^r$ are computed: A generator matrix of an $[r,k,d]$ linear error-correcting code over $\mathbb{F}_{2^c}$ determines how the ck input blocks of the 'big' compression function are XORed together to form the inputs to the underlying r functions. (In a generalization they consider the f^i as mapping from bcn' to bn' bits instead and use a code over $\mathbb{F}_{2^{bc}}$). Under a broad—but *prima facie* not unreasonable—assumption related to the complexity of finding collisions in parallel compression functions, Knudsen and Preneel show that any attack needs time at least $2^{(d-1)n/2}$ to find a collision in their construction. For preimage resistance, Knudsen and Preneel *conjecture* that attacks will require at least $2^{(d-1)n}$ time. They also give preimage- and collision-finding attacks that are mostly independent of the minimum distance.

Watanabe [205] was the first to point out a collision attack demonstrating that the *proven* collision-resistance *lower* bound given by Knudsen and Preneel is incorrect whenever $2k > r$ and $d > 3$. For a

15. They actually consider multi-call, multi-block-length compression functions based on smaller PuRFs $2n \to n$ and $3n \to n$ bits rather than blockciphers.

code with minimum distance $d = 3$, he matches the Knudsen–Preneel 2^n collision-resistance lower bound, but does not violate it. Yet this is the first indication that something is amiss with the claim by Knudsen and Preneel.

2.4 Contributions

The contributions of this thesis are three-fold: Firstly, in Chapter 4, following the generalization proposed by Stam for single-call blockcipher-based compression functions, we provide a novel framework for the security analysis of a special class of DBL blockcipher-based hash functions (i.e., with $\kappa = 2n$ and $m = n$). Secondly, in Chapter 5 we introduce a new double-call DBL compression function, inspired by incidence geometry, with improved collision resistance bound for the same class (i.e., with $\kappa = m = n$ as MDC-2 and MJH). Finally, in Chapter 6, we (re)analyze the preimage and collision resistance of the Knudsen–Preneel compression functions, a large class of compression functions briefly introduced in Section 2.3.5. Below, we describe these contributions in more detail.

Another Look at Double-Block-Length Hash Functions

In Chapter 4, we propose a novel framework for the security analysis for double-call DBL blockcipher-based compression and hash functions (Definition 2.3.8) by extending the recent generalization presented by Stam at FSE '09 for single-call constructions (Definition 2.3.6). We focus on the designs with parameters $\kappa = s = 2n$ and $m = n$, and restrict ourselves to two parallel calls (both identical and two independently sampled blockciphers). We analyze the kind of pre and postprocessing functions that are sufficient to obtain close-to-optimal collision resistance, either in the compression function or in the iteration.

We group the schemes according to two important characteristics: Firstly, we distinguish between collision resistance in the compression function (Type-I) and collision resistance in the iteration (Type-II), as was done by Black et al. [26]. Secondly, for secure compression functions only, we differentiate between schemes where the two blockciphers E^1 and E^2 are distinct (and independently sampled) and schemes where only a single blockcipher is used, so $E^1 = E^2$.

We show that Type-I schemes enjoy near-optimal collision resistance in the ideal cipher model. For preimage resistance, we are only able to prove resistance up to the birthday bound. The recent work of Armknecht et al. [8] improves our result on preimage resistance to almost optimal bound. For Type-II schemes, the generalization from the classical single-call case is less straightforward. In fact, we can only prove slightly suboptimal collision resistance, although we do believe that our conditions suffice for optimal collision resistance.

Regarding preimage resistance, we show that preimages can be found for Type-II schemes with $\mathcal{O}(2^{3n/2})$ queries (for a digest size of $2n$) by providing a concrete preimage-finding attack. Finally, we investigate the ramifications of our framework for $\mathbb{F}_2$-'block'-linear instances of C^{pre} and C^{post} for Type-I and Type-II schemes. The contributions detailed in Chapter 4 are published in [150], at the proceedings of the 12th IMA International Conference, Cryptography and Coding 2009.

A Compression Function Exploiting Discrete Geometry

In Chapter 5, we study the open question of constructing a double-call (PuRF-based) DBL compression function with $m = \kappa = n$ having collision resistance bound beyond $2^{n/2}$ queries. This is similar to MDC-2 or the recently proposed MJH. However, unlike these proposals, we show for any $\delta > 0$ and $q \leq 2^{2n(1-\delta)/3}$ that $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q) = o(1)$ (asymptotically in n). To the best of our knowledge, this is the first construction (of this type) attaining collision resistance lower bound beyond $2^{n/2}$ queries.

A key challenge for our design is to bound the yield(q); to this end, we use an innovative preprocessing function C^{pre} where any given valid input pair to the underlying ideal primitives corresponds to an incidence between a point and a line (i.e., the point lies on the line) in the affine plane $\mathbb{F}^2_{2^n}$. A classic result from discrete geometry—the Szemerédi–Trotter theorem over finite fields—is subsequently applied to bound the number of these incidences, and with it the yield. An appropriate postprocessing function, together with a careful (non-trivial) analysis, then formally gives us the remarkable collision-resistance lower bound. For the preimage resistance, we show for any $\delta > 0$ and $q \leq 2^{n(1-\delta)}$ that $\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q) = o(1)$ (asymptotically in n). This work [81] will appear at the proceedings of TCC (Theory of Cryptography Conference) 2012.

Attacking the Knudsen–Preneel Compression Functions

In Chapter 6, we (re)analyze the preimage and collision resistance of the Knudsen–Preneel compression functions in the setting of public random functions by mounting concrete attacks against them. Our attacks are based on two key observations. First, by using the right kind of queries it is possible to mount (non-adaptive) preimage or collision attacks of a surprisingly low query-complexity (optimal even in the case of preimage attacks). Second, by exploiting the *dual code* and the *dual code* of the *shortened code*, the subsequent problem of reconstructing a preimage, respectively a collision from the queries can be dealt with very efficiently.

Our new preimage attack consistently beats the one given by Knudsen and Preneel and demonstrates that the gap between Watanabe's collision attack and the actual preimage resistance is surprisingly small. Moreover, our new attack falsifies the Knudsen–Preneel (conjectured) preimage resistance security bound. Complementing our attack is a formal analysis of the query-complexity (both lower and upper bounds) of preimage-finding attacks. In this analysis we show that for many concrete codes the time-complexity of our attack is optimal.

For collision resistance, we also fold in the idea originating from Watanabe and we actually arrive at a family of attacks. This makes expressing the final time-complexity in closed form difficult; yet our best attack has a time-complexity strictly smaller than the block-size (thus beating Watanabe's attack) for all but two of the parameter sets suggested by Knudsen and Preneel.

Consequently, our new attacks falsify both (conjectured) preimage resistance and proven collision resistance security bound given by Knudsen and Preneel; and we conclude that, with the possible exception of two of the proposed parameter sets, the Knudsen–Preneel compression functions do not achieve the security level they were designed for. The proceeding versions [149, 151] (presented in Asiacrypt 2010 and FSE 2010) of our contributions are currently being (jointly) prepared for a submission to the Journal of Cryptology.

3 Setting the Stage

This chapter is dedicated to the introduction of the tools required in the subsequent chapters: First, in Section 3.1 we recall the mathematical basics and related notation that we will be using throughout. In Section 3.2 we introduce the generalization of the compression functions we considered in Chapter 2 to the multi-call multi-block-length setting, as well as the blockwise-linear constructions, the designs often used in practice for their simplicity. Finally, in Section 3.3 we discuss the techniques for the probabilistic analysis of (adaptive) adversaries, the tools we will be using very often.

3.1 Some Mathematical Basics

Linear Error-Correcting Codes

An $[r,k,d]_{2^e}$ linear error-correcting code $\mathscr{C}$ is the set of elements (codewords) in a k-dimensional subspace of $\mathbb{F}_{2^e}^r$ (for $r \geq k$), where the minimum distance d is defined as the minimum Hamming weight (taken over all non-zero codewords in $\mathscr{C}$). The dual code $[r, r-k, d^\perp]_{2^e}$ is the set of all elements in the $r-k$-dimensional subspace orthogonal to $\mathscr{C}$ (with respect to the usual inner product), and its minimum distance is denoted $d^\perp$. The Singleton bound puts a limit on the minimum distance: $d \leq r-k+1$. Codes matching the Singleton bound are called maximum distance separable (MDS). An important property of an MDS code is that its dual is MDS as well, so $d^\perp = k+1$.

An $[r,k,d]_{2^e}$ code $\mathscr{C}$ can be generated by a matrix $G \in \mathbb{F}_{2^e}^{k\times r}$, meaning that $\mathscr{C} = \{x \cdot G | x \in \mathbb{F}_{2^e}^k\}$ (using row vectors throughout). A generator matrix G is called systematic if and only if it has the form $G = [I_k|P]$ where I_k the identity matrix in $\mathbb{F}_{2^e}^{k\times k}$ and P is a matrix in $\mathbb{F}_{2^e}^{k\times(r-k)}$. Furthermore, G is the generator matrix of an MDS code if and only if any k columns are linearly independent.

For an index set $I \subseteq \{1,\ldots,r\}$ we define $G_I \in \mathbb{F}_{2^e}^{k\times|I|}$ as the restriction of G to those columns indexed by I. For a code and any index set $I \subseteq \{1,\ldots,r\}$, we want to define $\tilde{I} \subset \{1,\ldots,r\}$ such that $G_{\tilde{I}}$ is invertible (thus in particular $|\tilde{I}| = k$) and $\tilde{I} \subseteq I$ or $I \subseteq \tilde{I}$. For MDS codes, the existence of such an $\tilde{I}$ can be shown easily (and we can impose uniqueness e.g., by virtue of an ordering). For non-MDS codes there exist some I for which such an $\tilde{I}$ does not exist (for example the $I \subset \{1,\ldots,r\}$ for which $|I| = k$ but G_I is not invertible), however for any target cardinality it is possible to find an I (of that cardinality) that does have an $\tilde{I}$ (e.g., by first going through the systematic columns, i.e., the I_k part); we call such an I *admissible*.

A given $[r,k,d]_{2^e}$ code $\mathscr{C}$ can be *shortened* to obtain a new, derived code $\mathscr{C}'$. Let $i \in \{1,\dots,r\}$, then consider the set of all codewords in $\mathscr{C}$ that are 0 on position i. The new code $\mathscr{C}'$ consists of these codewords with position i dropped, however we sometimes 'quasi-shorten' and keep the superfluous zeros present (we always keep the original indexing). It is easy to see that $\mathscr{C}'$ is an $[r-1,k-1,d]_{2^e}$ code unless all codewords in $\mathscr{C}$ had a 0 on position i or $k=1$ (in the latter case the shortening might result in the trivial one-codeword code $\{0^{r-1}\}$). The shortening of an MDS code is an MDS code itself. By repeated applications we can shorten by any index set $I_0 \subset \{1,\dots,r\}$ for which $\theta = |I_0| < k$ to obtain a derived $[r-\theta,k-\theta,d]$ MDS code.

Binary Field Representation

There are multiple ways to represent any given (binary) finite field $\mathbb{F}_{2^e}$. One common approach is to construct the field by adjoining an appropriate root ω, that is $\mathbb{F}_{2^e} = \mathbb{F}_2(\omega)$. If we let f be the minimal polynomial of ω, we reach an alternative representation by considering $\mathbb{F}_2[x]/f(x)$. In this case, we know that f has degree e and there is a one-to-one correspondence between elements in $\mathbb{F}_{2^e}$ and polynomials in $\mathbb{F}_2[x]$ of degree smaller than e (being the unique representatives of the equivalence classes in $\mathbb{F}_2[x]/f(x)$). This allows us to view the field as an e-dimensional vector space $\mathbb{F}_2^e$ over $\mathbb{F}_2$, for instance by regarding $(1,x,\dots,x^{e-1})$ as a basis.

The mapping $\mathbb{F}_{2^e} \to \mathbb{F}_2[x]/f(x)$ is easily seen to be a group isomorphism and, more generally, let $\psi : \mathbb{F}_{2^e} \to \mathbb{F}_2^e$ be a group isomorphism that takes the field to the vector space. The multiplicative operation in the field consequently induces an action on the vector space (as it is a bijective transformation). The field paradigms imply that for any given element $g \in \mathbb{F}_2^e$, the corresponding map $\mathscr{L}_g$ (on the vector space) is linear: in order to see that let $\mathscr{L}_g : \mathbb{F}_2^e \to \mathbb{F}_2^e$ be the map defined by $\mathscr{L}_g(x) = \psi(gx)$. Then

$$\mathscr{L}_g(x+y) = \psi(g(x+y)) = \psi(gx+gy)) = \psi(gx) + \psi(gy) = \mathscr{L}_g(x) + \mathscr{L}_g(y)$$

and for any $a \in \mathbb{F}_2$, we obtain

$$\mathscr{L}_g(ax) = \psi(gax) = a\psi(gx) = a\mathscr{L}_g(x)$$

because a is simply equal to either zero or one. As $\mathscr{L}_g$ is linear, it can be represented by a matrix. In other words, there is a map $\varphi : \mathbb{F}_{2^e} \to \mathbb{F}_2^{e\times e}$ such that $\varphi(g)\psi(h) = \psi(gh)$ for all $g,h \in \mathbb{F}_{2^e}$. It is not too hard to see that this map is in fact an injective ring homomorphism:

1. As for all $f,g,h \in \mathbb{F}_{2^e}$ it holds that

   $$\varphi(fg)\psi(h) = \psi(fgh) = \varphi(f)\psi(gh) = \varphi(f)\varphi(g)\psi(h)\ ,$$

   we also have that $\varphi(fg) = \varphi(f)\varphi(g)$.

2. Similarly

   $$\varphi(f+g)\psi(h) = \psi((f+g)h) = \psi(fh+gh) = \psi(fh) + \psi(gh) = \varphi(f)\psi(h) + \varphi(g)\psi(h)\ ,$$

   which results in $\varphi(f+g)\psi(h) = (\varphi(f)+\varphi(g))\psi(h)$.

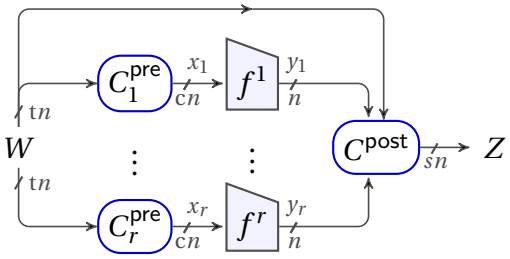

Figure 3.1 – General form of a tn-to-sn–bit single-layer PuRF-based compression function with feedforward based on r calls to underlying PuRFs with cn-bit inputs and n-bit outputs.

3.2 Multi-Call Multi-Block-Length Compression Functions

In Chapter 2, we formally introduced the single-block- and double-block-length primitive-based compression functions and discussed several related security properties. Here our goal is to make a further step and generalize our definitions to the multi-call and multi-block-length setting. We note that our security definitions (Definitions 2.3.2 and 2.3.3) naturally (and nicely) extend to the multi-call multi-block-length setting; in order to avoid repetitions we do not re-introduce these definitions here.

Throughout this thesis, we study *single-layer* compression functions. This means that the oracle calls are (or at least can be) made in parallel and the output of the compression function is computed based on the results of these calls, as well as the input itself (so we allow for feedforward). For simplicity, consider PuRF-based compression functions $\mathsf{h}^{f^1,\ldots,f^r}$.

Definition 3.2.1. Let $f^1,\ldots,f^r \in \mathrm{Func}(cn,n)$ and let t,s be two positive integers such that $t > s$. We call a PuRF-based compression function $\mathsf{h}^{f^1,\ldots,f^r} : \{0,1\}^{sn} \times \{0,1\}^{(t-s)n} \to \{0,1\}^{sn}$ multi-call multi-block-length if, given input (V,M) (for $V \in \{0,1\}^{sn}$ and $M \in \{0,1\}^{(t-s)n}$), the output is computed as follows:

1. Prepare the inputs: $x_i \leftarrow C_i^{\text{pre}}(V,M)$ for all $i \in \{1,\ldots,r\}$;
2. Make the calls: $y_i \leftarrow f^i(x_i)$ for all $i \in \{1,\ldots,r\}$;
3. Output the digest: $Z \leftarrow C^{\text{post}}(V,M,y_1,\ldots,y_r)$.

We call $C^{\text{pre}} = (C_1^{\text{pre}},\ldots,C_r^{\text{pre}})$ the preprocessing function where $C_i^{\text{pre}} \colon \{0,1\}^{tn} \to \{0,1\}^{cn}$ for $i = 1,\ldots,r$ and $C^{\text{post}} \colon \{0,1\}^{tn} \times (\{0,1\}^n)^r \to \{0,1\}^{sn}$ the postprocessing function.

Of note here is that we can interpret blockcipher-based constructions PuRF-based as well because a blockcipher operating on n-bit blocks and $(c-1)n$-bit key can be interpreted as a function from $\{0,1\}^{cn} \to \{0,1\}^n$. One difference for the blockcipher-based case is that we should assign two input blocks; one for the key and one for the plaintext of the underlying blockciphers. As another difference, sampling ideal ciphers is quite different from sampling random functions: By choosing a random element of $\mathrm{Block}(\kappa,n)$, denoted $E \xleftarrow{\$} \mathrm{Block}(\kappa,n)$, we mean that for each $K \in \{0,1\}^\kappa$ we choose a permutation $E_K(\cdot)$ over $\{0,1\}^n$ uniformly at random.

Most PuRF-based (and blockcipher-based) compression functions defined via Definition 3.2.1 are of a special type. Instead of arbitrary pre and postprocessing functions, we find only functions that are blockwise-linear (see Definition 3.2.2). The PGV compression and hash functions, as well

as many other concrete examples from the literature, are simple examples of this. An advantage of a blockwise approach is that it yields simple-looking hash functions whose security is easily seen to be determined by the block-size n. Linearity allows for relatively efficient implementation via bitwise exclusive-or of n-bit blocks. (In contrast, Lucks' and Stam's DBL schemes [112, 191] require full $\mathbb{F}_{2^n}$-arithmetic.) So, let us define formally a blockwise-linear single-layer PuRF-based compression function, an unwieldy name that we shorten to *blockwise-linear scheme*. For simplicity, we consider a single compression function input $W \in \{0,1\}^{tn}$ such that $W = (V, M)$ (for $V \in \{0,1\}^{sn}$ and $M \in \{0,1\}^{(t-s)n}$) and $\mathsf{h}^{f^1,\ldots,f^r}(V, M) = \mathsf{h}^{f^1,\ldots,f^r}(W) = Z$.

Definition 3.2.2 (Blockwise-linear scheme)**.** Let b, c, r, t, s be positive integers and let matrices $\mathsf{C}^{\mathsf{pre}} \in \mathbb{F}_2^{rcb\times tb}$, $\mathsf{C}^{\mathsf{post}} \in \mathbb{F}_2^{sb\times(t+r)b}$ be given. We define $\mathsf{h} = \mathsf{BL}^b(\mathsf{C}^{\mathsf{pre}}, \mathsf{C}^{\mathsf{post}})$ to be a family of single-layer PuRF-based (multi-call multi-block-length) compression functions $\mathsf{h}_n : \{0,1\}^{tn} \to \{0,1\}^{sn}$, for all positive integers n with $b|n$. Specifically, let $bn' = n$ and $f^1, \ldots, f^r \in \mathrm{Func}(cn, n)$. Then on input $W \in \{0,1\}^{tn}$ (interpreted as column vector), $\mathsf{h}_n^{f^1,\ldots,f^r}(W)$ computes the digest $Z \in \{0,1\}^{sn}$ as follows:

1. Compute $X \leftarrow (\mathsf{C}^{\mathsf{pre}} \otimes I_{n'}) \cdot W$;
2. For $X = (x_1||\ldots||x_r)$ compute $y_i = f^i(x_i)$;
3. For $Y = (y_1||\ldots||y_r)$ output $Z = (\mathsf{C}^{\mathsf{post}} \otimes I_{n'}) \cdot (W||Y)$,

where $(W||Y) \in \{0,1\}^{(t+r)n}$ is interpreted as column vector, $\otimes$ denotes the Kronecker product and $I_{n'}$ is the identity matrix in $\mathbb{F}_2^{n'\times n'}$.

In the definition above we identified $\{0,1\}^n$ with the vector space $\mathbb{F}_2^n$. The map corresponding to $(\mathsf{C}^{\mathsf{pre}} \otimes I_{n'})$ will occasionally be denoted by C^{pre}. It will be convenient for us to write the co-domain of C^{pre} as a direct sum, so we identify $\{0,1\}^{rcn}$ with $\bigoplus_{i=1}^r V_i$ where $V_i = \mathbb{F}_2^{cn}$ for $i = 1, \ldots, r$. If $x_1 \in V_1$ and $x_2 \in V_2$, then consequently $x_1 + x_2$ will be in $V_1 \oplus V_2$. This extends naturally to $L_1 + L_2$ when $L_1 \subset V_1, L_2 \subset V_2$. If we want to add 'normally' in $\mathbb{F}_2^{cn}$ we write $x_1 \oplus x_2$ which conveniently corresponds to exclusive or and the result will be in $\mathbb{F}_2^{cn}$ as expected.

Now our goal is to determine a lower bound for $\mathrm{yield}(q)$ of blockwise-linear schemes; Theorem 3.2.3 serves for this purpose. This result is important in that it provides a good indication for the security of the schemes that can be defined via Definition 3.2.2. The interpretation and the discussion of Theorem 3.2.3 will follow shortly.

Theorem 3.2.3. *Let* $\mathsf{h} = \mathsf{BL}^b(\mathsf{C}^{\mathsf{pre}}, \mathsf{C}^{\mathsf{post}})$ *be a blockwise-linear scheme with parameters* c, r, s, t*. Consider* h_n *with* b *dividing* n *(in particular,* $bn' = n$*). Then*

$$\mathrm{yield}^{\mathsf{h}}(q) \geq 2^{\lfloor \frac{\log q}{bc} \rfloor bt} \approx q^{t/c} ,$$

where q *is the total number of queries made to each primitive (hence the total number of queries is bounded by* rq*).*

Proof. Recall that t is the number of external n-bit input blocks and c the number of internal n-bit input blocks and that all n-bit blocks are subdivided into b n'-bit blocks. Set $n_q = \lfloor \log q/(bc) \rfloor$ and $\mathscr{X} = (0^{n'-n_q} \times \{0,1\}^{n_q})^{bc}$. For each of the bc internal input sub-blocks, set the leftmost $n' - n_q$ bits to zero and let the rest range over all possibilities in $\{0,1\}^{n_q}$. All combinations of the internal input sub-blocks are combined (under concatenation) to give $(2^{n_q})^{bc} \leq q$ distinct inputs for any particular

Figure 3.2 – An illustration of the bit-strings used in the proof of Theorem 3.2.3 is provided for a single PuRF input $X = (x_1, \ldots, x_e) \in \{0,1\}^{cn}$. The input block X is divided into $e = bc$ chunks each consisting of n' bits, in particular $x_i \in (0^{n'-n_q} \times \{0,1\}^{n_q})$. The shaded rectangles illustrate the (presumably) non-zero part of each chunk, whereas the rest shows the zero-bit blocks.

internal function. Query the f^i on these inputs (precisely corresponding to $\mathscr{X}$ defined above), for $i = 1, \ldots, r$.

Consider an external input W that consists of a concatenation of sub-blocks each with the first $n' - n_q$ bits set to zero. Then, due to linearity, $(\mathsf{C}^{\mathsf{pre}} \otimes I_{n'}) \cdot W$ will map to a collection of PuRF-inputs all corresponding to the queries formed above, and hence this W will contribute to the yield. As there are $2^{n_q bt}$ possible W that adhere to the format, we achieve the stated lower bound on the yield. The approximation follows by ignoring the floor and simplifying the resulting expression. Although this can lead to slight inaccuracies, for increasing n there will be more and more values of q for which the expression is precise (so when $q = 2^{\alpha n}$ for rational α the expression is precise infinitely often). □

Intuitively, when the yield for a tn-to-sn compression function gets close to $2^{sn/2}$, a collision is expected (birthday bound) and once it surpasses 2^{sn} a collision is guaranteed (pigeonhole) and a preimage expected. The bounds for permutation-based compression functions by Rogaway and Steinberger [174] are based on formalizing this intuition (under the assumption that the yield results in more or less uniform values). In the claims below we relate what the bound on the yield implies for preimage and collision resistance, assuming that the yield results in more or less uniform values. Note that the assumption certainly does not hold in general (hence 'presumably').

Claim 3.2.4 (Consequences for blockwise-linear schemes)**.** *Let* $\mathsf{h} = \mathsf{BL}^b(\mathsf{C}^{\mathsf{pre}}, \mathsf{C}^{\mathsf{post}})$ *be a blockwise-linear scheme with parameters* c, t, s, r*. Consider* h_n *with* b *dividing* n*.*

1. *If* $q \geq 2^{scn/t}$ *then* $\mathrm{yield}(q) \geq 2^{sn}$ *and a collision in* h_n *can be found with certainty (due to the pigeonhole principle); preimages can presumably be found with high probability.*
2. *If* $q \geq 2^{scn/(2t)}$ *then* $\mathrm{yield}(q) \geq 2^{sn/2}$ *and collisions in* h_n *can presumably be found with high probability (due to the birthday paradox).*

3.3 On the Probabilistic Analysis of Adaptive Adversaries

In this section we look at some of the existing techniques (and also introduce novel ones) to analyze the advantages of adaptive adversaries in achieving certain events. The collision- and preimage-finding experiments, which we consider in this thesis, can be seen as special examples of the events considered here. In addition, by looking at the common techniques for hash function security proofs, we develop a higher level intuition on how these proofs actually work; such a unified approach will provide a background for the analysis performed in the next chapters. We believe this contribution to be of independent theoretical interest [1].

1. We remark that in [81], we provide a similar framework using game-playing arguments.

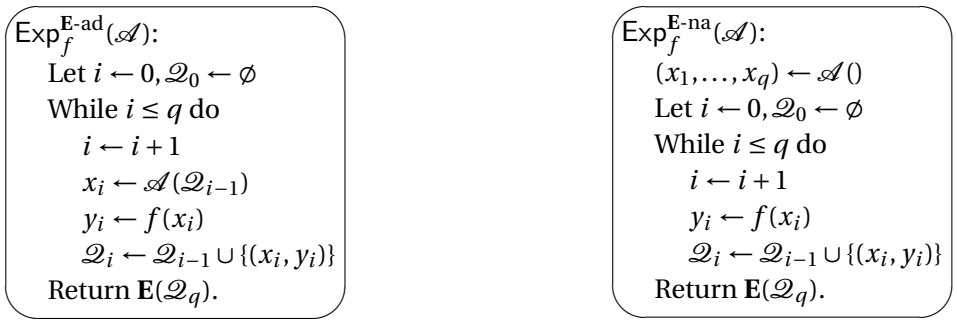

Figure 3.3 – Standard adaptive ($\mathsf{Exp}^{\mathbf{E}\text{-ad}}(\mathscr{A})$) and non-adaptive ($\mathsf{Exp}^{\mathbf{E}\text{-na}}(\mathscr{A})$) security games for (monotone) condition **E**. Here, $\mathscr{Q}_i$ denotes the list of queries and corresponding answers up to (and including) the i'th step.

Most of the security proofs in the literature for compression and hash functions rely on the same principle. Consider the game given in Figure 3.3, where the adversary has access to some underlying primitive f (source of randomness) and tries to set a predicate **E** that is defined for all collections of query-response pairs. We are primarily interested in monotone predicates **E**, that once set cannot be 'unset' by additional queries. A predicate **E** is monotone if for all $\mathscr{Q} \subseteq \mathscr{Q}'$, $\mathbf{E}(\mathscr{Q}) \Rightarrow \mathbf{E}(\mathscr{Q}')$. Additionally, we impose non-triviality of the predicate meaning that the predicate is not set from the outset (i.e., $\mathbf{E}(\emptyset) = \texttt{false}$). For collision resistance, one should read coll (Definition 2.3.2) for **E** and for preimage resistance epre_Z [2] (Definition 2.3.3). Note that coll and epre_Z are always monotone and that, for our analysis, both coll and epre_Z are non-trivial.

Bounding an advantage is then tantamount to bounding $\Pr[\mathbf{E}(\mathscr{Q})]$, where the probabilities are taken over the randomness of f and the coins of $\mathscr{A}$, if any. In the following, we show how we can analyze such events in a stepwise approach to determine useful upper bounds.

There is a crucial distinction between adaptive and non-adaptive adversaries (see Figure 3.3). The latter are required to commit to a fixed set of queries at the very beginning of the game. Consequently, maximizing over all q-query (non-adaptive) adversaries becomes equivalent to maximizing over all possible query sets of cardinality q. This considerably simplifies the proofs. For instance, when providing a proof in the ideal cipher model (using a union bound), for a non-adaptive adversary every response can be considered *fully* random, whereas for an adaptive adversary previous queries to the cipher might influence the outcome.

Maurer [119] (see also Pietrzak [155]) develops a methodology (using *random systems*) to equate adaptive and non-adaptive adversaries in certain cases. Although it is a possibility to use his results in our analyses, for many of our winning conditions—or those relevant to (blockcipher-based) hash functions in general—adaptive adversaries *do* have an advantage over non-adaptive adversaries making the application of adaptive vs. non-adaptive comparison problematic. Instead, we opt for a more direct approach, where we take our inspiration from both common hash-function security proofs and the techniques introduced by Maurer and Pietrzak. Henceforth, unless otherwise stated, we consider adaptive adversaries only (and consequently drop the "ad" suffix in naming experiments and advantages) and try to determine their advantages in corresponding experiments.

2. We remark that in epre_Z game (or in other security related games) the adversaries are also given the target digest Z as input (respectively additional inputs) although it is not explicitly mentioned in the simplified game given in Figure 3.3.

3.3.1 Preliminaries

Following the terminology and notation of [119, 155], we use P^f_A for probabilities (over f) $\Pr[A = a]$ where the subscript corresponds to the random variable A and the superscript f corresponds to the source of randomness. If it is clear from the context, we often omit the source of randomness and the specific values and simply use, e.g., $\mathsf{P}_{A|BC}$ to denote $\mathsf{P}[A = a|B = b \wedge C = c]$.

Random Systems Various cryptographic systems can be seen as random systems [119] that are modeled as the mathematical abstraction of interactive systems: An $(\mathscr{X},\mathscr{Y})$-random system takes the inputs $X_1, X_2, \ldots \in \mathscr{X}$ and for each input X_i it generates an output $Y_i \in \mathscr{Y}$ depending probabilistically on $X^i = (X_1, \ldots, X_i)$ and $Y^{i-1} = (Y_1, \ldots, Y_{i-1})$ (cf. the primitive f from Figure 3.3). Random systems have been used in the literature (see e.g., [119–121]) to unify, simplify, generalize, and in some cases strengthen security proofs. Definition 3.3.1 formally defines the term.

Definition 3.3.1 (Random system)**.** An $(\mathscr{X},\mathscr{Y})$-random system f is a (possibly infinite) sequence of conditional probability distributions $\mathsf{P}^f_{Y_i|X^iY^{i-1}}$ for $i \geq 1$; specifically, the distribution of the outputs Y_i conditioned on $X^i = x^i$ (i.e., the i'th query x_i and all previous queries $x^{i-1} = (x_1, \ldots, x_{i-1})$) and $Y^{i-1} = y^{i-1}$ (i.e., all previous outputs $y^{i-1} = (y_1, \ldots, y_{i-1})$). Define

$$\mathsf{P}^f_{Y^i|X^i} := \prod_{j=1}^{i} \mathsf{P}^f_{Y_j|X^jY^{j-1}},$$

where, for completeness,

$$\mathsf{P}^f_{Y_1|X^1Y^0} := \mathsf{P}^f_{Y_1|X^1} = \mathsf{P}^f_{Y_1|X_1}\,.$$

Two $(\mathscr{X},\mathscr{Y})$-random systems f and g are said to be equivalent (denoted by $f \equiv g$) if

$$\mathsf{P}^f_{Y_i|X^iY^{i-1}} = \mathsf{P}^g_{Y_i|X^iY^{i-1}}$$

for all $i \geq 1$.

Example 3.3.2 (Random system)**.** *Random functions and random permutations are special cases of random systems. If $(\mathscr{X},\mathscr{Y})$ is any pair of sets, a random function $\mathscr{X} \to \mathscr{Y}$ is a random variable whose values are functions $\mathscr{X} \to \mathscr{Y}$. For any finite set $\mathscr{X}$, a random permutation is a random variable taking values in the set of permutations of $\mathscr{X}$. A uniformly random function f is a random function with uniform distribution over all functions $\mathscr{X} \to \mathscr{Y}$. Using random systems, we have the following probabilities for a uniformly random function f:*

$$\mathsf{P}^f_{Y_i|X^iY^{i-1}} = \begin{cases} 1 & \text{if } x_i = x_j \text{ for some } j < i \text{ and } y_i = y_j\,, \\ 0 & \text{if } x_i = x_j \text{ for some } j < i \text{ and } y_i \neq y_j\,, \\ 1/|\mathscr{Y}| & \text{else.} \end{cases} \tag{3.1}$$

A uniformly random permutation is defined analogously; these two primitives are the two idealized objects that are used throughout the thesis.

One of the key concepts in cryptographic security definitions and proofs is the notion of indistinguishability. The first implicit reference to computational indistinguishability appears in the fundamental work of Blum and Micali [29] where one gives a precise mathematical meaning to

the idea that a random variable X *looking uniformly random* is a relative notion depending on a specified model of computation and a specified amount of computational resources. This naturally leads to a rigorous definition of a pseudo-random bit generator by requiring that the output bits cannot be distinguished from truly random bits by an efficient algorithm (distinguisher). More generally, computational indistinguishability allows for reducing the security analysis of a cryptographic primitive to analyzing an ideal primitive (e.g., a uniformly random function or permutation) that is trivially secure. Both pseudo-randomness and computational indistinguishability are thus notions from the complexity-theoretic setting.

Indistinguishability has a natural analogue in the information-theoretic setting as well (which is the main setting in this thesis) by imposing different restrictions on the distinguisher's resources: Here, we impose a constraint on the number of samples queried by the distinguisher and allow for unlimited computational power[3]. The simplest example in this setting is how easy it is to distinguish two random variables X and Y by a *distinguisher* that is allowed to query one sample from one of the two variables chosen uniformly at random. It is not hard to see that the success probability of the optimal distinguishing algorithm (the distinguisher's advantage) is simply the statistical distance of the two probability distributions for X and Y.

With the increasing number of sophisticated cryptographic schemes appearing in the literature (e.g., blockciphers, compression functions, message authentication codes), the level of complexity of proving indistinguishability becomes very complicated and technical; random systems provide a nice tool for that purpose. In order to distinguish two $(\mathcal{X},\mathcal{Y})$-random systems f and g, we model the distinguisher as a random system itself. A distinguisher interacts with random systems by making queries to either f or g and outputs a binary decision bit after a certain number of queries. Definition 3.3.3 formally introduces the concept of a distinguisher.

Definition 3.3.3 (Distinguisher)**.** An $(\mathcal{X},\mathcal{Y})$-distinguisher $\mathscr{A}$ is a $(\mathcal{Y},\mathcal{X})$-random system defined by (possibly infinite) sequence of conditional probability distributions $\mathsf{P}^{\mathscr{A}}_{X_i|Y^{i-1}X^{i-1}}$. That is, it is a $(\mathcal{Y},\mathcal{X})$-random system that is one query ahead. An $(\mathcal{X},\mathcal{Y})$-distinguisher $\mathscr{A}$ and an $(\mathcal{X}',\mathcal{Y}')$-random system f are said to be compatible if $\mathcal{X}'=\mathcal{X}$ and $\mathcal{Y}'=\mathcal{Y}$.

We model the interaction of a distinguisher with a random system via a random experiment (analogous to the one given in Figure 3.3) that is a sequence of conditional probability distributions denoted by $\mathsf{P}^{\mathscr{A}\Diamond f}_{X_iY_i|X^{i-1}Y^{i-1}}$ and defined simply as

$$\mathsf{P}^{\mathscr{A}\Diamond f}_{X_iY_i|X^{i-1}Y^{i-1}} := \mathsf{P}^{f}_{Y_i|X^iY^{i-1}}\mathsf{P}^{\mathscr{A}}_{X_i|X^{i-1}Y^{i-1}}\,.$$

Intuitively, this models the probabilities of the distinguisher choosing a given query x_i at the i'th step and the random system returning a given response y_i conditioned on the history. Moreover, we define

$$\mathsf{P}^{\mathscr{A}\Diamond f}_{X^iY^i} := \prod_{j=1}^{i}\mathsf{P}^{\mathscr{A}\Diamond f}_{X_jY_j|X^{j-1}Y^{j-1}} = \prod_{j=1}^{i}\mathsf{P}^{f}_{Y_j|X^jY^{j-1}}\mathsf{P}^{\mathscr{A}}_{X_j|X^{j-1}Y^{j-1}}\,.$$

Now we can formulate the monotone predicates in the context of random systems. The monotonicity

3. As far as we know, the explicit term "indistinguishability" in the information-theoretic setting first appeared in Maurer's work [119] where he used the term "quasi-random" as an information-theoretic analogue of "pseudo-random".

condition gives rise to a sequence of binary probabilities $\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|X^iY^i} \in \{0,1\}$ with the property that

$$\forall i \geq 1, \quad \mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|X^iY^i} = 1 \Rightarrow \mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_{i-1})|X^{i-1}Y^{i-1}} = 1\,.$$

Associated to a random system with a monotone predicate, we have the probability distributions $\mathsf{P}^f_{Y_i|X^iY^{i-1}}$ (data defining f), as well as the binary probabilities $\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|X^iY^i}$ (binary probabilities for $\mathbf{E}$). Using $\mathsf{P}^f_{Y_i|X^iY^{i-1}}$ and $\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|X^iY^i}$, we can now derive various other probabilities:

Event probabilities for E: These are the probabilities denoted by

$$\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}}\,.$$

Intuitively, it is the probability of setting the predicate $\neg\mathbf{E}(\mathscr{Q}_i)$ to true conditioned on the query/response history, as well as on the fact that $\neg\mathbf{E}(\mathscr{Q}_{i-1})$ holds. It is derived from $\mathsf{P}^f_{Y_i|X^iY^{i-1}}$ and $\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|X^iY^i}$ as follows:

$$\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} = \sum_{y_i} \mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|X^iY^i}\mathsf{P}^f_{Y_i|X^iY^{i-1}}. \tag{3.2}$$

Here, we can also derive the probability distributions $\mathsf{P}^f_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}}$ simply by using

$$\mathsf{P}^f_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} = 1 - \mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}}\,.$$

It is important to note that if the condition $\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}$ evaluates to `false` for all y_i for a given (x^i, y^{i-1}), this probability is set to zero; hence $\mathscr{A}$ provokes $\mathbf{E}(\mathscr{Q}_i)$ with probability one. It simply means that $\mathscr{A}$ provokes $\mathbf{E}(\mathscr{Q}_i)$ regardless of the response y_i.

A random system conditioned on E not failing: Consider the conditional probabilities

$$\mathsf{P}^f_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)X^iY^{i-1}}$$

that can be derived from Bayes' rule as follows:

$$\mathsf{P}^f_{Y_i|X^iY^{i-1}}\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|X^iY^i} = \mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)Y_i|X^iY^{i-1}} = \mathsf{P}^f_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)X^iY^{i-1}}\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|X^iY^{i-1}}\,, \tag{3.3}$$

where the middle term (which has not been defined yet) is a formal symbol for the corresponding probability. Assuming that (along with the monotonicity of $\mathbf{E}$ and that $\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_{i-1})|X^{i-1}Y^{i-1}} = 1$)

$$\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|X^iY^{i-1}} = \mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} \neq 0\,,$$

(which is not an invalid assumption given the predicates and constructions we are interested in) we can thus derive the conditional probabilities

$$\mathsf{P}^f_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)X^iY^{i-1}} = \frac{\mathsf{P}^f_{Y_i|X^iY^{i-1}}\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|X^iY^i}}{\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}}}\,. \tag{3.4}$$

Intuitively, this looks like a random system except that we have conditioned on the predicate $\neg\mathbf{E}(\mathcal{Q}_i)$. Note that this need not be a probability distribution: For instance, consider the example of a uniformly random function $f\colon \{0,1\}^n \to \{0,1\}^n$ and define the predicate $\neg\mathbf{E}(\mathcal{Q}_q)$ as the predicate that is set if there exists no collision between an input and an output. It might occur that $x_2 = y_1$ (as a clever adaptive adversary can choose x_2 adaptively based on previous responses) in which case $\neg\mathbf{E}(\mathcal{Q}_2)X^2Y^1$ will always evaluate to `false` and thus, the probability $\mathsf{P}^f_{Y_2|\neg\mathbf{E}(\mathcal{Q}_2)X^2Y^1} = 0$ for all y_2, so it will not represent a well-defined distribution on the variable Y_2. In cases when this degeneracy does not occur, we can consider the random system f conditioned on $\mathbf{E}$ (denoted $f|\mathbf{E}$) as a true random system g, i.e., $f|\mathbf{E} \equiv g$.

A random system with a predicate E: This is the random system that can be derived by

$$\mathsf{P}^f_{\neg\mathbf{E}(\mathcal{Q}_i)Y_i|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^iY^{i-1}} := \mathsf{P}^f_{Y_i|\neg\mathbf{E}(\mathcal{Q}_i)X^iY^{i-1}}\mathsf{P}^f_{\neg\mathbf{E}(\mathcal{Q}_i)|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^iY^{i-1}}\,, \tag{3.5}$$

where the probabilities on the right-hand side are assumed to be well-defined (see the discussion above). We also define

$$\mathsf{P}^f_{\neg\mathbf{E}(\mathcal{Q}_i)Y^i|X^i} := \prod_{j=1}^{i}\mathsf{P}^f_{\neg\mathbf{E}(\mathcal{Q}_j)Y_j|\neg\mathbf{E}(\mathcal{Q}_{j-1})X^jY^{j-1}}\,.$$

Moreover, we consider distinguishers (or adversaries) trying to provoke $\mathbf{E}$ again via a sequence of probability distributions. To indicate the link with $\mathbf{E}$, we denote these distributions by

$$\mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^{i-1}Y^{i-1}}\,.$$

As in the case of true random systems, this models the probability distribution of an adversary choosing the i'th query based on the previous responses and the predicate $\neg\mathbf{E}(\mathcal{Q}_{i-1})$ (meaning that the desired event $\mathbf{E}(\mathcal{Q}_{i-1})$ has not occurred after the $(i-1)$'st query/response pair).

Using this data, we can derive various probabilities and distributions by imposing Bayes' rule. We define the probabilities for the random experiment $\mathscr{A}\lozenge f$ by

$$\mathsf{P}^{\mathscr{A}\lozenge f}_{\neg\mathbf{E}(\mathcal{Q}_i)X_iY_i|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^{i-1}Y^{i-1}} := \mathsf{P}^f_{\neg\mathbf{E}(\mathcal{Q}_i)Y_i|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^iY^{i-1}}\mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^{i-1}Y^{i-1}}\,. \tag{3.6}$$

Intuitively, this models the probability of choosing a particular query, obtaining a particular response and the predicate $\neg\mathbf{E}(\mathcal{Q}_i)$ conditioned on the history and the predicate $\neg\mathbf{E}(\mathcal{Q}_{i-1})$. Finally, let

$$\mathsf{P}^{\mathscr{A}\lozenge f}_{\neg\mathbf{E}(\mathcal{Q}_i)X^iY^i} := \prod_{j=1}^{i}\mathsf{P}^{\mathscr{A}\lozenge f}_{\neg\mathbf{E}(\mathcal{Q}_j)X_jY_j|\neg\mathbf{E}(\mathcal{Q}_{j-1})X^{j-1}Y^{j-1}}\,. \tag{3.7}$$

Similarly, we define an expression for $\mathbf{E}(\mathcal{Q}_i)$. We are now ready to define the advantage of the distinguisher (adversary) $\mathscr{A}$ in provoking the desired event $\mathbf{E}(\mathcal{Q}_i)$:

Definition 3.3.4. Given $q > 0$ (i.e., the total number of queries made by an adversary $\mathscr{A}$ to f), define $\mathsf{Adv}^{\mathbf{E}}_f(\mathscr{A})$ to be the advantage of the (adaptive) distinguisher $\mathscr{A}$ in provoking the event $\mathbf{E}(\mathcal{Q}_q)$ in the random experiment $\mathscr{A}\lozenge f$. That is

$$\mathsf{Adv}_f^{\mathbf{E}}(\mathscr{A}) = \sum_{(x^q,y^q)\in\mathscr{X}^q\times\mathscr{Y}^q} \mathsf{P}^{\mathscr{A}\Diamond f}_{\mathbf{E}(\mathscr{Q}_q)X^qY^q} = \sum_{(x^q,y^q)\in\mathscr{X}^q\times\mathscr{Y}^q} \left(\prod_{i=1}^{q} \mathsf{P}^{f}_{\neg\mathbf{E}(\mathscr{Q}_i)Y_i|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^{i-1}Y^{i-1}} \right).$$

Furthermore, for all $q \geq 1$, define

$$\mathsf{Adv}_f^{\mathbf{E}}(q) := \max_{\mathscr{A}} \left\{ \mathsf{Adv}_f^{\mathbf{E}}(\mathscr{A}) \right\} .$$

3.3.2 Known Techniques

We start with the known techniques for the probabilistic analysis of adaptive adversaries; our goal is to provide a unified approach for describing these techniques, as well as to establish a background for the extensions we present subsequently. First, we study what we call the straightforward approach (Proposition 3.3.6): It is used in many papers [26,27,191] in the context of hash functions. Second, we deal with the case using an auxiliary monotone flag $\mathbf{F}(\mathscr{Q})$ (Proposition 3.3.7): Several recent examples that make use of this method can be found in the literature [107, 192]. In the subsequent chapters, we benefit from both methods.

The Straightforward Approach

The standard way of dealing with adaptive adversaries, as exemplified for instance by the security proofs [26,27,191] for the PGV compression functions [160], is the following: Suppose an adversary $\mathscr{A}$ makes q queries in total. These are necessarily made in sequence, so we denote by $\mathscr{Q}_i$ the set of query-responses (x_i is the i'th query made by $\mathscr{A}$ and y_i is the corresponding response) after i queries have been made (where $i \in \{0,\ldots,q\}$). The overall probability of setting $\mathbf{E}(\mathscr{Q}_q)$ can then be upper bounded by a sum (over i) of the maximum (over all adversaries) probability of winning on the i'th step, where these 'stepwise' probabilities are only taken over the choice of y_i. This makes the derivation of the overall bound relatively easy (even when taking into account the accompanying maximization). We first state an hypothesis that is commonly used in random system proofs; corresponding proposition follows the hypothesis.

Hypothesis 3.3.5. *Let f be an $(\mathscr{X},\mathscr{Y})$-random system and let $\mathbf{E}$ be a monotone predicate on f (hence we assume that the probabilities given in* (3.2) *are well defined). There exists an $(\mathscr{X},\mathscr{Y})$-random system g such that $f|\mathbf{E} \equiv g$, i.e., for all $i \geq 1$ and all $(x^i,y^i) \in \mathscr{X}^i \times \mathscr{Y}^i$,*

$$\mathsf{P}^{f}_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)X^iY^{i-1}} = \mathsf{P}^{g}_{Y_i|X^iY^{i-1}} .$$

Proposition 3.3.6. *Let f be an $(\mathscr{X},\mathscr{Y})$-random system and let $\mathbf{E}$ be a monotone predicate on f. Assuming that*

$$\sum_{i=1}^{q} \max_{(x^i,y^{i-1})} \left\{ \mathsf{P}^{f}_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} \right\} < 1 ,$$

for all $q \geq 1$ and all $(x^q,y^{q-1}) \in \mathscr{X}^q \times \mathscr{Y}^{q-1}$, we have

$$\mathsf{Adv}_f^{\mathbf{E}}(q) \leq \sum_{i=1}^{q} \max_{(x^i,y^{i-1})} \left\{ \mathsf{P}^{f}_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} \right\} .$$

Proof. We first show that Hypothesis 3.3.5 holds for all $i \le q$ and all (x^i, y^i). This is equivalent to at least one of the probabilities $\mathsf{P}^f_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)X^iY^{i-1}}$ being non-zero. Since assumption implies that

$$\mathsf{P}^f_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{1-1}} < 1 \quad \text{i.e.,} \quad \mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} > 0$$

and since

$$\mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} = \sum_{y_i} \mathsf{P}^f_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)X^iY^{i-1}} \mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|X^iY^{i-1}},$$

it follows that at least one summand is non-zero, i.e., there exists y_i for which

$$\mathsf{P}^f_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)X^iY^{i-1}} \neq 0$$

which implies the hypothesis. Thus, there exists an $(\mathscr{X},\mathscr{Y})$-random system g such that $f|\mathbf{E} \equiv g$, i.e., for all $i \ge 1$ and all $(x^i, y^i) \in \mathscr{X}^i \times \mathscr{Y}^i$,

$$\mathsf{P}^f_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)X^iY^{i-1}} = \mathsf{P}^g_{Y_i|X^iY^{i-1}}\,.$$

Instead of directly studying $\mathsf{Adv}^{\mathbf{E}}_f(q)$, we now look at an equivalent problem, where we minimize the advantage of any adversary $\mathscr{A}$ whose goal is to provoke $\neg\mathbf{E}(\mathscr{Q}_q)$. We write

$$\begin{aligned}
1 - \mathsf{Adv}^{\mathbf{E}}_f(q) &= \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} \mathsf{P}^{\mathscr{A}\Diamond f}_{\neg\mathbf{E}(\mathscr{Q}_q)X^qY^q} \right\} \\
&= \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} \left(\prod_{i=1}^{q} \mathsf{P}^f_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)X^iY^{i-1}} \mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^{i-1}Y^{i-1}} \right) \right\} \\
&= \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} \left(\prod_{i=1}^{q} \mathsf{P}^g_{Y_i|X^iY^{i-1}} \mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^{i-1}Y^{i-1}} \right) \right\} \\
&= \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} \left(\prod_{i=1}^{q} \mathsf{P}^g_{Y_i|X^iY^{i-1}} \left(1 - \mathsf{P}^f_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}}\right) \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^{i-1}Y^{i-1}} \right) \right\}.
\end{aligned}$$

Because of the inequality

$$\prod_{i=1}^{q} \left(1 - \mathsf{P}^f_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}}\right) \ge 1 - \sum_{i=1}^{q} \mathsf{P}^f_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} := B_q\,,$$

we arrive at

$$1 - \mathsf{Adv}^{\mathbf{E}}_f(q) \ge \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} B_q \left(\prod_{i=1}^{q} \mathsf{P}^g_{Y_i|X^iY^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^{i-1}Y^{i-1}} \right) \right\}.$$

Note that we can bound B_q in the following way:

$$B_q \ge 1 - \sum_{i=1}^{q} \max_{(x^i,y^{i-1})} \left\{ \mathsf{P}^f_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} \right\} := B'_q\,.$$

Since B'_q is independent of $\mathscr{A}$, we can pull it out the minimum and derive

$$
\begin{aligned}
1-\mathsf{Adv}_f^{\mathbf{E}}(q) &\geq B'_q \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} \left(\prod_{i=1}^{q} \mathsf{P}^{g}_{Y_i|X^iY^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^{i-1}Y^{i-1}} \right) \right\} \\
&= B'_q \min_{\mathscr{A}} \left\{ \sum_{(x^{q-1},y^{q-1})} \left(\prod_{i=1}^{q-1} \mathsf{P}^{g}_{Y_i|X^iY^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^{i-1}Y^{i-1}} \right) \right. \\
&\qquad \left. \sum_{x_q} \mathsf{P}^{\mathscr{A}}_{X_q|\neg\mathbf{E}(\mathscr{Q}_{q-1})X^{q-1}Y^{q-1}} \sum_{y_q} \mathsf{P}^{g}_{Y_q|X^qY^{q-1}} \right\} \\
&= B'_q \min_{\mathscr{A}} \left\{ \sum_{(x^{q-1},y^{q-1})} \left(\prod_{i=1}^{q-1} \mathsf{P}^{g}_{Y_i|X^iY^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^{i-1}Y^{i-1}} \right) \right\} = B'_q \,.
\end{aligned}
$$

The last line follows by (finite) induction and the facts that

$$
\mathsf{P}^{g}_{Y_i|X^iY^{i-1}} \quad \text{and} \quad \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^{i-1}Y^{i-1}}
$$

are conditional probability distributions. Hence, we conclude the proof. □

Using an Auxiliary Flag

Although easy, the standard approach has the disadvantage that for more complex constructions, the maximum probabilities can get too large. This is typically due to the maximum being attained only for relatively obscure values for $\mathscr{Q}_i$, values that themselves are extremely unlikely to occur. To weed out these unwanted cases, the analysis is often enhanced by splitting the monotone predicate into a set of auxiliary events. For some positive integer k, let $\mathbf{E_1},\ldots,\mathbf{E_k}$ be (monotone, non-trivial) predicates such that (for all $\mathscr{Q}$)

$$
\mathbf{E}(\mathscr{Q}) \Rightarrow \bigvee_{i=1}^{k} \mathbf{E_i}(\mathscr{Q}) \,,
$$

then a union bound implies

$$
\Pr[\mathbf{E}(\mathscr{Q})] \leq \sum_{i=1}^{k} \Pr[\mathbf{E_i}(\mathscr{Q})] \,.
$$

Several examples of proofs using auxiliary events can be found in the realm of double-block-length hash functions (e.g., [107, 192]).

The events $\mathbf{E_i}(\mathscr{Q})$ themselves are usually composed as the conjunction of a monotone event and a *negated* monotone event. In the simplest case, consider a second (non-trivial) monotone predicate $\mathbf{F}$. If we define $\mathbf{E_1} = \mathbf{E} \wedge \neg\mathbf{F}$ and $\mathbf{E_2} = \mathbf{F}$, then $\mathbf{E} \Rightarrow \mathbf{E_1} \vee \mathbf{E_2}$ is satisfied. To bound $\Pr[\mathbf{E_2}(\mathscr{Q})] = \Pr[\mathbf{F}(\mathscr{Q})]$ we can use Proposition 3.3.6; for $\Pr[\mathbf{E_1}(\mathscr{Q})] = \Pr[\mathbf{E}(\mathscr{Q}) \wedge \neg\mathbf{F}(\mathscr{Q})]$ Proposition 3.3.7 shows how the use of the predicate $\mathbf{F}$ effectively allows us to consider a more restricted class of $\mathscr{Q}$ (see Figure 3.4 for the corresponding game).

All this can be rigorously modeled using random systems as follows: Suppose that f is a random system with a monotone predicate $\mathbf{F}$ (here, $\mathbf{F}$ represents the flag event). Suppose further that f conditioned on $\mathbf{F}$, i.e., $f|\mathbf{F}$, is equivalent to another random system g (i.e., $f|\mathbf{F} \equiv g$). Now, we simply impose a monotone predicate $\mathbf{E}$ on g. Equivalently, we need to specify the corresponding probabilities and

distributions as we did previously. Suppose that we are given the following data:

1. Event probabilities for the random system g:

$$\mathsf{P}^{g}_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})X^iY^{i-1}} \quad \text{also denoted by} \quad \mathsf{P}^{f}_{\neg\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})\neg\mathbf{F}(\mathscr{Q}_i)X^iY^{i-1}}$$

   to indicate better what they are supposed to model.

2. Conditional probabilities:

$$\mathsf{P}^{g}_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)X^iY^{i-1}} = \mathsf{P}^{f}_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)\neg\mathbf{F}(\mathscr{Q}_i)X^iY^{i-1}} \,.$$

3. Distinguisher $\mathscr{A}$ relative to $\mathbf{E}$, namely, probability distributions:

$$\mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathscr{Q}_{i-1})\neg\mathbf{F}(\mathscr{Q}_{i-1})X^{i-1}Y^{i-1}} \,.$$

This data allows us to upper bound the advantage $\mathsf{Adv}_f^{\mathbf{E}\wedge\neg\mathbf{F}}(q)$ following exactly the same steps as in the straightforward method (for the random system g and the monotone event $\mathbf{E}$). Moreover, we assume all the corresponding notation. The following proposition provides an upper bound on the adaptive advantage.

Proposition 3.3.7. *Let f be an $(\mathscr{X},\mathscr{Y})$-random system with a monotone predicate $\mathbf{F}$ with the property that there exists a random system g such that $f|\mathbf{F} \equiv g$. Let $\mathbf{E}$ be a monotone predicate on g. Assuming that*

$$\sum_{i=1}^{q} \max_{(x^i,y^{i-1})} \left\{ \mathsf{P}^{f}_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})\neg\mathbf{F}(\mathscr{Q}_i)X^iY^{i-1}} \right\} < 1 \,,$$

for all $q \geq 1$ and all $(x^q, y^{q-1}) \in \mathscr{X}^q \times \mathscr{Y}^{q-1}$, we have

$$\mathsf{Adv}_f^{\mathbf{E}\wedge\neg\mathbf{F}}(q) \leq \sum_{i=1}^{q} \max_{(x^i,y^{i-1})} \left\{ \mathsf{P}^{f}_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})\neg\mathbf{F}(\mathscr{Q}_i)X^iY^{i-1}} \right\} .$$

Proof. That a slightly different version of Hypothesis 3.3.5 is satisfied can be shown as in the proof of Proposition 3.3.6. That is, we can show that there exists an $(\mathscr{X},\mathscr{Y})$-random system h such that $g|\mathbf{E} \equiv h$; hence for all $i \geq 1$ and all $(x^i, y^i) \in \mathscr{X}^i \times \mathscr{Y}^i$,

$$\mathsf{P}^{f}_{Y_i|\neg\mathbf{E}(\mathscr{Q}_i)\mathbf{F}(\mathscr{Q}_i)X^iY^{i-1}} = \mathsf{P}^{h}_{Y_i|X^iY^{i-1}} \,.$$

It simply follows from the hypothesis (as in Proposition 3.3.6) that

$$\sum_{i=1}^{q} \max_{(x^i,y^{i-1})} \left\{ \mathsf{P}^{f}_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})\neg\mathbf{F}(\mathscr{Q}_i)X^iY^{i-1}} \right\} < 1 \,.$$

Now observe that

$$\mathsf{P}^{\mathscr{A}\Diamond f}_{\mathbf{E}(\mathscr{Q}_q)\neg\mathbf{F}(\mathscr{Q}_q)} \leq \mathsf{P}^{\mathscr{A}\Diamond f}_{\mathbf{E}(\mathscr{Q}_q)|\neg\mathbf{F}(\mathscr{Q}_q)} = \mathsf{P}^{\mathscr{A}\Diamond g}_{\mathbf{E}(\mathscr{Q}_q)} \,.$$

Therefore, it is sufficient to get an upper bound for the latter probability to bound $\mathsf{Adv}_f^{\mathbf{E}\wedge\neg\mathbf{F}}(q)$. In the following, we proceed as in the proof of Proposition 3.3.6: we have

$\mathsf{Exp}_f^{\mathbf{E}\wedge\neg\mathbf{F}}(\mathscr{A})$:
Let $i \leftarrow 0, \mathcal{Q}_0 \leftarrow \emptyset$
While $i \leq q$ do
$i \leftarrow i+1$
$x_i \leftarrow \mathscr{A}(\mathcal{Q}_{i-1})$
$y_i \leftarrow f(x_i)$
$\mathcal{Q}_i \leftarrow \mathcal{Q}_{i-1} \cup \{(x_i, y_i)\}$
Return $\mathbf{E}(\mathcal{Q}_q) \wedge \neg\mathbf{F}(\mathcal{Q}_q)$.

$\mathsf{Exp}_f^{\mathbf{E},\kappa}(\mathscr{A})$:
Let $i \leftarrow 0, \mathcal{Q}_0 \leftarrow \emptyset, \mathsf{ctr}(0) \leftarrow 0$
While $i \leq q$ do
$i \leftarrow i+1$
$x_i \leftarrow \mathscr{A}(\mathcal{Q}_{i-1})$
$y_i \leftarrow f(x_i)$
$\mathcal{Q}_i \leftarrow \mathcal{Q}_{i-1} \cup \{(x_i, y_i)\}$
If hit_i then $\mathsf{ctr}_i \leftarrow \mathsf{ctr}_{i-1} + 1$
If $\mathsf{ctr}_q > \kappa$ then return `true`
Return `false` .

Figure 3.4 – The experiments analyzed in Propositions 3.3.7 and 3.3.10 illustrated (on the left and right, respectively).

$$
\begin{aligned}
1 - \max_{\mathscr{A}} \left\{ \mathsf{P}^{\mathscr{A}\Diamond f}_{\mathbf{E}(\mathcal{Q}_q)|\neg\mathbf{F}(\mathcal{Q}_q)} \right\} &= \min_{\mathscr{A}} \left\{ \mathsf{P}^{\mathscr{A}\Diamond f}_{\neg\mathbf{E}(\mathcal{Q}_q)|\neg\mathbf{F}(\mathcal{Q}_q)} \right\} = \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} \mathsf{P}^{\mathscr{A}\Diamond f}_{\neg\mathbf{E}(\mathcal{Q}_q)X^qY^q|\neg\mathbf{F}(\mathcal{Q}_q)} \right\} \\
&= \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} \left(\prod_{i=1}^{q} \mathsf{P}^{f}_{Y_i|\neg\mathbf{E}(\mathcal{Q}_i)\neg\mathbf{F}(\mathcal{Q}_i)X^iY^{i-1}} \mathsf{P}^{f}_{\neg\mathbf{E}(\mathcal{Q}_i)|\neg\mathbf{E}(\mathcal{Q}_{i-1})\neg\mathbf{F}(\mathcal{Q}_i)X^iY^{i-1}} \right.\right. \\
&\qquad \left.\left. \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathcal{Q}_{i-1})\neg\mathbf{F}(\mathcal{Q}_{i-1})X^{i-1}Y^{i-1}} \right) \right\} \\
&\geq \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} \left(1 - \sum_{i=1}^{q} \mathsf{P}^{f}_{\mathbf{E}(\mathcal{Q}_i)|\neg\mathbf{E}(\mathcal{Q}_{i-1})\neg\mathbf{F}(\mathcal{Q}_i)X^iY^{i-1}} \right) \right. \\
&\qquad \left. \left(\prod_{i=1}^{q} \mathsf{P}^{h}_{Y_i|X^iY^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathcal{Q}_{i-1})\neg\mathbf{F}(\mathcal{Q}_{i-1})X^{i-1}Y^{i-1}} \right) \right\} \\
&= \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} \left(1 - \sum_{i=1}^{q} \mathsf{P}^{f}_{\mathbf{E}(\mathcal{Q}_i)|\neg\mathbf{E}(\mathcal{Q}_{i-1})\neg\mathbf{F}(\mathcal{Q}_i)X^iY^{i-1}} \right) \right\} .
\end{aligned}
$$

Using the stepwise maximization we thus arrive at the desired claim as in Proposition 3.3.6. □

3.3.3 Considering More General Games

Now we aim to generalize the methods introduced in the previous section to be able to capture a broader spectrum of events and to derive better upper bounds. To this end, we consider two methods whose analyses are given in Propositions 3.3.8 and 3.3.10. In the first experiment, we simply take the straightforward approach with a crucial twist in the analysis: Instead of taking step-specific maximization, we opt for a direct upper bound B_Σ where we take the maximum of the sum of stepwise probabilities. The second, however, is the natural generalization of the first. These two techniques are crucial in the analysis performed in Chapter 5 (where the bounds given in Propositions 3.3.6 and 3.3.7 are not sufficiently good).

Before proceeding, let us develop some intuition for the analysis performed using B_Σ. For concreteness, consider a collision-finding adversary that adaptively makes three queries. Customarily (using Proposition 3.3.6 or 3.3.7), we would upper bound the maximum probability B_i (for $i = 1,2,3$) that

an adversary causes a collision on the i'th query, say with $1/4$ each; taking a union bound leads to an overall bound $3/4$. In this case, using the union bound is fine. Now, instead of step-specific bound, consider the global requirement $B_1 + B_2 + B_3 \le 1/2$. Here, each B_i could be $1/2$ itself; yet using $\sum_i B_i$ would lead to an overall bound of $3/2$ (which is vacuous for a probability). Nevertheless, (intuitively) no adversary should be able to do better than $1/2$.

In the following proposition, we make this observation formal and use this fact to obtain (possibly) better upper bounds for some of the experiments we are interested in. We remark that a further complication arises when we require the adversary to obtain a success at least twice, or more (e.g., finding not a single but multiple preimages for a digest Z). While it is still easy to deal with non-adaptive adversaries, properly taking care of adaptive adversaries is non-trivial. We consider this event (counting more successes) in Proposition 3.3.10 (see also Figure 3.4) using the bounds obtained in Proposition 3.3.8.

Proposition 3.3.8. *Let f be a random system with a monotone predicate* $\mathbf{E}$. *If there exists a value $B_\Sigma \in (0,1)$ such that for all $(x^q, y^q) \in \mathcal{X}^q \times \mathcal{Y}^q$*

$$\sum_{i=1}^{q} \mathsf{P}^{f}_{\mathbf{E}(\mathcal{Q}_i)|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^iY^{i-1}} \le B_\Sigma \,,$$

then

$$\mathsf{Adv}^{\mathbf{E}}_{f}(q) \le B_\Sigma \,.$$

Proof. We first show that Hypothesis 3.3.5 holds for all $i \le q$ and all (x^i, y^i); this follows as in the proof of Proposition 3.3.6. Specifically, $B_\Sigma < 1$ implies that

$$\mathsf{P}^{f}_{\mathbf{E}(\mathcal{Q}_i)|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^iY^{i-1}} < 1 \quad \text{and hence} \quad \mathsf{P}^{f}_{\neg\mathbf{E}(\mathcal{Q}_i)|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^iY^{i-1}} > 0 \,.$$

So, for a random system g, Hypothesis 3.3.5 holds; specifically, for all $i \ge 1$ and all $(x^i, y^i) \in \mathcal{X}^i \times \mathcal{Y}^i$,

$$\mathsf{P}^{f}_{Y_i|\neg\mathbf{E}(\mathcal{Q}_i)X^iY^{i-1}} = \mathsf{P}^{g}_{Y_i|X^iY^{i-1}} \,.$$

Following the similar steps from the proof of Proposition 3.3.6, we thus arrive at:

$$\begin{aligned} 1 - \mathsf{Adv}^{\mathbf{E}}_{f}(q) &\ge \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} \left(1 - \sum_{i=1}^{q} \mathsf{P}^{f}_{\mathbf{E}(\mathcal{Q}_i)|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^iY^{i-1}}\right) \prod_{i=1}^{q} \left(\mathsf{P}^{f}_{Y_i|\neg\mathbf{E}(\mathcal{Q}_i)X^iY^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^{i-1}Y^{i-1}}\right) \right\} \\ &= \min_{\mathscr{A}} \left\{ \sum_{(x^q,y^q)} \left(1 - \sum_{i=1}^{q} \mathsf{P}^{f}_{\mathbf{E}(\mathcal{Q}_i)|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^iY^{i-1}}\right) \prod_{i=1}^{q} \left(\mathsf{P}^{g}_{Y_i|X^iY^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^{i-1}Y^{i-1}}\right) \right\} , \end{aligned}$$

where we pulled out the term

$$\left(1 - \sum_{i=1}^{q} \mathsf{P}^{f}_{\mathbf{E}(\mathcal{Q}_i)|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^iY^{i-1}}\right)$$

by using step-specific maximization. Now instead, we leave it inside the minimum intentionally. Now suppose that there exists a value $B_\Sigma > 0$ such that for all $(x^q, y^q) \in \mathcal{X}^q \times \mathcal{Y}^q$

$$\sum_{i=1}^{q} \mathsf{P}^{f}_{\mathbf{E}(\mathcal{Q}_i)|\neg\mathbf{E}(\mathcal{Q}_{i-1})X^iY^{i-1}} \le B_\Sigma \,.$$

As B_Σ is independent of $\mathscr{A}$, we can pull the term containing B_Σ out of the minimum and achieve

$$1 - \mathsf{Adv}_f^{\mathbf{E}}(q) \geq (1 - B_\Sigma) \min_{\mathscr{A}} \left\{ \sum_{(x^q, y^q)} \left(\prod_{i=1}^{q} \mathsf{P}^{g}_{Y_i|X^i Y^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i|\neg\mathbf{E}(\mathscr{Q}_{i-1}) X^{i-1} Y^{i-1}} \right) \right\} = 1 - B_\Sigma \ .$$

The claim follows after noting that the big sum results in one. □

We remark that the above approach is not limited to the experiment $\mathsf{Exp}_f^{\mathbf{E}}(\mathscr{A})$ and it can well be used for the analysis of $\mathsf{Exp}_f^{\mathbf{E}\wedge\neg\mathbf{F}}(\mathscr{A})$. The following proposition states our corresponding claim.

Proposition 3.3.9. *Let f be a random system with a monotone predicate $\mathbf{F}$ with the property that there exists a random system g such that $f|\mathbf{F} \equiv g$. Let $\mathbf{E}$ be a monotone predicate on g. Suppose that there exists a value $B_\Sigma \in (0,1)$ such that for all $(x^q, y^q) \in \mathscr{X}^q \times \mathscr{Y}^q$*

$$\sum_{i=1}^{q} \mathsf{P}^{f}_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1})\neg\mathbf{F}(\mathscr{Q}_i) X^i Y^{i-1}} = \sum_{i=1}^{q} \mathsf{P}^{g}_{\mathbf{E}(\mathscr{Q}_i)|\neg\mathbf{E}(\mathscr{Q}_{i-1}) X^i Y^{i-1}} \leq B_\Sigma \ .$$

Then

$$\mathsf{Adv}_f^{\mathbf{E}\wedge\neg\mathbf{F}}(q) \leq B_\Sigma \ .$$

Proof. The proof follows from the proof of Propositions 3.3.7 and 3.3.8. □

In Proposition 3.3.8 we are mainly interested in estimating the maximum probability of one success occurring. Nevertheless, in some scenarios, the major monotone predicate $\mathbf{E}$ might depend on an auxiliary event that requires a higher number of *successes* to hold. As an example, let p be a uniformly random permutation $p\colon \mathscr{X} \to \mathscr{X}$ for $\mathscr{X} = \{0,1\}^n$ and let $\mathbf{E}(\mathscr{Q}_i)$ be the predicate that is set to true if $y_j = x_j$ for more than κ values of $j \leq i$ where $y_j = p(x_j)$ and κ is a positive integer. More precisely, $\mathbf{E}(\mathscr{Q}_i)$ is the predicate that is set to true if there exist more than κ fixed points after the i'th query.

Such a general problem can be modeled and studied using random systems as follows: Suppose that f is an $(\mathscr{X},\mathscr{Y})$-random system. We then attach a predicate called hit_i to the random system f; this is the success event at step i. Note that hit_i is not monotone. Moreover, we introduce a random variable ctr_i to indicate the number of successes up to step i. In other words, $\mathsf{ctr}_0 = 0$ and for every $j \geq 1$, $\mathsf{ctr}_j = \mathsf{ctr}_{j-1} + 1$ if hit_j occurs and $\mathsf{ctr}_j = \mathsf{ctr}_{j-1}$ otherwise. Finally, we can associate monotone predicates $\mathbf{E}_\kappa(\mathscr{Q}_i)$ for every integer $\kappa \geq 0$, so that $\mathbf{E}_\kappa(\mathscr{Q}_i)$ is the predicate that is set to true if there are more than κ successes after the i'th query (see Figure 3.4 on page 55 for the corresponding game). Obviously, for $\kappa = 0$ we are in the same scenario as in Proposition 3.3.8.

In order to attach the success event to the random system, we provide the following additional probabilities besides $\mathsf{P}^{f}_{Y_i|X^i Y^{i-1}}$ (data defining f):

1. Binary probabilities $\mathsf{P}^{f}_{\mathsf{hit}_i|X^i Y^i}$ for every $x^i \in \mathscr{X}^i$ and $y^i \in \mathscr{Y}^i$.
2. Probabilities $\mathsf{P}^{f}_{\mathsf{hit}_i|X^i Y^{i-1}}$ (derived from $\mathsf{P}^{f}_{Y_i|X^i Y^{i-1}}$ and $\mathsf{P}^{f}_{\mathsf{hit}_i|X^i Y^i}$ via Bayes' rule) for every $x^i \in \mathscr{X}^i$ and $y^{i-1} \in \mathscr{Y}^{i-1}$:
$$\mathsf{P}^{f}_{\mathsf{hit}_i|X^i Y^{i-1}} = \sum_{y_i} \mathsf{P}^{f}_{\mathsf{hit}_i|X^i Y^i} \mathsf{P}^{f}_{Y_i|X^i Y^{i-1}} .$$
3. Monotone predicate probabilities (as in (3.2)) denoted by $\mathsf{P}^{f}_{\neg\mathbf{E}_\kappa(\mathscr{Q}_i)|\neg\mathbf{E}_\kappa(\mathscr{Q}_{i-1}) X^i Y^{i-1}}$.

Proposition 3.3.10. *Let κ be a non-negative integer and suppose that there exists a value $B_\Sigma \in (0,1)$ such that for all $(x^q, y^q) \in \mathcal{X}^q \times \mathcal{Y}^q$,*

$$\sum_{i=0}^{q} \mathsf{P}^f_{\mathsf{hit}_i | X^i Y^{i-1}} \le B_\Sigma \qquad \textit{and} \qquad \mathsf{P}^f_{\mathsf{hit}_i | X^i Y^{i-1}} > 0 \quad \forall i \le q \,.$$

Then

$$\mathsf{Adv}^{\mathbf{E},\kappa}_f(q) \;\le\; B_\Sigma^{\kappa+1} \,.$$

Proof. We use induction on $\kappa + q$ where the inductive hypothesis is the following:

Hypothesis (κ, q)**:** For any random system f, any success events $\{\mathsf{hit}_i\}_{i=1}^q$ and any $B_\Sigma > 0$ for which

$$\sum_{i=0}^{q} \mathsf{P}^f_{\mathsf{hit}_i | X^i Y^{i-1}} \le B_\Sigma$$

holds for any $(x^q, y^q) \in \mathcal{X}^q \times \mathcal{Y}^q$, it follows that

$$\mathsf{Adv}^{\mathbf{E},\kappa}_f(q) \le B_\Sigma^{\kappa+1} \,.$$

We first check Hypothesis (κ, q) when either $\kappa = 0$ or $q = 1$. For $\kappa = 0$, this is Proposition 3.3.8. If $q = 1$, the statement is trivial to verify. Next, fix a pair (κ, q) such that $\kappa \ge 1$ and $q \ge 2$ and assume Hypothesis (κ', q') whenever $\kappa' + q' < \kappa + q$. Our goal is to find lower bounds on

$$\begin{aligned} 1 - \mathsf{Adv}^{\mathbf{E},\kappa}_f(\mathscr{A}) &= \sum_{(x^q, y^q)} \left(\prod_{i=1}^{q} \mathsf{P}^f_{\neg\mathbf{E}_\kappa(\mathscr{Q}_i) Y_i | \neg\mathbf{E}_\kappa(\mathscr{Q}_{i-1}) X^i Y^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i | \neg\mathbf{E}_\kappa(\mathscr{Q}_{i-1}) X^{i-1} Y^{i-1}} \right) \\ &= \sum_{x^1} \mathsf{P}^{\mathscr{A}}_{X_1} \sum_{y^1} \mathsf{P}^f_{Y_1 | X^1} \sum_{\substack{(x_2,\ldots,x_q) \\ (y_2,\ldots,y_q)}} \left(\prod_{i=2}^{q} \mathsf{P}^f_{\neg\mathbf{E}_\kappa(\mathscr{Q}_i) Y_i | \neg\mathbf{E}_\kappa(\mathscr{Q}_{i-1}) X^i Y^{i-1}} \mathsf{P}^{\mathscr{A}}_{X_i | \neg\mathbf{E}_\kappa(\mathscr{Q}_{i-1}) X^{i-1} Y^{i-1}} \right) . \end{aligned}$$

Here, we have used the fact that $\neg\mathbf{E}_\kappa(\mathscr{Q}_1)$ holds whenever $\kappa > 0$, i.e.,

$$\mathsf{P}^f_{Y_1 | \neg\mathbf{E}_\kappa(\mathscr{Q}_1) X^1} = \mathsf{P}^f_{Y_1 | X^1} \,.$$

Next, fix $x_1 \in \mathcal{X}$ and let $p(x_1) = \mathsf{P}^f_{\mathsf{hit}_1 | X_1}$. Then

$$\mathsf{P}^f_{Y_1 | X^1} = \mathsf{P}^f_{Y_1 | \mathsf{hit}_1 X^1} \mathsf{P}^f_{\mathsf{hit}_1 | X^1} + \mathsf{P}^f_{Y_1 | \neg\mathsf{hit}_1 X^1} \mathsf{P}^f_{\neg\mathsf{hit}_1 | X^1} = p(x_1) \mathsf{P}^f_{Y_1 | \mathsf{hit}_1 X^1} + (1 - p(x_1)) \mathsf{P}^f_{Y_1 | \neg\mathsf{hit}_1 X^1} \,. \tag{3.8}$$

The sums over $(x_2, \ldots, x_q)$ and $(y_2, \ldots, y_q)$ can be interpreted in terms of new random systems that have already been introduced by Maurer and Gazi [68], namely, projected random systems denoted by $f[x^1, y^1]$ and defined as follows:

$$\mathsf{P}^{f[x^1,y^1]}[Y_i = y_i' | X^i = (x_1', \ldots, x_i'), Y^{i-1} = (y_1', \ldots, y_{i-1}')] = \mathsf{P}^f[Y_{i+1} = y_i' | X^{i+1} = (x_1, x_1', \ldots, x_i'), Y^i = (y_1, y_1', \ldots, y_{i-1}')] \,.$$

Moreover, consider a success event for $f[x^1, y^1]$ derived from the success event hit_i: The success event at step i for $f[x^1, y^1]$ will correspond to the success event at step $i+1$ for f where the first query/response pair is exactly (x_1, y_1). The probabilities thus satisfy:

$$\mathsf{P}^{f[x^1,y^1]}[\mathsf{hit}_i | X^i = (x_1', \ldots, x_i'), Y^{i-1} = (y_1', \ldots, y_{i-1}')] = \mathsf{P}^f[\mathsf{hit}_{i+1} | X^{i+1} = (x_1, x_1', \ldots, x_i'), Y^i = (y_1, y_1', \ldots, y_{i-1}')] \,.$$

Furthermore, consider the naturally defined projected distinguisher for the projected random system g. The above formula for $\mathsf{Adv}_f^{\neg\mathbf{E},\kappa}(\mathscr{A})$ together with (3.8) imply

$$\mathsf{Adv}_f^{\neg\mathbf{E},\kappa}(\mathscr{A}) = \sum_{x_1} \mathsf{P}_{X_1}^{\mathscr{A}} \sum_{y^1} \mathsf{P}_{Y_1|\mathsf{hit}_1 X^1}^{f} p(x_1) \mathsf{Adv}_{f[x^1,y^1]}^{\neg\mathbf{E},\kappa-1}(\mathscr{A}') + \sum_{x_1} \mathsf{P}_{X_1}^{\mathscr{A}} \sum_{y^1} \mathsf{P}_{Y_1|\neg\mathsf{hit}_1 X^1}^{f} (1-p(x_1)) \mathsf{Adv}_{f[x^1,y^1]}^{\neg\mathbf{E},\kappa}(\mathscr{A}') ,$$

where $\mathscr{A}'$ behaves as the adversary $\mathscr{A}$ after the first query/response pair is obtained. As

$$\sum_{i=1}^{q-1} \mathsf{P}_{\mathsf{hit}_i|X^i Y^{i-1}}^{f[x^1,y^1]} \le B_\Sigma - p(x_1) ,$$

we can apply the inductive hypothesis to get

$$\mathsf{Adv}_{f[x^1,y^1]}^{\neg\mathbf{E},\kappa-1}(\mathscr{A}') \ge 1 - (B_\Sigma - p(x_1))^\kappa \qquad \text{and} \qquad \mathsf{Adv}_{f[x^1,y^1]}^{\neg\mathbf{E},\kappa}(\mathscr{A}') \ge 1 - (B_\Sigma - p(x_1))^{\kappa+1} .$$

Substituting these bounds and using

$$\sum_{y_1} \mathsf{P}_{Y_1|\mathsf{hit}_1 X_1}^{f} = 1 \quad \text{and} \quad \sum_{y_1} \mathsf{P}_{Y_1|\neg\mathsf{hit}_1 X_1}^{f} = 1$$

(these follow from the condition $0 < \mathsf{P}_{\mathsf{hit}_1|X_1}^{f} < 1$), we finally get

$$\mathsf{Adv}_f^{\neg\mathbf{E},\kappa}(\mathscr{A}) \ge \sum_{x_1} \mathsf{P}_{X_1}^{\mathscr{A}} \left(1 - p(x_1)(B_\Sigma - p_1)^\kappa - (1-p(x_1))(B_\Sigma - p(x_1))^{\kappa+1}\right) .$$

As

$$1 - p(x_1)(B_\Sigma - p(x_1))^\kappa - (1-p(x_1))(B_\Sigma - p(x_1))^{\kappa+1} \ge 1 - B_\Sigma^{\kappa+1}$$

whenever $0 < p(x_1) < B_\Sigma < 1$ and as

$$\sum_{x_1} \mathsf{P}_{X_1}^{\mathscr{A}} = 1 ,$$

we obtain the desired bound. □

4 Another Look at Double-Block-Length Hash Functions

In this chapter, we take Stam's approach [191] (see Theorem 2.3.7) on single-call blockcipher-based compression functions, introduced in Section 2.3.3, as a basis and generalize his method to the double-call DBL blockcipher-based compression functions (Definition 2.3.8) with $\kappa = s = 2n$ and $m = n$ (see Figure 4.1). Our motivation is not very different from that of Stam: There are certain double-call DBL blockcipher-based compression functions (e.g., [75, 76]), with the above mentioned parameters, already proposed in the literature; nevertheless, there is not a single work that captures a deeper understanding and reasoning of what makes these designs secure or insecure. Our goal is therefore to provide a more conceptual understanding for a large class of double-block-length compression functions of this type and study the sufficient conditions required to make them secure. Moreover, we work out the sufficient conditions to get a close-to-optimal collision resistant (iterated) hash function even if the underlying compression is not optimally collision resistant (i.e., without using the Merkle–Damgård paradigm).

To this end, we group the schemes according to two important characteristics: Firstly, we distinguish between collision resistance in the compression function (Type-I) and collision resistance in the (Merkle–Damgård) iteration (Type-II), similarly to the classic case by Black et al. [26, 27]. Secondly, for secure compression functions only (Type-I), we differentiate between schemes where the two blockciphers E^1 and E^2 are distinct and independently sampled from $\mathrm{Block}(2n, n)$ (see Section 2.3.1 for the definition of $\mathrm{Block}(\kappa, n)$); and schemes where only a single blockcipher is used, so $E^1 = E^2$. Each type is defined, similar to Theorem 2.3.7, by a set of conditions on pre and postprocessing functions. For Type-I schemes with distinct blockciphers, the requirements are a rather straightforward generalization of those by Stam for single-call constructions. For Type-I schemes with only a single blockcipher, we follow Hirose's [76] (and Nandi's [128]) approaches for implicit domain separation based on some extra conditions on $C_2^{\mathsf{pre}}(C_1^{-\mathsf{pre}}(\cdot))$ (see Definition 2.3.8 for the definition of C_1^{pre} and C_2^{pre}, as well as Figure 4.1).

We show that for Type-I schemes we need to ask $\Omega(2^n)$ queries (asymptotically in n) to the underlying blockciphers in order to find a collision with high probability (see Theorems 4.1.2 and 4.2.2 for the precise statements). For preimage resistance, we are able to show the same bound; obviously it is suboptimal (as the digest size is $2n$ bits). We make use of the recent work of Armknecht et al. [8] to improve our result on preimage resistance to an almost optimal bound. For Type-II schemes, the generalization from the classic single-call case is less straightforward. In fact, we can only prove (slightly) suboptimal collision resistance—we lose a factor that is logarithmic in n—although we

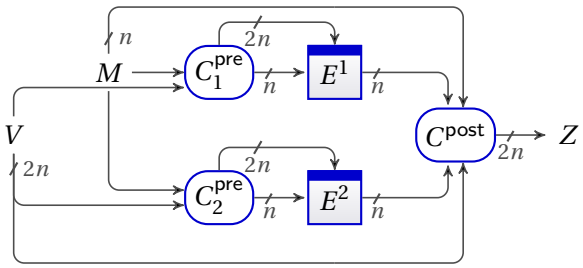

Figure 4.1 – Double-call DBL blockcipher-based compression functions (see Definition 2.3.8 with parameters $\kappa = s = 2n$ and $m = n$) considered in Chapter 4. In Section 4.2, only a single blockcipher is used, so $E^1 = E^2$. In Section 4.3, we consider compression functions without feedforward (i.e., V and M are not fed into C^{post}).

do believe that our conditions suffice for optimal collision resistance (see Figure 4.3 on page 76 for the comparison of our collision resistance bounds obtained in this chapter for $n = 128$). Finally, we conclude that the preimage resistance of Type-II schemes are not optimal; we illustrate this with a concrete preimage attack with $\mathcal{O}(2^{3n/2})$ queries.

The chapter is organized as follows: In Section 4.1, we study the constructions with distinct and independently sampled blockciphers. In Section 4.2, we investigate the secure compression functions instantiated with a single blockcipher. In Section 4.3, we look at the security in the iteration and finally in Section 4.4 we detail the ramifications of our results on $\mathbb{F}_2$-'block'-linear instances of C^{pre} and C^{post}.

4.1 Compression Functions with Distinct and Independent Blockciphers

Definition 4.1.1 introduces the Type-I compression functions, which we construct by naturally extending the conditions given in Theorem 2.3.7 for single-call blockcipher-based compression functions. In the following we use the notation used in Definition 2.3.8.

Definition 4.1.1. A double-call DBL blockcipher-based compression function $\mathsf{h}^{E^1,E^2} : \{0,1\}^{3n} \to \{0,1\}^{2n}$ (see Definition 2.3.8 with $\kappa = s = 2n$ and $m = n$, as well as Figure 4.1) is called Type-I if and only if the following three conditions hold:

1. $C_1^{pre} : \{0,1\}^{3n} \to \{0,1\}^{3n}$ (the map that takes V, M to k_1 and x_1) and $C_2^{pre} : \{0,1\}^{3n} \to \{0,1\}^{3n}$ (the map that takes V, M to k_2 and x_2) are both bijections;
2. $C^{post}(V, M, \cdot, \cdot)$ (the map $\{0,1\}^{2n} \to \{0,1\}^{2n}$ that takes y_1 and y_2 to Z for a fixed (V, M)) is a bijection (from $2n$ to $2n$ bits, for all (V, M));
3. $C_1^{aux} = C^{post}(C_1^{-pre}(k_1, \cdot), y_1, \cdot)$ is a bijection (for all k_1, y_1) and $C_2^{aux} = C^{post}(C_2^{-pre}(k_2, \cdot), \cdot, y_2)$ as well (for all k_2, y_2). Here C_1^{aux} is the function that maps x_1 and y_2 to Z for a fixed (k_1, y_1); and C_2^{aux} is the function that maps x_2 and y_1 to Z for a fixed (k_2, y_2).

The first condition is bijectivity of C_1^{pre} and C_2^{pre}; it guarantees that each E^1, D^1, E^2 and D^2 query leads to a single compression function output. This limits the growth of the maximum number of compression function evaluations given the query quota, i.e., yield(q) (Definition 2.3.1). The second condition ensures the transfer of the full randomness from the E^1 and E^2 outputs to the compression

function output Z. We note that the compression function input (V, M) is fed to C^{post} (we call it feedforward) to avoid straightforward collision- and preimage-finding (see Theorem 4.3.2 for the corresponding attacks when the feedforward is omitted). The third condition is similar to C^{post} in the sense that the unpredictability of the decryption queries followed by an encryption query is used optimally for the compression function output (note the difference between calling D^1 and D^2 due to two different auxiliary functions C_1^{aux} and C_2^{aux}).

The following result gives close-to-optimal collision resistance for DBL Type-I compression functions: It states that for any number of queries q at most 2^{n-1}, the advantage of any collision-finding adversary is less than $1/2$ (see Figure 4.3 on page 76 for the illustration of the bound obtained from Theorem 4.1.2 as a function of $\log_2(q)$). In the proof below, the adversary makes at most q queries to each of its two primitives E^1 and E^2 (where the decryption queries to D^1 and D^2 are counted for their respective encryption primitive). Because a compression function evaluation itself takes two queries, this means we lose a factor of at most four in the bound below, compared to the lower bound based on a generic (birthday) collision attack against the compression function.

Theorem 4.1.2. *Let h^{E^1,E^2} be a DBL Type-I compression function (Definition 4.1.1). Then the advantage of an adversary in finding a collision in h^{E^1,E^2} after $q \le 2^{n-1}$ queries can be upper bounded by*

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q) \le \frac{q(q-1)}{2(2^n-q)^2} .$$

Proof. A collision consists of two compression function input pairs (V,M) and (V',M') that satisfy $\mathsf{h}^{E^1,E^2}(V,M) = \mathsf{h}^{E^1,E^2}(V',M')$ and $(V,M) \neq (V',M')$. We construct a list of triples (V,M,Z) such that $Z = \mathsf{h}^{E^1,E^2}(V,M)$ and we assume that the adversary has made the relevant queries to E^1 and E^2 (where the decryption queries to D^1 and D^2 are counted for their respective encryption primitive). We bound the probability of a collision occurring in this list by making use of Proposition 3.3.6. In particular, we upper bound the maximum probability that any adversary can trigger collisions on the i'th step, given the adversary's i'th query, query history up to the i'th step and that collisions are not present yet (note that we let $\mathsf{coll}(\mathscr{Q}) \equiv \mathbf{E}$ in the terminology of Proposition 3.3.6; in other words, $\mathsf{coll}(\mathscr{Q})$ does not depend on other auxiliary monotone events). Then the union bound (over q queries) provides the overall upper bound.

Bijectivity of C_1^{pre} ensures that any query to E^1 (or D^1) adds at most one triple (V,M,Z) to $\mathsf{yieldset}^{\mathsf{h}}(\mathscr{Q})$ (Definition 2.3.1). Moreover, any query to E^1 corresponds uniquely to a forward query (x_2,k_2) to E^2, by the bijectivity of C_2^{pre}. (In the case of a query to D^1, the relevant value of (x_2,k_2) is only known after the D^1-query has been answered.) The case for queries made to E^2 is similar due to symmetry. We assume that an adversary asks a query to E^1 and the corresponding query to E^2 in conjunction—or rather, we consider a derived adversary obeying this rule—and henceforth refer to these tuples as query pairs. Note that after q queries by the original adversary, there are at most q query pairs for the derived adversary and that the advantage of the derived adversary is at least that of the original one [1].

As a result, after $i-1$ query pairs $\mathsf{yieldset}^{\mathsf{h}}(\mathscr{Q})$ contains exactly $i-1$ triples (V,M,Z). We claim that the probability of the i'th query pair causing a collision with any of these triples is at most

1. To justify this, simply note that $\mathscr{A}$ queries E^1 followed by E^2 (or vice versa); or D^1 (or D^2) followed by E^2 (E^1, respectively). The derived adversary $\mathscr{A}'$ runs $\mathscr{A}$ and collects queries without making extra ones. Therefore, the query list of the derived adversary $\mathscr{A}'$ contains the same set of queries (and responses) as $\mathscr{A}$; thus a collision is present for $\mathscr{A}'$ whenever $\mathscr{A}$ finds a collision and $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(\mathscr{A}) \le \mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(\mathscr{A}')$.

$(i-1)/(2^n-i+1)^2$. Using a union bound, the probability of a collision after q queries can then be upper bounded by

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

To finalize the proof, we need to back-up our claim for success on the i'th query pair. Let us distinguish the analysis depending on the first query of the pair.

Consider a forward query (k_1, x_1) to E^1. By the bijectivity of C_1^{pre} and C_2^{pre}, the corresponding values (V, M) and (k_2, x_2) are uniquely determined. Suppose that so far $t_1 \leq i-1$ queries to E^1 have been made involving key k_1 and $t_2 \leq i-1$ to E^2 involving key k_2. The answer to a fresh query to $E^1_{k_1}(\cdot)$ is therefore distributed uniformly over a set of $2^n - t_1$ possible outcomes and, similarly, the answer to a fresh query to $E^2_{k_2}(\cdot)$ is distributed uniformly over a set of at least $2^n - t_2$ possible outcomes. Each possible answer (y_1^*, y_2^*) is combined under C^{post} with the pair (V, M) that is consistent with the (k_1, x_1) and (k_2, x_2) queries being made, leading to a possible compression function outcome Z^*. Because C^{post} is bijective when the pair (V, M) is fixed, distinct (y_1^*, y_2^*) lead to distinct Z^*, so there are at least $(2^n-(i-1))^2$ possible outcomes Z^*, all equally likely. The probability of hitting a set of size $(i-1)$ is therefore at most $(i-1)/(2^n-i+1)^2$, as claimed. Because the situation is symmetric for a forward query made to E^2, the same analysis follows.

Similarly, consider an inverse query (k_1, y_1). This yields a unique x_1 and hence by the bijectivity of C_1^{pre}, there is a unique pair (V, M) corresponding to this query once answered. Thus, each inverse query adds one triple (V, M, Z) to the adversary's list of computable values. Again, the unique pair (V, M) fully determines (k_2, x_2) and (obviously) y_2, by the bijectivity of C_2^{pre} and E^2. Now, suppose that so far t_1 queries to E^1 (or D^1) have been made involving key k_1, resulting in t_1 plaintext-ciphertext pairs. The answer to a fresh query to $D^1_{k_1}(\cdot)$ is therefore different from the previous plaintexts. Moreover, each of the $2^n - t_1$ answers is equally likely if E^1 is an ideal cipher. Each possible answer x_1^* is combined under $C_1^{-\mathsf{pre}}$ and produces a unique (V, M) pair which also determines the forward query to E^2. Assuming E^2 is an ideal cipher, the output y_2^* of this query is distributed uniformly over a set of at least $2^n-(i-1)$ possible outcomes. This time, the bijectivity of C_1^{aux} implies that the uncertainty in x_1^* and y_2^* is fully inherited by Z; in other words, Z is uniform over a set of size at least $(2^n-(i-1))^2$. Again the probability of hitting any of the $i-1$ previously computed digests is as claimed. The result for the inverse query made to D^2 follows similarly because C_2^{aux} is bijective. Hence, the statement follows. □

The following result gives an upper bound for $\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q)$, which implies a bound by Hirose [75]. However, this expression is far from optimal and vacuous for $q \geqslant 2^n$. Indeed, the denominator of the upper bound gets closer to zero once the number of queries surpasses 2^{n-1}. Nevertheless, it states that the success probability of finding a preimage is negligible if $q \leq 2^{n-1}$.

Theorem 4.1.3. *Let h^{E^1,E^2} be a DBL Type-I compression function (Definition 4.1.1). Then the advantage of an adversary in finding a preimage in h^{E^1,E^2} after $q \leq 2^{n-1}$ queries can be upper bounded by*

$$\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q) \leq \frac{q}{(2^n-q)^2} \,.$$

Proof. Let $\mathscr{A}$ be an adversary trying to find a preimage for its input σ, asking a total of q queries to (E^1, D^1, E^2, D^2). As done in the proof of Theorem 4.1.2, we only consider a derived adversary

that makes its queries to E^1 and E^2 in pairs, allowing it to construct triples (V, M, Z) such that $Z = \mathsf{h}^{E^1,E^2}(V, M)$. A preimage of σ is found if and only if σ occurs as Z in the list of triples constructed this way, so it suffices to bound this probability instead.

As before, the i'th query pair adds one triple (V, M, Z) to the list of computable values where Z is uniform over a set of size at least $(2^n - i + 1)^2$ (by the bijectivity of C^{post} for forward-forward queries and C_j^{aux} for backward-forward queries for $j = 1, 2$). Thus, the probability that the i'th query pair finds a preimage is at most $1/(2^n - i + 1)^2$. With a union bound, the probability of finding a preimage after q queries can then be upper bounded by

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

which concludes the proof. □

Unfortunately in the above bound, once the number of queries q approaches 2^n, the denominator tends to zero; thus the upper bound gets too large. Consequently, we cannot prove resistance beyond 2^n queries using the above proof technique. The main reason for this is the term q appearing in the denominator; if we can somehow remove it from the denominator or at least diminish its negative effect, then there will be a hope to attain better security bounds.

Armknecht et al. [8] manage to find a way to achieve this by using "super queries". The idea is the following. Assume that an adversary makes queries to the blockcipher with the very same key K. Once the number of queries with this K is above a certain threshold, say 2^{n-1}, then the remaining 2^{n-1} queries (again with the same key K) are given to the adversary for *free*. By "free" we mean that they are not counted as the resource used by the adversary; but they may well be used by the adversary (non-adaptively) for preimage-finding. By a super query, we mean the set of 2^{n-1} free queries that are given to $\mathscr{A}$. Although the information gathered by the adversary, as well as its preimage-finding advantage increase considerably this way, this approach does not lead to an unacceptable security bound, as shown in Theorem 4.1.4: Almost optimal preimage resistance can be guaranteed using DBL Type-I blockcipher-based compression functions.

Theorem 4.1.4 (Armknecht et al. [8])**.** *Let h^{E^1,E^2} be a DBL Type-I blockcipher-based compression function (Definition 4.1.1). If (additionally) there exists a permutation $\xi : \{0,1\}^{2n} \to \{0,1\}^{2n}$ such that $k_2 = \xi(k_1)$, where k_i is the key used in E^i for $i = 1, 2$, then the advantage of an adversary in finding a preimage in h^{E^1,E^2} after $q \leq 2^{2n-3}$ queries can be upper bounded by*

$$\mathsf{Adv}_{\mathsf{h}}^{\mathsf{epre}}(q) \leq \frac{8q}{2^{2n}} .$$

Proof. Let $\mathscr{A}$ be an adversary that tries to find a preimage for its input σ. We follow a similar strategy as in the proof of Theorem 4.1.3 when $\mathscr{A}$ selects its queries. Specifically, when $\mathscr{A}$ makes an E^1-query, the corresponding E^2-query is asked in conjunction; or as preferred by Armknecht et al., it is given to $\mathscr{A}$ for free. Similarly, when $\mathscr{A}$ makes a D^1-query, the corresponding E^2-query is given for free as well. We note again that these free queries are not counted as queries asked by $\mathscr{A}$.

The strategy for using super queries is as follows. Assume that $\mathscr{A}$ makes 2^{n-1} queries (without loss of generality) to E^1 all with the key k_1; the remaining 2^{n-1} queries that contain k_1 are given to $\mathscr{A}$ for

free. The adversary $\mathscr{A}$ simply uses these queries non-adaptively to find a preimage. In addition, by using the fact that there exists a permutation $\xi : \{0,1\}^{2n} \to \{0,1\}^{2n}$ such that $k_2 = \xi(k_1)$, we give the corresponding E^2 queries (with the key $k_2 = \xi(k_1)$) to $\mathscr{A}$ for free as well. Note that a query (with k_1) is part of a super query if and only if the query with the corresponding key $k_2 = \xi(k_1)$ is as well, where we use the term super query to denote the set of 2^{n-1} free query pairs. Note that once a super query is completed (with a specific pair of keys, e.g., k_1 and $\xi(k_1)$), $\mathscr{A}$ cannot make any more queries with the keys corresponding to the super query in question. We stress that $\mathscr{A}$ can only find a preimage either using a normal [2] query or a query that is part of a super query. We bound the probability of finding a preimage for each case and, with a union bound, complete the proof. Let us investigate each case one by one; we begin with finding a preimage with normal queries.

Consider a forward query (k_1, x_1) to E^1. By the bijectivity of C_1^{pre} and C_2^{pre}, the corresponding values (V, M) and (k_2, x_2) are uniquely determined. Moreover, by the bijectivity of C^{post}, there is a unique pair (y_1, y_2) that can lead to a preimage for σ. Now assume that there are at most 2^{n-1} queries to E^1 that have been made involving key k_1 and 2^{n-1} to E^2 involving key k_2. The answers to the fresh queries to $E^1_{k_1}(\cdot)$ and $E^2_{k_2}(\cdot)$ are therefore distributed uniformly over a set of at least $(2^{n-1})^2$ possible outcomes. The probability of hitting Z is then at most $1/(2^{n-1})^2 = 4/2^{2n}$. As the situation is symmetric for a forward query made to E^2, the same analysis follows. Furthermore, bounding the probability for an inverse query followed by a forward query is analogous. In this case, only the bijectivity of C_1^{aux} and C_2^{aux} is used. Because $\mathscr{A}$ makes at most q queries, the probability of obtaining a collision with normal queries is at most $4q/2^{2n}$.

Now consider the event that $\mathscr{A}$ completes the preimage by using a query that is part of a super query. Recall that if (x_1, k_1) is part of a super query, so is the corresponding E^2-query (x_2, k_2); hence we consider these queries as pairs. It is easy to see that out of q queries, we can form a set of super queries of cardinality at most $q/2^{n-1}$ (as each super query requires an initial cost of 2^{n-1}); and each super query contains 2^{n-1} query pairs. For each pair in a super query, the probability of completing a preimage is $1/(2^{n-1})^2$. Indeed, as these queries are asked non-adaptively by $\mathscr{A}$, their responses are distributed randomly over the responses; using that C^{post} is bijective we attain the above probability. With a first union bound over 2^{n-1} pairs in a single super query, followed by a second over the set of super queries (of cardinality at most $q/2^{n-1}$) lead to the overall upper bound

$$2^{n-1}\left(\frac{q}{2^{n-1}}\right)\left(\frac{4}{2^{2n}}\right) = \frac{4q}{2^{2n}}\,.$$

Summing up the various probabilities completes the proof. □

4.2 Using a Single Blockcipher: Implicit Domain Separation

Previously, we assumed two independently sampled blockciphers. In practice, it is more realistic if only one blockcipher is used, twice in this case per compression function. From a theoretical point of view, this can easily be enforced by using explicit domain separation at only a small cost. In particular, we can define $E^1_{K'}(X) = E_{0||K}(X)$ and $E^2_{K'}(X) = E_{1||K}(X)$. Here the domain separation is explicit in the sense that the key spaces of E^1 and E^2 are ensured explicitly to be different. Although the cost of this explicit separation is only one key bit, it is not an entirely satisfactory situation; many

2. By normal query, we mean a query that is not a part of super query.

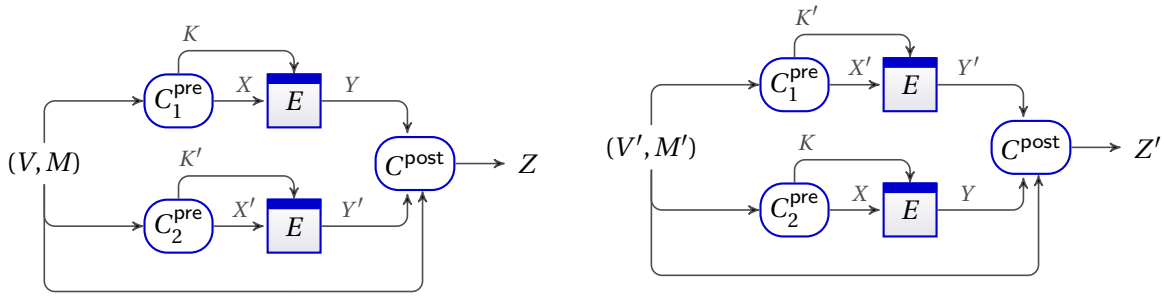

Figure 4.2 – The conjugate pairs (V,M)-(V',M') and (K,X)-(K',X') for IDS DBL Type-I compression functions illustrated for $p = C_2^{\text{pre}}(C_1^{-\text{pre}}(\cdot))$; hence $p(K,X) = (K',X')$.

existing constructions do not use this explicit separation and they would consequently fall outside any framework that relies upon it.

Recently Nandi [128] and Hirose [76] have shown how implicit domain separation can be achieved based on an involution without fixed points. In particular, consider $p = C_2^{\text{pre}}(C_1^{-\text{pre}}(\cdot))$; then p is an involution without fixed points if and only if $p(x) \neq x$ yet $p^2(x) = x$ for all $x \in \{0,1\}^{3n}$. (Note that Nandi and Hirose always assume C_1^{pre} to be the identity function.) The focus of Nandi and Hirose is on the hash function constructions with multiple calls to the same primitive; however, during execution, the inputs of these primitives are guaranteed to be different. This input separation is not achieved explicitly; the same blockciphers are used during execution. In contrast, it is imposed implicitly using the fact that $p(x) \neq x$. More precisely, an E-query $(K,\ X)$ corresponds to yet another E-query $(K',X') = p(K,X)$ and because $p(x) \neq x$ holds, both queries are guaranteed to be different. The same holds for the decryption queries. We incorporate this approach into our general framework.

In the definition below, we first repeat the first three requirements from Definition 4.1.1 (where the domain separation was still explicit). New requirement 4. below captures the notion of $p = C_2^{\text{pre}}(C_1^{-\text{pre}}(\cdot))$ being an involution without fixed points. The advantage of using involutions is that, as in the case for explicit domain separation, relevant queries to E still come in pairs. That is, for $(K,X) \in \{0,1\}^{3n}$ we call $(K',X') = p(K,X)$ its *conjugate* (and note that $(K,X) = p(K',X')$). Queries $E_K(X)$ and $E_{K'}(X')$ are similarly called conjugate queries. An inverse query $D_K(Y)$, once answered, has a uniquely defined conjugate (forward) query as well. Finally $(V,M) \in \{0,1\}^{3n}$ has conjugate

$$(V',M') = C_2^{-\text{pre}}(C_1^{\text{pre}}(V,M))$$

(see Figure 4.2 for an illustration). Fixed points in p are prohibited. They would correspond to the situation where

$$C_1^{\text{pre}}(V,M) = C_2^{\text{pre}}(V,M)\ ,$$

so a single query (to E) would suffice to evaluate the compression function (for that particular input (V,M)). If p would consist of only fixed points (i.e., equal the identity function), then the construction is essentially a supercharged single-call compression function [191] (see Section 2.3.3). As it is known that even in this case almost optimal collision resistance can be achieved, in principle a smaller number of fixed points in p could theoretically also be dealt with.

Similarly, new requirement 5. below, which states that two conjugate message-state pairs never collide with each other, could be relaxed to a situation where conjugate message-state pairs only

cause a collision (with each other) with a certain probability[3]. The theorem statement below can be amended by adding the sum of the largest such 'conjugate-internal' collision probabilities to the upper bound on the advantage.

Definition 4.2.1. A double-call DBL blockcipher-based compression function $\mathsf{h}^E : \{0,1\}^{3n} \to \{0,1\}^{2n}$ (see Definition 2.3.8 where $E^1 = E^2$) with $\kappa = s = 2n$ and $m = n$ is called IDS DBL Type-I if and only if the following five conditions hold: 1., 2. and 3. as in Definition 4.1.1 and additionally

4. For all V, M, V', M', if $C_1^{\mathsf{pre}}(V,M) = C_2^{\mathsf{pre}}(V',M')$ then

$$(V,M) \neq (V',M') \quad \text{and} \quad C_1^{\mathsf{pre}}(V',M') = C_2^{\mathsf{pre}}(V,M)\ .$$

5. For all $(V,M) \neq (V',M')$ with $C_1^{\mathsf{pre}}(V,M) = C_2^{\mathsf{pre}}(V',M')$ and all y_1, y_2, it holds that

$$C^{\mathsf{post}}(V,M,y_1,y_2) \neq C^{\mathsf{post}}(V',M',y_2,y_1)\ .$$

As in the proof of Theorem 4.1.2 (and, of course, in the proofs given by Hirose and Nandi [76, 128]) we only consider a derived adversary that always asks conjugate queries together. However, in contrast to the scenario with two distinct blockciphers, in this case such a query pair yields two compression function evaluations (for some (V,M) and its conjugate). Consequently, the bound here is bigger (roughly by a factor of two) for the same number of queries compared to that of Theorem 4.1.2 (see Figure 4.3 on page 76 for a more concrete comparison).

Finally, we do not consider more general functions p; we only look at involutions. In particular, our framework does not include the well-known Abreast-DM [102] scheme. In that case, we can generalize by regarding longer cycles in p as well. This has recently been worked out (independently) by Lee and Kwon [104] and Fleischmann et al. [65]. Roughly speaking, they prove that if p^c is the identity function for (smallest) $c > 2$ (and given certain natural restrictions on the postprocessing), then the collision-finding advantage can be upper bounded by

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

For Abreast-DM it holds that $c = 6$. We remark that the proof of Theorem 4.2.2 can easily be generalized to capture this case as well.

Theorem 4.2.2. *Let h^E be an IDS DBL Type-I compression function (Definition 4.2.1). Then the advantage of an adversary in finding a collision in h^E after $q < 2^{n-1}$ queries can be upper bounded by*

$$\mathsf{Adv}_{\mathsf{h}}^{\mathsf{coll}}(q) \leq \frac{2q(q-1)}{(2^n - 2q)^2}\ .$$

Proof. Let $\mathscr{A}$ be a collision-finding adversary asking its oracles E and D a total of q queries. We consider the derived adversary $\mathscr{A}'$ that asks conjugate queries in pairs: I.e., if $\mathscr{A}$ queries (K,X) to E, then $\mathscr{A}'$ queries both (K,X) and its conjugate $(K',X') = p(K,X)$. We bound the probability of the i'th query pair causing a collision.

3. We remark that many concrete examples in this class do not satisfy the requirement 5. We leave this rather strong requirement here for simplicity; as it can be seen from the proof of Theorem 4.2.2, requirement 5. can be easily dropped at the expense of an additive term in the bound that provides an upper bound for the probability of occurring of requirement 5.

Our claim is that the probability that the i'th query pair causing a collision is upper bounded by

$$\frac{4(i-1)}{(2^n-(2i-2))(2^n-(2i-1))}\,.$$

Note that the new query pair adds two triples (V,M,Z) and (V',M',Z') to the list of computable digests, where (V,M) and (V',M') are conjugates. Moreover (by requirement 5. in Definition 4.2.1), we are guaranteed that $Z \neq Z'$; so we only need to worry about either Z or Z' being in the list of digests already known.

We proceed similarly to the proof of Theorem 4.1.2: As each query pair adds two tuples to the list, there are $2(i-1)$ possible digest values to hit, either by (V,M,Z) or by (V',M',Z'). Although the two distributions are strongly dependent, each of Z and Z' is distributed individually over a set of size at least $(2^n-(2i-2))(2^n-(2i-1))$ (see below for the justification). With a union bound, the probability of a collision on the i'th query can therefore be upper bounded by

$$\frac{4(i-1)}{(2^n-2i+1)^2}\,.$$

Using a further union bound, the probability of a collision after q queries can be upper bounded by

$$\frac{2q(q-1)}{(2^n-2q)^2}\,.$$

We still need to prove our claim on the distribution of Z and Z'.

For a forward-forward pair, y_1 is distributed uniformly random over a set of cardinality at least $(2^n-(2i-2))$ and y_2 over a set of cardinality at least $(2^n-(2i-1))$ (assuming y_1 is queried first, otherwise the roles change). Because C^{post} is bijective, the claim follows. For a mixed inverse-forward pair first x_1 is returned by a query (k_1,y_1) to D. It is distributed uniformly over a set of size at least $(2^n-(2i-2))$. Subsequently y_2 is returned by the conjugate query (k_2,x_2) to E; it is distributed uniformly over a set of size at least $(2^n-(2i-1))$. Bijectivity of C_1^{aux} and C_2^{aux} finishes the proof. □

The following result gives an upper bound for $\mathsf{Adv}_{\mathsf{h}}^{\mathsf{epre}}(q)$ for IDS-DBL Type-I compression functions (Definition 4.2.1). Similar to the one given in Theorem 4.1.3, it is vacuous for $q \geqslant 2^{n-1}$ and only concludes that the success probability of finding a preimage is negligible for $q \leq 2^{n-2}$. Similar to the distinct and independently sampled blockciphers case, we can improve the bound given in Theorem 4.2.3 to an almost optimal bound using the methods of Armknecht et al.; we present this case in Theorem 4.2.4.

Theorem 4.2.3. *Let h^E be a DBL compression function of IDS-DBL Type-I (Definition 4.2.1). Then the advantage of an adversary in finding a preimage in h^E after $q < 2^{n-1}$ queries can be upper bounded by*

$$\mathsf{Adv}_{\mathsf{h}}^{\mathsf{epre}}(q) \leq \frac{2q}{(2^n-2q)^2}\,.$$

Proof. Let $\mathscr{A}$ be an adversary that tries to find a preimage for its input σ. We assume that $\mathscr{A}$ asks each of its oracles E and D a total of q queries. Similarly to the proof of Theorem 4.2.2, we consider the derived adversary $\mathscr{A}'$ that asks conjugate queries in pairs. We want to bound the probability that the i'th query pair leads to a preimage for σ.

After $i-1$ queries, we know that the list of computable values contains exactly $2(i-1)$ triples (V,M,Z). The i'th query adds the pair (V,M,Z) and (V',M',Z') to the list of computable digests, where (V,M) and (V',M') are conjugates. Our claim is that the probability that the i'th query pair leads to a preimage for σ is upper bounded by

$$\frac{2}{(2^n-(2i-2))(2^n-(2i-1))}\,.$$

By the same reasoning from the proof of Theorem 4.2.2, Z and Z' are distributed over a set of size at least $(2^n-(2i-2))(2^n-(2i-1))$. As we have only one value to hit, the probability of finding a preimage for each conjugate queries is at most

$$\frac{1}{(2^n-(2i-2))(2^n-(2i-1))}\,,$$

hence twice this probability for the i'th pair. Union bound (over q conjugate query pairs) together with the fact that

$$(2^n-(2i-2))(2^n-(2i-1)) > (2^n-2q)^2$$

give the desired result. □

Similar to Theorem 4.1.4, the bound given in Theorem 4.2.3 can be improved by using the methods of Armknecht et al. [8] who only give a proof of close-to-optimal preimage resistance of Hirose's scheme [76], which can be seen as an example of the IDS-DBL Type-I compression function with a slight modification (i.e., without requirement 5.). Here, we generalize their result to IDS-DBL Type-I compression functions.

Theorem 4.2.4. *Let h^E be an IDS-DBL Type-I blockcipher-based compression function (Definition 4.2.1). If (additionally) there exists a permutation $\xi : \{0,1\}^{2n} \to \{0,1\}^{2n}$ such that $\xi(K) = K'$, where K and K' denote the keys used in conjugate query pairs, then the advantage of an adversary in finding a preimage in h^E after $q \le 2^{2n-4}$ queries can be upper bounded by*

$$\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q) \le \frac{8q}{2^{2n}} + \frac{4q}{2^{n-1}(2^{n-1}-1)}\,.$$

Proof. The proof goes along similar lines with that of Theorem 4.1.4 with a minor modification that considers the conjugate queries. Let $\mathscr{A}$ be a preimage-finding adversary that tries to find a preimage for its input σ. When $\mathscr{A}$ makes an E-query (K, X), its conjugate query pair (K', X') (for $p(K,X) = (K',X')$ and $\xi(K) = K'$) is given to $\mathscr{A}$ for free. Similarly, when $\mathscr{A}$ makes a D-query (K, Y), (after the response is returned) the conjugate query pair (K', X') is given to $\mathscr{A}$ for free as well. Furthermore, we again use the concept of a super query in the sense detailed and explained in the proof of Theorem 4.1.4. In the following, we divide the analysis into two where a preimage is found using a query from a super query or a normal query (here again the latter refers to a query that is not part of a super query).

First, we look at the event that a preimage for σ is found using a query that is not part of a super query. Assume, without loss of generality, that $\mathscr{A}$ makes a forward query (K,X) (the case for a decryption query follows analogously). Because C^{pre} is bijective, there is a unique (V,M) that corresponds to (K,X); moreover as C^{post} is bijective, for this particular (V,M) there exists a unique (Y,Y') that leads

to a preimage. As (by assumption) (K, X) is not part of a super query, we know that there exist at most $2^{n-1}-2$ (taking into account also the free conjugates) queries that have been asked by $\mathscr{A}$ using K; similarly, the case for the queries with key $\xi(K) = K'$ is at most $2^{n-1}-1$. Therefore, there are at most

$$(2^n - (2^{n-1}-2))(2^n - (2^{n-1}-1)) > 2^{2(n-1)}$$

possible outcomes for the queries (K, X) and (K', X'). So, the probability of finding a preimage with this query pair is at most $1/2^{2(n-1)}$; considering the conjugate pair, along with a union bound, we obtain an upper bound of

$$\frac{2q}{2^{2(n-1)}} = \frac{8q}{2^{2n}}$$

for the probability of finding a preimage with a query that is not a part of super query.

Second, let us study the case for queries that are part of a super query. Consider the super query corresponding to the key K. Necessarily (because conjugate queries come in pairs and there exists a permutation ξ such that $\xi(K) = K'$), we assume that 2^{n-1} queries (paired in 2^{n-2} conjugates) have already been asked by $\mathscr{A}$ using K and K'. Moreover, we observe that $\mathscr{A}$ can construct a set of super queries of cardinality at most $q/2^{n-2}$.

Let (K, X) and (K', X') be a part of the super query with corresponding compression function inputs (V, M) and (V', M'), respectively. Now we set up some notation: We define $R_{\mathsf{post}}(V, M, \cdot, Y')$, $R_{\mathsf{post}}(V, M, Y, \cdot)$, $R_{\mathsf{post}}(V', M', \cdot, Y)$ and $R_{\mathsf{post}}(V', M', Y', \cdot)$ to be the effective ranges of $C^{\mathsf{post}}(V, M, \cdot, Y')$, $C^{\mathsf{post}}(V, M, Y, \cdot)$, $C^{\mathsf{post}}(V', M', \cdot, Y)$ and $C^{\mathsf{post}}(V', M', Y', \cdot)$, respectively. Clearly, we end up with a preimage if there exist conjugate query pairs to the underlying blockcipher such that

$$\sigma \in R_{\mathsf{post}}(V, M, \cdot, Y') \cap R_{\mathsf{post}}(V, M, Y, \cdot) \quad \text{or} \quad \sigma \in R_{\mathsf{post}}(V', M', \cdot, Y) \cap R_{\mathsf{post}}(V', M', Y', \cdot)\,.$$

Because the output of the query (K, X) is returned uniformly at random, the probability of the event that $\sigma \in R_{\mathsf{post}}(V, M, \cdot, Y')$ or $\sigma \in R_{\mathsf{post}}(V', M', \cdot, Y)$ is at most $2/2^{n-1}$. Now conditioned on the above event, the probability of obtaining $\sigma \in R_{\mathsf{post}}(V', M', \cdot, Y)$ or $\sigma \in R_{\mathsf{post}}(V', M', Y', \cdot)$ is at most $2/(2^{n-1}-1)$. Hence, by Bayes' rule, the probability that (V, M) or (V', M') is a preimage is at most

$$\frac{4}{2^{n-1}(2^{n-1}-1)}\,.$$

A first union bound over 2^{n-2} pairs, followed by a second over $q/2^{n-2}$ super queries gives an upper bound of

$$\frac{4q}{2^{n-1}(2^{n-1}-1)}$$

for finding a preimage with a query that is part of a super query. Summing up the obtained probabilities completes the proof. □

4.3 Towards Close-to-Optimal Collision Resistance in the Iteration

We have seen how to construct collision-resistant compression functions by using two blockcipher calls. In this section, our goal is to consider security in the iteration for the same class (without imposing high-level security in the compression function level). Our hope is that this will lead to new schemes with less overhead than existing options.

In the definition below we define DBL Type-II compression functions where we do not allow explicitly a feedforward from the input of the compression function to the postprocessing (thus making the Type-II schemes disjunct from the Type-I schemes). From a practical point of view, not having a feedforward is certainly good news. Indeed, as exploited by Bogdanov et al. [33], feedforward has a considerable cost in hardware implementations and removing it altogether leads to significantly more efficient schemes. To illustrate, in [33] Bogdanov et al. show that when Hirose's scheme [76] is instantiated with the lightweight blockcipher Present-128, feedforward takes up to 20% of the overall area requirements, in a hardware implementation.

The bound we give on the collision resistance is still somewhat removed from optimal: It only proves collision resistance up to roughly $2^n/n$ queries. Moreover for a smaller number of queries, the advantage is too high. We do not believe that this is a problem inherent with the scheme at hand, but rather a shortcoming of the current proof and testament to the complications that arise concerning security in the iteration.

Definition 4.3.1. A double-call DBL blockcipher-based compression function $\mathsf{h}^{E^1,E^2} : \{0,1\}^{3n} \to \{0,1\}^{2n}$ (Definition 2.3.8) with $\kappa = s = 2n$ and $m = n$ is called DBL Type-II if and only if the following three conditions hold:

1. $C_1^{\mathsf{pre}} : \{0,1\}^{3n} \to \{0,1\}^{3n}$ and $C_2^{\mathsf{pre}} : \{0,1\}^{3n} \to \{0,1\}^{3n}$ are both bijections;
2. $C^{\mathsf{post}}(y_1, y_2)$ (the map $C^{\mathsf{post}} : \{0,1\}^{2n} \to \{0,1\}^{2n}$ that takes (y_1, y_2) to the Z) is a bijection and independent of V and M;
3. $C_1^{-\mathsf{pre}}(k_1, \cdot)$ and $C_2^{-\mathsf{pre}}(k_2, \cdot)$ are injections when restricted to V with effective ranges $R_{-\mathsf{pre}}(k_1, \cdot)$ and $R_{-\mathsf{pre}}(k_2, \cdot)$, respectively. Assume that there are at most 2^n different effective ranges (when considering different keys). Note that $C_1^{-\mathsf{pre}}(k_1, \cdot)$ and $C_2^{-\mathsf{pre}}(k_2, \cdot)$ are maps from n bits to n bits; therefore, there exist at most 2^n different effective ranges although there exist 2^{2n} such maps.

The following theorem illustrates that the collisions and preimages for the DBL Type-II compression functions can be found after roughly $2^{n/2}$ and 2^n queries, respectively; hence the compression functions in question are not optimal. Moreover, preimages for the iterated hash function constructed with DBL Type-II compression functions can be found after $2^{3n/2}$ queries, which is suboptimal for a DBL hash function.

Theorem 4.3.2. *Let* h^{E_1,E_2} *be a DBL Type-II compression function (Definition 4.3.1). Then collisions and preimages for* h^{E_1,E_2} *can be found (asymptotically in n) with* $\mathcal{O}(2^{n/2})$ *and* $\mathcal{O}(2^n)$ *queries, respectively. Preimages for the (Merkle–Damgård) iterated hash function constructed with* h^{E_1,E_2} *can be found (asymptotically in n) with* $\mathcal{O}(2^{3n/2})$ *queries.*

Proof. We first describe the collision attack on h^{E_1,E_2}; a very similar preimage attack on h^{E_1,E_2} and its extended version (to the iterated hash function) follow later. Given the DBL Type-II compression function h^{E_1,E_2}, consider the following collision-finding strategy: Let an adversary $\mathscr{A}$ first fix the ciphertext y_1; this limits the cardinality of the effective range $R_{\mathsf{post}}(y_1, \cdot)$ of $C^{\mathsf{post}}(y_1, \cdot)$ to 2^n. Now, $\mathscr{A}$ makes successive E^1 queries (k_1^*, y_1) by only changing the k_1^* component. Each fresh answer x_1^*, which satisfies $E^1_{k_1^*}(x_1^*) = y_1$, defines the unique compression function input (V^*, M^*), as well as the E^2 inputs (k_2^*, x_2^*). As the corresponding E^2 output y_2^* is returned uniformly random over a set of size at most 2^n, the compression function output $Z^* = C^{\mathsf{post}}(y_1, y_2^*)$ will be a uniform element from $R_{\mathsf{post}}(y_1, \cdot)$. Therefore after $2^{n/2}$ iterations, the collisions are present by the birthday paradox

with high probability. In addition, because C_1^{pre} is bijective, each different k_1^* defines a different compression function input (V^*, M^*). Furthermore, as C_2^{pre} is bijective, each different (V^*, M^*) pair defines a different E^2 input (k_2^*, x_2^*). Hence, the collision is output with reasonable probability.

Preimage attack goes along similar lines; let the target for which the preimage is to be found be σ. We first note that because C^{post} is bijective, there is a unique (y_1, y_2) that corresponds to σ. Let a preimage-finding adversary fix y_1 and follow the same strategy as in the collision attack with as sole exception that this time the goal is to hit the specific y_2, rather than finding collisions. As E^2 outputs are uniformly random over a set of size at most 2^n, after 2^n queries we hit the desired y_2 hence obtain the preimage with high probability.

Finally, we describe our preimage attack on the iterated hash function. Here, we do not consider any specific padding scheme; the attack described here should be extended to the hash function constructed using any concrete padding scheme (e.g., Merkle's padding as given in Section 2.2.1). The only difference between the attack to be presented here and the one with a concrete padding scheme is that a preimage needs to be found for the compression function somewhere in the iteration; as we show above that the preimages can be found for the compression function with $\mathcal{O}(2^n)$ queries, the overall query-complexity of our attack would not change.

Let the target digest Z be given. We use the same notation as in Definition 2.2.1. Our aim is to find $\mathsf{M} = (M_1, \ldots, M_4)$ for $M_i \in \{0,1\}^n$ and $i = 1,2,3,4$ such that $\mathsf{H}^{\mathsf{h}}(\mathsf{M}) = v_4 = Z$. The attack starts with inverting v_4 under h^{E_1,E_2}. In this step, we call our preimage-finding algorithm (given above) against h^{E_1,E_2} as a subroutine; yet with a minor modification. Our goal is now to find $2^{n/2}$ preimages, rather than only one. Iterating a single step of the above preimage-finding algorithm against the compression function $2^{3n/2}$ times instead of 2^n, we achieve the desired number of preimages. Note again that as C_1^{pre} is bijective we obtain different preimages. Moreover, we have enough freedom to mount the attack as $2^{3n/2} < 2^{2n}$. Therefore, with $2^{3n/2}$ queries, we can generate $2^{n/2}$ preimages (v_3, M_4) such that $\mathsf{h}^{E_1,E_2}(v_3, M_4) = v_4 = \sigma$.

Next, starting from any given IV, our aim is to hit one of v_3 values generated in the previous step. Observe that we can generate at most 2^n digests at each iteration as the message length is n. We suggest to proceed in the following way: Pick $2^{n/2}$ random M_i for each $i = 1,2,3$, iterate the compression function accordingly with a given IV and look for a correspondence with the existing v_3 values. Note that we can generate $2^{3n/2}$ such forward states v_3 and there already exist $2^{n/2}$ of those that were generated in the first step. Moreover, because E^1 and E^2 are chosen to be ideal and C^{post} is bijective, all the fresh forward states v_3 are uniform over a set of size at most 2^{2n}. Thus, the probability of hitting a single target v_3 value with a forward node generated in this step is at least 2^{-2n}; as there exist $2^{n/2}$ targets and $2^{3n/2}$ forward nodes, we expect to end up with a single match. Hence, the claim follows. □

Theorem 4.3.3. *Let* h^{E_1,E_2} *be a DBL Type-II compression function (Definition 4.3.1). Then the advantage of an adversary in finding a collision in the (Merkle and Damgård) iterated hash function* H^{h} *after* q *queries is upper bounded by*

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(q) \leq \frac{2\kappa q}{(2^n - q)} + \frac{q^2}{(2^n - q)^2} + 2^{n+1}\left(\frac{2^n q}{(2^n - q)^2}\right)^{\kappa+1}$$

for any positive integer κ.

Proof. As with [27], we only deal with message blocks of length n; the messages that are not multiples of n can be dealt with using standard padding techniques. As in the proof of Theorem 4.1.2, we consider a derived adversary only that makes corresponding queries to E^1 and E^2 in conjunction (as C_1^{pre} and C_2^{pre} are bijections as usual). Thus, the derived adversary either makes a query pair consisting of two forward queries, or a mixed pair consisting of an inverse query followed by a forward query.

We define a directed graph $G = (V_G, E_G)$ with vertex set $V_G = \{0,1\}^{2n}$, corresponding to all 2^{2n} possible chaining values, and initially an empty edge set $E_G = \emptyset$. We dynamically add edges based on the queries to E^1,E^2 and D^1, D^2. In particular, we add an arc (V, Z), labeled by M, if we know a message M such that $Z = \mathsf{h}^{E^1,E^2}(V, M)$ and the relevant queries to two primitives E^1 and E^2 have been made (where the decryption queries to D^1 and D^2 are counted for their respective encryption primitive). If an arc is added as a consequence of a forward query pair, then the node corresponding to Z becomes a forward node. If an arc is added as a consequence of a mixed query pair, both nodes become mixed nodes. Initially none of the nodes has a status except for the initial vector, which is treated as a forward node.

Our first observation is that in order to find a collision, we require the construction of a "ρ-shape": It occurs only if a cycle is found within the IV component or the IV component is connected to a cycle. Either way, one of the following three (bad) events needs to occur (on the i'th query pair, for some i):

1. A forward query pair leads to a digest that is already a forward node (i.e., previously resulted as digest of a forward query pair, or equals the initial value).
2. A forward query pair leads to a digest that is already a mixed node (i.e., previously resulted as digest or chaining variable of a mixed query pair).
3. A mixed query pair leads to a digest or chaining variable that is already a forward node (i.e., previously resulted as digest of a forward query pair, or equals the initial value).

In particular, we can ignore the fact that a mixed query pair leads to a digest or chaining variable that is already a mixed node. To see this, first observe that these are all the four possibilities for constructing longer chains within the graph. Secondly, by allowing an arbitrary number of occurrences of *only* the fourth (discarded) event, a collision can never be found for the simple reason that the initial vector is not part of the chain (because hitting the initial vector with a mixed query would have triggered the third event). We now bound the probability of any of the above bad events occurring on the i'th query pair.

1. Consider a forward query pair to E^1 and E^2, corresponding to a unique pair (V, M). Suppose that so far $t_1 \leq i$ queries to E^1 have been made involving key k_1. The answer y_1 to a fresh query to $E^1_{k_1}(\cdot)$ is therefore distributed uniformly over a set of $2^n - t_1$ possible outcomes. Similarly, suppose that $t_2 \leq i$ queries to E^2 have been made involving key k_2, so that the answer y_2 to $E^2_{k_2}(\cdot)$ is uniform over a set of $2^n - t_2$. Moreover, y_1 and y_2 are independent, so that the probability that (y_1, y_2) hits any particular value in $\{0,1\}^{2n}$ is at most $1/(2^n - i)^2$. As C^{post} is bijective when the pair (V, M) is fixed, the probability that any particular digest Z is hit is at most $1/(2^n - i)^2$. There are at most i possible Z-values labeled as a forward node, therefore, the probability of the i'th forward pair prompting this bad event is at most $i/(2^n - i)^2$.

2. Again, consider a forward query pair to E^1 and E^2, corresponding to a unique pair (V, M). As there are at most $2(i-1)$ nodes labeled as mixed node at this point, the probability of the i'th forward pair

prompting this bad event is at most $2(i-1)/(2^n-i)^2$.

3. Finally, consider a mixed query pair. We assume it consists of a D^1-query followed by an E^2-query, the other case is analogous. This time we can argue that x_1 and y_2 are both distributed uniformly and independently over sets of cardinality at least $2^n - i$. Consequently, the same holds for chaining variable V and digest Z. As there are at most i forward nodes that can be hit, this gives (with a union bound) a probability of success upper bounded by $2i/(2^n-i)$. Unfortunately, this upper bound is too large to be useful.

Although there are up to i forward nodes, not all of them are relevant. Indeed, the query to D^1 essentially fixes k_1 and y_1. Because C^{post} is bijective, this leaves only 2^n possible digests. Similarly, as $C_1^{-\mathsf{pre}}$ is injective, it leaves only 2^n possible chaining variables. Thus, we only need to consider the number of forward nodes in the effective range of C^{post} and $C_1^{-\mathsf{pre}}$, respectively after fixing (k_1, y_1). Let this number be κ (maximum for C^{post} or $C_1^{-\mathsf{pre}}$), then the probability of the bad event occurring (on the i'th mixed query) is at most $2\kappa/(2^n-i)$.

The question is whether good bounds on the number κ of relevant forward nodes can be found. Let us fix some k_1 and consider the probability that a forward query lands in $R_{-\mathsf{pre}}(k_1,\cdot)$. We have already seen that a forward query leads to an almost uniform distribution of the resulting digest and we know that $R_{-\mathsf{pre}}(k_1,\cdot)$ has cardinality 2^n. Hence if the total number of forward query pairs is q, then each query pair has a probability at most $2^n/(2^n-q)^2$ to hit our set (effective range). Now, writing $B\left[Q;p\right]$ for the random variable counting the number of successes in Q independent Bernoulli trials, each with success probability p, a Chernoff bound can be used to bound the tail probability for any $\kappa > Qp$, namely

$$\Pr\left[B\left[Q;p\right] > \kappa\right] < \left(\frac{epQ}{\kappa}\right)^{\kappa}.$$

So, with a Chernoff bound, we have that for any given set the probability it is hit more than κ times is less than

$$\left(\frac{e2^n q}{\kappa(2^n-q)^2}\right)^{\kappa}.$$

Note that we can obtain a similar bound if we use Proposition 3.3.10 as well. Using the terminology and the notation of Proposition 3.3.10, define (for D^1 query/response pairs (X^i, Y^i))

$$\mathsf{P}^{D^1}_{\mathsf{hit}_i|X^iY^{i-1}} = \frac{2^n}{(2^n-q)^2} \quad \text{and} \quad B_\Sigma = \frac{2^n q}{(2^n-q)^2}$$

and observe that D^1 can be defined as a random system as it is simply a uniformly random permutation for any fixed key. It is not difficult to see that the hypotheses of Proposition 3.3.10 are satisfied for values of q for which $B_\Sigma < 1$ (otherwise the bound is too loose to be useful). Then, we achieve that for any given set the probability that we have more than κ hits is at most $B_\Sigma^{\kappa+1}$. As there are at most 2^{n+1} different sets of size 2^n (2^n for each of the effective ranges of C^{post} and $C_1^{-\mathsf{pre}}$) to consider we can apply a union bound, resulting in an extra factor of 2^{n+1}. Summing up the various probabilities proves the theorem statement. □

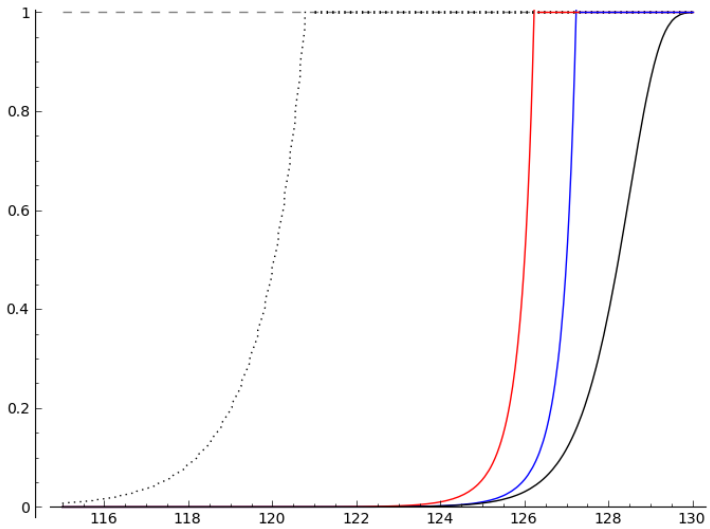

Figure 4.3 – Collision resistance bounds illustrated for $n = 128$. The horizontal axis is $\log_2(q)$ and vertical axis is $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(q)$. The black, solid curve is the birthday bound; the dotted curve is the bound obtained from Theorem 4.3.3 (with an optimization for κ). Red and blue curves are the bounds from Theorem 4.2.2 and 4.1.2, respectively; note that we use Theorem 2.3.4 to derive the corresponding bounds for the hash function.

Optimization of the Bound and Comparison

Note that the bound given in Theorem 4.3.3 is dependent on an integer κ that can be chosen arbitrarily. Here we deal with how to choose the value of κ so that the overall upper bound gets optimized; furthermore, we compare the resulting upper bound with the previous collision resistance bounds given in this chapter. We start with the following observation: Once the terms containing κ are chosen to be close-to-equal, then the overall bound becomes close-to-optimal. Moreover, we have $q \leq 2^{n-1}$; hence $2^n - q \geq 2^{n-1}$. Therefore, the overall bound becomes:

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(q) \leq \frac{2\kappa q}{2^{n-1}} + \left(\frac{q}{2^{n-1}}\right)^2 + 2^{n+1}\left(\frac{e2^n q}{\kappa(2^{n-1})^2}\right)^{\kappa} .$$

We simply look at the equation:

$$\frac{2\kappa q}{2^{n-1}} = 2^{n+1}\left(\frac{e2^n q}{\kappa(2^{n-1})^2}\right)^{\kappa} .$$

Solving for κ in turn gives

$$\kappa = \frac{2n+1-\log_2(q)-\log_2(\kappa)}{n-2-\log_2(q)-\log_2(e)+\log_2(\kappa)} \leq \frac{2n+1-\log_2(q)}{n-2-\log_2(q)-\log_2(e)} = \bar{\kappa} .$$

We pick $\kappa = \lceil \bar{\kappa} \rceil$ for our bound. In Figure 4.3, we illustrate, for $n = 128$, various bounds we obtained for $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(q)$ as a function of $\log_2(q)$, when the iterated hash function is constructed using DBL Type-I , DBL Type-II and IDS DBL Type-I compression functions. The bounds given for DBL Type-I and IDS DBL Type-I compression functions (in Theorem 4.1.2 and 4.2.2, respectively) are rather straightforward to compute (along with a standard use of Theorem 2.3.4); for the bound given in Theorem 4.3.3, we optimize the bound by finding an appropriate κ as detailed above. Consequently, the bounds obtained in Theorem 4.1.2 and 4.2.2 are very close to the birthday bound, whereas the

security in the iteration for a DBL Type-II compression function can only be guaranteed up to roughly $2^n/n$ queries.

4.4 Implications for Linear Schemes

In this section, we investigate the implications of our results, on the designs with $\mathbb{F}_2$-'block'-linear instances of C^{pre} and C^{post}, by following the analogue of the analysis by Stam [191] on single-call blockcipher-based constructions. Our focus is again on the double-call DBL blockcipher-based compression functions (Definition 2.3.8) with parameters $\kappa = s = 2n$ and $m = n$. The reason for restricting ourselves to $\mathbb{F}_2$-'block'-linear instances is mainly efficiency related: $\mathbb{F}_2$-'block'-linear C^{pre} and C^{post} result in relatively simple and fast implementations. Moreover, when chosen properly, it is possible to profit from formal security proofs. PGV compression functions (see Section 2.3.3) for the single-block-length case, as well as many suggested double-block-length compression functions are of this type (see below for examples with the parameters in question).

In Section 4.4.1, we concern ourselves with compression functions with two independent blockciphers. In Section 4.4.2, we revisit the well-known construction of Hirose [76] who suggests a scheme with a single blockcipher; and finally in Section 4.4.3, we look at the constructions that are collision resistant when iterated.

4.4.1 Secure Compression Functions with Distinct and Independent Blockciphers

Here our focus is on the secure compression functions that use two distinct and independently sampled blockciphers, along with $\mathbb{F}_2$-'block'-linear pre and postprocessing functions. Hirose [75] suggests a method to construct DBL Type-I compression functions that are optimally collision resistant with $\mathbb{F}_2$-'block'-linear instances of C^{pre} and C^{post} and that are based on two different blockciphers. Our constructions are a generalization of his work, and indeed, when we restrict ourselves to $\mathbb{F}_2$-'block'-linear constructions without output mixing ($\mathbf{T}_1 = (10)$ and $\mathbf{T}_2 = (01)$ in the notation below), we get exactly the same set of schemes as Hirose does.

Let us first set up some notation. We treat V and Z as $V = (v_1||v_2)$ and $Z = (z_1||z_2)$ where $v_i, z_i \in \{0,1\}^n$ for $i = 1,2$; and we represent the schemes using matrices. We use $\mathbb{Z}_2^{3\times 3}$ to express the way (k_i, x_i) are functions of M and (v_1, v_2) for $i \in \{1,2\}$. Thus C_1^{pre} is represented by matrix $\binom{\mathbf{K}_1}{\mathbf{X}_1}$ with $\mathbf{K}_1 \in \mathbb{Z}_2^{2\times 3}$ and $\mathbf{X}_1 \in \mathbb{Z}_2^{1\times 3}$. The vector $\mathbf{X}_1 \in \mathbb{Z}_2^{1\times 3}$ corresponds to

$$x_1 = \mathbf{X}_1 \cdot (M, v_1, v_2)^T ,$$

making a distinction between the linear map $\mathbf{X}_1 \in \mathbb{Z}_2^{1\times 3}$ and the value $x_1 \in \{0,1\}^n$. Similarly, $\mathbf{K}_1 \in \mathbb{Z}_2^{2\times 3}$ corresponds to

$$k_1 = \mathbf{K}_1 \cdot (M, v_1, v_2)^T .$$

We represent C_2^{pre} analogously by $\mathbf{K}_2$ and $\mathbf{X}_2$. Postprocessing function C^{post} is given by matrices

$$\begin{pmatrix}\mathbf{T}_1\\ \mathbf{T}_2\end{pmatrix} \in \mathbb{Z}_2^{2\times 2} \quad \text{and} \quad \begin{pmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{pmatrix} \in \mathbb{Z}_2^{2\times 3} .$$

The compression function output is then computed as follows:

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \oplus \begin{pmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ M \end{pmatrix} .$$

We are now ready to see what the requirements from Definition 4.1.1 mean in terms of $\mathbf{K}_i, \mathbf{X}_i, \mathbf{U}_i$ and $\mathbf{T}_i$ and hence for the classification and security of Hirose's schemes.

Corollary 4.4.1. *An $\mathbb{F}_2$-'block'-linear DBL blockcipher-based compression function $\mathsf{h}^{E^1,E^2} : \{0,1\}^{3n} \to \{0,1\}^{2n}$ is DBL Type-I (Definition 4.1.1) if and only if*

1. $\left(\begin{smallmatrix} \mathbf{K}_1 \\ \mathbf{X}_1 \end{smallmatrix}\right)$ *and* $\left(\begin{smallmatrix} \mathbf{K}_2 \\ \mathbf{X}_2 \end{smallmatrix}\right)$ *are both invertible matrices.*
2. $\left(\begin{smallmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{smallmatrix}\right)$ *is an invertible matrix.*
3. $\left(\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} \mathbf{K}_1 \\ \mathbf{X}_1 \end{smallmatrix}\right)^{-1} \left(\begin{smallmatrix} k_1 \\ x_1 \end{smallmatrix}\right) \oplus \left(\begin{smallmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right)$ *is bijective as a function of x_1 and y_2 for all k_1 and y_1.*
4. $\left(\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} \mathbf{K}_2 \\ \mathbf{X}_2 \end{smallmatrix}\right)^{-1} \left(\begin{smallmatrix} k_2 \\ x_2 \end{smallmatrix}\right) \oplus \left(\begin{smallmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right)$ *is bijective as a function of x_2 and y_1 for all k_2 and y_2.*

Proof. We check the three conditions in Definition 4.1.1. First, C_1^{pre} and C_2^{pre} are bijections. This is equivalent to $\left(\begin{smallmatrix} \mathbf{K}_1 \\ \mathbf{X}_1 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} \mathbf{K}_2 \\ \mathbf{X}_2 \end{smallmatrix}\right)$ being invertible. The second condition is that $C^{\mathsf{post}}(M, V, \cdot, \cdot)$ is a bijection (from $2n$ to $2n$ bits, for all (M, V)) which is the case if and only if $\left(\begin{smallmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{smallmatrix}\right)$ is invertible. Finally, we should have C_1^{aux} and C_2^{aux} are bijections for all k_1, y_1 and k_2, y_2, respectively, which is simply rephrased to the linear case in the third and fourth requirements from the corollary. □

Tweaked Abreast-DM To make the discussion above more concrete, we consider a tweaked version of Abreast-DM, dubbed *Abreast-DM-t*, that can be proven collision resistant using our framework (it was also considered by Hirose [75]). Abreast-DM (see Figure 4.4) and its sister design Tandem-DM, both proposed in the early 1990s [102] by Lai and Massey, are two of the classic examples of double-block-length compression (and hash) function designs. In [102], Lai and Massey conjecture that Abreast-DM (as well as Tandem-DM) is optimally collision and preimage resistant as they could not find a successful attack. Later, it was proved that the claim of Lai and Massey is correct: An almost optimal collision resistance for Abreast-DM has been proved recently in [64, 104]. Preimage resistance, however, has been studied by Armknecht et al. [8] and similarly shown to be close-to-optimal. We show, using Theorem 4.1.2 and Corollary 4.4.1, that *Abreast-DM-t* enjoys similar collision resistance [4].

As it stands, Abreast-DM does not fit the framework above as it uses the same blockcipher twice. Let us therefore define the tweaked version Abreast-DM-t as follows:

$$z_1 = v_1 \oplus E^1_{v_2 \| M}(v_1) \quad \text{and} \quad z_2 = v_2 \oplus E^2_{M \| v_1}(v_2) .$$

4. We note that the application of Theorem 4.1.4 is not immediate as *Abreast-DM-t* does not satisfy the additional requirement of Theorem 4.1.4. Nevertheless, the preimage resistance proof of Armknecht et al. on Abreast-DM should be applicable to *Abreast-DM-t* as well.

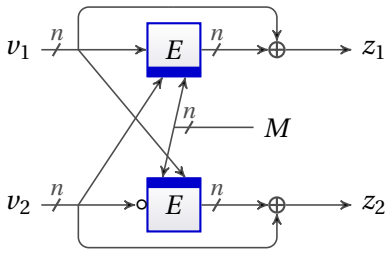

Figure 4.4 – The Abreast-DM compression function illustrated where $\circ$ denotes the bitwise complementation. Abreast-DM-t makes use of two distinct and independently sampled blockciphers and omits bitwise complementation.

According to the notation introduced above, we have

$$\mathbf{K}_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{X}_1 = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}, \quad \mathbf{U}_1 = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}, \quad \mathbf{T}_1 = \begin{pmatrix} 1 & 0 \end{pmatrix},$$
$$\mathbf{K}_2 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \quad \mathbf{X}_2 = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}, \quad \mathbf{U}_2 = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}, \quad \mathbf{T}_2 = \begin{pmatrix} 0 & 1 \end{pmatrix}.$$

So, $\binom{\mathbf{K}_1}{\mathbf{X}_1}$, $\binom{\mathbf{K}_2}{\mathbf{X}_2}$, and $\binom{\mathbf{T}_1}{\mathbf{T}_2}$ are all invertible matrices. For the third condition, we obtain that

$$\begin{pmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{K}_1 \\ \mathbf{X}_1 \end{pmatrix}^{-1} \begin{pmatrix} k_1 \\ x_1 \end{pmatrix} \oplus \begin{pmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} k_1^1 \\ k_1^2 \\ x_1 \end{pmatrix} \oplus \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 \oplus y_1 \\ k_1^1 \oplus y_2 \end{pmatrix},$$

is bijective as a function of x_1 and y_2 for all k_1 and y_1. Similarly, the fourth condition is verified:

$$\begin{pmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{K}_2 \\ \mathbf{X}_2 \end{pmatrix}^{-1} \begin{pmatrix} k_2 \\ x_2 \end{pmatrix} \oplus \begin{pmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} k_2^1 \\ k_2^2 \\ x_2 \end{pmatrix} \oplus \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} k_2^2 \oplus y_1 \\ x_2 \oplus y_2 \end{pmatrix}.$$

Therefore, Abreast-DM-t satisfies the requirements of Corollary 4.4.1 and the bounds from Theorems 4.1.2 and 4.1.3 apply.

A Simpler Proposal Now we consider a simpler proposal; simpler in the sense that it requires the keys of the underlying blockciphers to be the same (which might result in a more efficient scheme in practice). Moreover, it satisfies the requirements of both Theorem 4.1.2 and Theorem 4.1.4. Therefore, almost optimal collision and preimage resistance are guaranteed. Here is the proposal:

$$z_1 = v_1 \oplus E^1_{v_2 \| M}(v_1) \quad \text{and} \quad z_2 = v_1 \oplus E^2_{v_2 \| M}(v_1)\,.$$

Using the notation introduced above, we have

$$\mathbf{K}_1 = \mathbf{K}_2 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \mathbf{X}_1 = \mathbf{X}_2 = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}, \mathbf{U}_1 = \mathbf{U}_2 = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

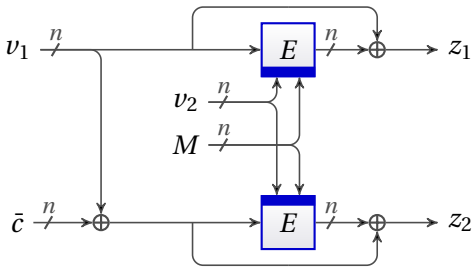

Figure 4.5 – The Hirose's DBL compression function illustrated where $\bar{c} \in \{0,1\}^n \setminus \{0\}^n$.

It is easy to check that the hypotheses of Corollary 4.4.1 are satisfied. Moreover, because the keys of the underlying blockciphers are the same, Theorem 4.1.4 applies automatically.

4.4.2 Using a Single Blockcipher

Abreast-DM suffers from a performance drawback that, although run in parallel, the underlying blockciphers require two separate key-schedule routines. Hirose's construction [76] overcomes this problem by sharing the key scheduling for the two blockcipher calls (see Fig. 4.5). Even nicer, the design also enjoys almost optimal collision resistance (recently, it has been proved by Armknecht et al. [8] that the construction has close-to-optimal preimage resistance as well).

Hirose's scheme is one of the existing schemes which is covered by Definition 4.2.1 (and a suitable adaptation of Corollary 4.4.1), with a minor modification. We first recall the scheme (see Figure 4.5):

$$z_1 = v_1 \oplus E_{v_2 \| M}(v_1) \quad \text{and} \quad z_2 = v_1 \oplus \bar{c} \oplus E_{v_2 \| M}(v_1 \oplus \bar{c}) \,,$$

where $\bar{c}$ is a non-zero constant in $\{0,1\}^n$. Let us check the requirements given in Definition 4.2.1 with

$$p(v_2 \| M, v_1) = (v_2 \| M, v_1 \oplus \bar{c}) \,.$$

Clearly, p is an involution without fixed points because $\bar{c} \in \{0,1\}^n \setminus \{0\}^n$ (hence the requirement 4. from Definition 4.2.1 holds). Moreover, C_1^{pre} is a bijection as it is simply the identity map; for C_2^{pre}, we have

$$C_2^{\mathsf{pre}}(M, v_1 \| v_2) = (M, v_1 \oplus \bar{c} \| v_2)$$

which is indeed a bijection. Finally, it is easy to see that both

$$C_1^{\mathsf{aux}}(x_1, y_2) = \left(x_1 \oplus y_1 \| x_1 \oplus \bar{c} \oplus y_2\right) \quad \text{and} \quad C_2^{\mathsf{aux}}(x_2, y_1) = \left(x_2 \oplus \bar{c} \oplus y_1 \| x_2 \oplus y_2\right)$$

are bijections (for all y_1, k_1 and y_2, k_2, respectively). We note that requirement 5. from Definition 4.2.1 is not satisfied. Specifically, the conjugate pairs can collide once the following holds:

$$v_1 \oplus E_{v_2 \| M}(v_1) = v_1 \oplus \bar{c} \oplus E_{v_2 \| M}(v_1 \oplus \bar{c}) \,,$$

which implies

$$E_{v_2 \| M}(v_1) \oplus E_{v_2 \| M}(v_1 \oplus \bar{c}) = \bar{c} \,.$$

The probability of the above event happening is low and the proof given in Theorem 4.2.2 can be adjusted by taking into account this extra case without badly affecting the upper bound; indeed, the proof given in [76] goes along with this idea.

4.4.3 Collision Resistant Constructions in the Iteration

We now study, as a final category, the linear implications of our results to the case where the compression functions are not optimally secure but turn out to be close-to-optimal collision resistant when iterated. Using the same notation as above, we consider the compression functions of the following form (note that the feedforward from the input to the compression function to C^{post} is removed):

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix},$$

where $y_1 = E^1_{k_1}(x_1)$, $y_2 = E^2_{k_2}(x_2)$,

$$x_i = \mathbf{X}_i \cdot (M, v_1, v_2)^T \quad \text{and} \quad k_i = \mathbf{K}_i \cdot (M, v_1, v_2)^T,$$

for $i = 1,2$, where $\mathbf{X}_i \in \mathbb{Z}_2^{1\times 3}$ and $\mathbf{K}_i \in \mathbb{Z}_2^{2\times 3}$. The following corollary is the analogue of Corollary 4.4.1.

Corollary 4.4.2. *An $\mathbb{F}_2$-'block'-linear DBL blockcipher-based compression function $\mathsf{h}^{E^1,E^2} : \{0,1\}^{3n} \to \{0,1\}^{2n}$ is DBL Type-II if and only if*

1. $\binom{\mathbf{K}_1}{\mathbf{X}_1}$ *and* $\binom{\mathbf{K}_2}{\mathbf{X}_2}$ *are both invertible matrices.*
2. $\binom{\mathbf{T}_1}{\mathbf{T}_2}$ *is an invertible matrix.*
3. $\binom{\mathbf{K}_1}{\mathbf{X}_1}^{-1}\binom{k_1}{x_1}$ *and* $\binom{\mathbf{K}_2}{\mathbf{X}_2}^{-1}\binom{k_2}{x_2}$ *are injections when restricted to V, as a function of x_1 and x_2, respectively.*

Proof. The proof follows from the proof of Corollary 4.4.1. □

An Example Here is an example of a scheme that satisfies the requirements of Corollary 4.4.2:

$$z_1 = E^1_{v_2 \| M}(v_1) \quad \text{and} \quad z_2 = E^2_{v_2 \| M}(v_1).$$

Using the notation introduced above, we have

$$\mathbf{K}_1 = \mathbf{K}_2 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \mathbf{X}_1 = \mathbf{X}_2 = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Note that these are the matrices used for Hirose's construction; thus the first and the second requirement of Corollary 4.4.2 hold. The third requirement is easy to verify as both $\binom{\mathbf{K}_1}{\mathbf{X}_1}^{-1}\binom{k_1}{x_1}$ and $\binom{\mathbf{K}_2}{\mathbf{X}_2}^{-1}\binom{k_2}{x_2}$ result in v_1, which is injective when restricted to V.

5 A Compression Function Exploiting Discrete Geometry

In Chapters 2 and 3, we classified the multi-block-length hash function constructions based on the dimensions of their compression function and its underlying primitives, as well as on the number of (parallel) primitive calls made per compression function evaluation. In a more fine-grained way, we can also distinguish *independent* and *identical* primitives. For instance, if the independent PuRFs (public random functions) whose input size matches that of the construction are used, we could quite easily call the (distinct) PuRFs on the compression function input and concatenate the results to form the digest: In fact this gives a perfect compression function [1] (complications arise when switching to blockciphers, especially identical ones, but in the context of collision resistance these can be resolved, as shown in Chapter 4).

A more challenging scenario is where the domain of the primitive is much 'smaller' than the domain of the intended compression function, because the inevitable consequence is that any single primitive query can be relevant for multiple compression function evaluations. This leaves open the question of constructing a compression function that uses only a few smaller primitives. Of particular interest is the construction of a provably collision-resistant compression function from $3n$ to $2n$ bits that makes two parallel calls to an ideal primitive from $2n$ to n bits (either a public random function or an ideal blockcipher with n-bit blocks and n-bit keys).

This setting corresponds to the double-call DBL primitive-based compression function introduced in Definition 3.2.1 (see also Definition 2.3.8) with parameters [2] $t = 3$ and $c = r = s = 2$. All existing double-call DBL primitive-based compression function constructions, from the same class, fall short: A (non-trivial) collision resistance is only provided in the iteration; the primitive calls need to be made in sequence; or the number of calls is higher. Still, known bounds [174, 190, 193] (see Conjecture 2.3.5) give no reason that such a construction should not be possible.

In this chapter, we deal with this problem: We study the open question of constructing a double-call

1. This claim can be proved by using the indistinguishability framework of Maurer [119]: Intuitively, the idea is as follows. Assume the existence of an adversary that tries to distinguish the construction in question from a random function. Observe that each (different) compression input results in a uniformly random compression function output. This is because, in this setting, the inputs of the underlying PuRFs can be assigned in such a way that each query to the compression function assigns a different PuRF input. Consequently, the corresponding PuRF outputs are uniformly random for each new query; as is the compression function output. Hence, any adversary that tries to distinguish such a construction from a random function would have a negligible chance at each step that it makes a fresh query.

2. We note, however, that these parameters are not necessarily the only ones that might possibly provide a good collision resistance bound (cf. Conjecture 2.3.5).

DBL compression function $\mathsf{h} : \{0,1\}^{3n} \to \{0,1\}^{2n}$ (Definition 3.2.1) with $t = 3$ and $c = r = s = 2$ that guarantees a collision resistance bound beyond $2^{n/2}$ queries. We concentrate on the PuRF scenario; a detailed analysis for the blockcipher-based instantiation is planned to be carried out in the full-version of [81]. Our construction has two key innovative components: A preprocessing function $C^{\mathsf{pre}} : \mathbb{F}_{2^n}^3 \to \left(\mathbb{F}_{2^n}^2\right)^2$ that transforms the $3n$-bit input into a pair of $2n$-bit strings that are passed as inputs to the two ideal primitive calls; and a postprocessing function $C^{\mathsf{post}} : \mathbb{F}_{2^n}^5 \to \mathbb{F}_{2^n}^2$ that combines the two outputs of the ideal primitives and the $3n$-bit compression function input into the $2n$-bit output of the compression function.

A concept that plays a central role in our collision resistance analysis is the maximum number of possible compression function evaluations given specific query quota, also known as the yield (Definition 2.3.1). This concept is not new; it has been used in the analysis of several schemes [118, 173, 180, 186], as well as in the derivation of feasibility results [174, 190, 193]. The main idea of yield-based security analysis is that we can reduce the task of designing a new compression function to meeting two main challenges: (i) reducing the yield of an adversary, and (ii) ensuring that, regardless of the adversarial strategy, the resulting compression function evaluations are close-to-uniform. It will be seen that consequently a yield-optimizing adversary is close-to-optimal and, for collision resistance, bound to a yield-based birthday bound. In our construction, the preprocessing function C^{pre} takes care of (i), whereas the postprocessing function C^{post} ensures (ii).

The main innovation of our design is the choice for the preprocessing function C^{pre}: The $3n$-bit input is transformed into a pair of a line in $\mathbb{F}_{2^n}^2$ and a point on that line. Therefore, any given valid input pair to the underlying ideal primitives corresponds to an incidence between a point and a line in the affine plane $\mathbb{F}_{2^n}^2$ over the finite field $\mathbb{F}_{2^n}$; hence the yield is estimated by the number of such point-line incidences. We then use a classic result of discrete geometry, the Szemerédi–Trotter theorem over finite fields, to bound the number of incidences between a set of q lines and a set of q points on $\mathbb{F}_{2^n}^2$:

$$\mathrm{yield}(q) \leq q^{3/2} + q$$

(see Definition 2.3.1 for the definition of $\mathrm{yield}(q)$). The preprocessing function requires one finite field multiplication, together with an additional XOR; it turns out that to define the aforementioned lines (hence to derive the above upper bound on $\mathrm{yield}(q)$), multiplication is necessary; although it leads to a less efficient scheme compared to existing constructions with linear C^{pre}.

The postprocessing function is inspired by the Rogaway–Steinberger construction [173], where a special type of $\mathbb{F}_{2^n}$-linear map is used for C^{post}. However, we crucially add the product of the two primitive outputs to the inputs to this linear map. This turns out to be important [3] for our collision resistance proof, where we show that the best strategy for any collision-finding adversary is (close to) maximizing $\mathrm{yield}(q)$. Putting the pieces together, we show (in Corollary 5.1.7), for any $\delta > 0$ and $q \leq 2^{2n(1-\delta)/3}$, that (asymptotically in n) $\mathsf{Adv}_{\mathsf{h}}^{\mathsf{coll}}(q) = o(1)$.

Our collision resistance bound (Theorem 5.1.6) can be seen as a function on certain parameters that can be set to any positive integer; after an optimization over these integers, we achieve an optimized

3. We present a concrete collision-finding attack in Section 5.1.2 against the same compression that uses the multiplication-less version of C^{post}. Moreover, our security proofs require the product of the two primitive outputs to be added in C^{post}. For the record, we study in Section 5.1.3 some other more efficient non-linear postprocessing functions as alternatives to the product of the two primitive outputs; we show that many simpler choices fall short and explain the design rationale behind our choice.

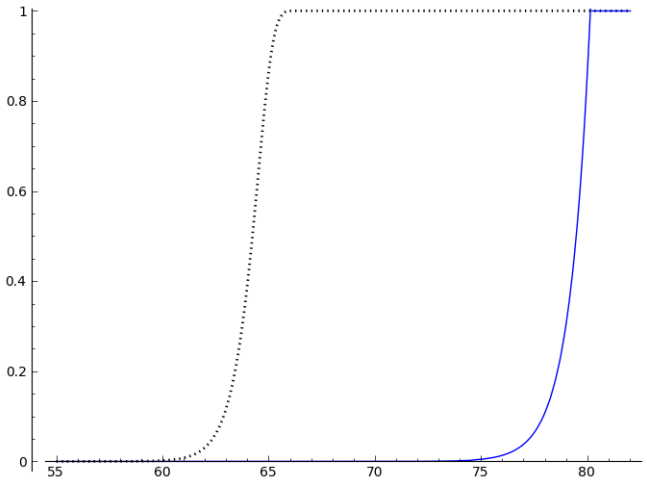

Figure 5.1 – Collision resistance bound (as a function of $\log_2(q)$) illustrated for $n = 128$. The vertical axis is $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q)$ and the horizontal axis is $\log_2(q)$. The dotted curve (shown in black) is the best known bound so far for the double-call DBL primitive-based compression functions in question; the solid curve (in blue) is the bound obtained from Theorem 4.3.3 (with an optimization over the constant values as detailed in Section 5.1.4 on page 97).

upper bound on $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q)$ for concrete values of the output length n of the underlying primitives. To demonstrate, for $n = 128$, we plot our bound on $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q)$ as a function of $\log_2(q)$ in Figure 5.1. There it is shown that in order to have a non-negligible advantage, any adversary needs to ask at least (roughly) 2^{80} queries to the underlying primitives to find a collision. To the best of our knowledge, this is the first construction of this type guaranteeing collision resistance beyond $2^{n/2}$ queries.

We also study (everywhere) preimage resistance; we show (in Corollary 5.1.9), for any $\delta > 0$ and $q \leq 2^{n(1-\delta)}$, that $\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q) = o(1)$. Additionally, we present a preimage attack that requires $\mathcal{O}(2^n)$ queries (and time). Again, using a similar optimization as done for our collision resistance proof for $n = 128$, we conclude that any adversary needs to ask at least (approximately) 2^{121} queries to the underlying primitives to find a preimage with high probability.

This chapter is organized as follows. In Section 5.1, we introduce our design, instantiated with two different PuRFs, as well as our security claims. It also contains our design rationale, together with some concrete attacks on somewhat weakened versions of our construction. In Sections 5.2 and 5.3, we present our proof of collision and (everywhere) preimage resistance, respectively . In Section 5.4, we discuss (without providing a proof) our construction when instantiated with two (distinct and independently sampled) blockciphers with n-bit blocks and n-bit keys. Finally, in Section 5.5, we conclude the chapter with some practical considerations and comparison with earlier work.

5.1 Our Construction and the Security Claims

5.1.1 The Design

In this section, we introduce our compression function $\mathsf{h}^{f^1,f^2}\colon \{0,1\}^{3n} \to \{0,1\}^{2n}$. It is a specific example of a double-call DBL PuRF-based compression function defined in Definition 3.2.1 with

parameters [4] $t = 3$ and $s = c = r = 2$. Therefore, the construction uses two parallel calls to independent PuRFs $f^1, f^2 \colon \{0,1\}^{2n} \to \{0,1\}^n$; moreover, the preprocessing functions C_1^{pre} and C_2^{pre} both map $3n$ bits to $2n$ bits, whereas the postprocessing function C^{post} maps $5n$ bits to $2n$ bits. Recall that C_1^{pre} and C_2^{pre} are the functions that take as input the compression function input and assign the inputs to the underlying PuRFs. C^{post}, however, is the postprocessing function that takes the input of the compression function as input, as well as the outputs of the underlying PuRFs, and produces the compression function output.

For notational convenience, we often write the input to the compression function $W \in \{0,1\}^{3n}$ as a triple $(a,b,c) \in (\{0,1\}^n)^3$ and identify $\{0,1\}^n$ with $\mathbb{F}_{2^n}$. Note that, in the following, we always treat the triple (a,b,c) in the same (alphabetic) order it is presented here. That is, we do not concern ourselves with triples (c,b,a), (b,a,c), etc. We present our concrete proposal in Construction 5.1.1 (see also Figure 5.2). Our security claims are given in Theorem 5.1.6 and 5.1.8 for collision and (everywhere) preimage resistance, respectively.

Construction 5.1.1. *Let $f^1, f^2 \colon \{0,1\}^{2n} \to \{0,1\}^n$ be two different and independently sampled PuRFs. Define a double-call DBL PuRF-based compression function (as in Definition 3.2.1 with parameters $t = 3$ and $c = r = s = 2$)*

$$\mathsf{h}^{f^1,f^2} \colon \{0,1\}^{3n} \to \{0,1\}^{2n}$$

by using the preprocessing function $C^{\mathsf{pre}} \colon \mathbb{F}_{2^n}^3 \to (\mathbb{F}_{2^n}^2)^2$ for

$$C^{\mathsf{pre}} = \left(C_1^{\mathsf{pre}}, C_2^{\mathsf{pre}}\right), \quad C_1^{\mathsf{pre}}(a,b,c) = (a,b) \quad \text{and} \quad C_2^{\mathsf{pre}}(a,b,c) = (c, ac+b);$$

and the postprocessing function $C^{\mathsf{post}} \colon \mathbb{F}_{2^n}^5 \to \mathbb{F}_{2^n}^2$

$$C^{\mathsf{post}}(a,b,c,y_1,y_2) = A \cdot \begin{pmatrix} a \\ c \\ y_1 \\ y_2 \\ y_1 y_2 \end{pmatrix}, \text{ where } \quad A = \begin{pmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \omega_{14} & \omega_{15} \\ \omega_{21} & \omega_{22} & \omega_{23} & \omega_{24} & \omega_{25} \end{pmatrix}$$

is a matrix (over $\mathbb{F}_{2^n}$) satisfying certain non-degeneracy conditions that are discussed below (see Table 5.1 on page 102 for a detailed explanations of the conditions on the entries of A).

Remark 5.1.2. We note that in C^{post} the input block shown by b is not used. The reason is that in our security proofs, we do not require b to be processed by C^{post}. This is similar to the well-known example of a single-block-length blockcipher-based compression function (Definition 2.3.6) Davies-Meyer, where the entire message block is not used in the corresponding C^{post} (see the compression function number 5 shown in Figure 2.3 on page 31).

In the security proofs, we abstract the properties required of the pre and postprocessing functions C^{pre} and C^{post}. In Section 5.1.2, we present several fairly generic attacks against somewhat simpler proposals, which indicates that we did not 'over-design' our proposal. We stress that in our security proofs, we do not focus on a special matrix A; on the contrary, we consider general A that satisfies certain non-degeneracy conditions as summarized in Table 5.1. Our security proofs are valid for any

4. A blockcipher analogue of this class of compression functions are introduced in Definition 2.3.8 with parameters $m = \kappa = n$ and $s = 2n$.

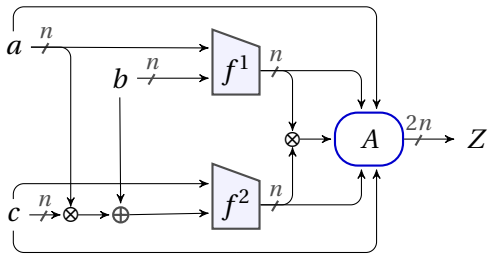

Figure 5.2 – Our compression function $\mathsf{h}^{f^1,f^2} : \{0,1\}^{3n} \to \{0,1\}^{2n}$ illustrated. The input W to h^{f^1,f^2} is represented as an element of $\mathbb{F}_{2^n}^3$, i.e., $W = (a,b,c) \in \mathbb{F}_{2^n}^3$ for $a,b,c \in \mathbb{F}_{2^n}$. The matrix A used in the postprocessing function C^{post} is specified in Section 5.1 (Construction 5.1.1) along with the conditions on its entries in Table 5.1 on page 102. In the figure, $\otimes$ and $\oplus$ denote multiplication and addition in $\mathbb{F}_{2^n}$, respectively.

matrix A with the conditions given in Table 5.1 on page 102. We remark that the conditions given in Table 5.1 exclude some lower dimensional subspaces from the ten-dimensional space containing A and there are plenty of matrices that satisfy our constraints (see page 102 for a more rigorous analysis). Hence, a randomly selected matrix A works for our constructions with high probability. In practice, however, we recommend using the following matrix (cf. Table 5.1 for the constraints on the entries of A) as it can be implemented using a few XOR instructions:

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

Therefore the output $Z \in \mathbb{F}_{2^n}$ of the compression function can be expressed as

$$Z = C^{\mathsf{post}}(a,b,c,y_1,y_2) = \begin{pmatrix} a + c + y_1 y_2 \\ y_1 + y_2 \end{pmatrix}.$$

Note that in the context of using the compression function in an iterated fashion, we need to specify which input blocks of the compression function represent the message block and which represent the state or chaining variable. Our security results on the compression function are independent of this choice; the choice may, however, significantly affect the efficiency of the design.

5.1.2 Challenges to Overcome

In this section, we provide several fairly generic collision attacks against some simpler, yet natural choices for C^{pre} and C^{post}. This demonstrates that our main construction is not over-designed; it also helps to develop some intuition for the general problem of collision resistance in the compression function. Firstly, we consider general linear preprocessing function C^{pre} and show that in order to have a collision resistance beyond $2^{n/2}$ queries, we need to use a non-linear C^{pre}. Similarly, we show that non-linear C^{post} is necessary. Furthermore, we provide evidence that a feedforward from the input to the compression function to C^{post} is required to improve collision resistance. Our final observation is a distinguishing attack against our construction.

The Need for Non-linear Preprocessing

The notion of yield (Definition 2.3.1) is particularly useful as a first step towards understanding the security of the compression function. Intuitively, for the construction that interests us, when $\text{yield}(q) \geq 2^n$, a collision is expected (due to the birthday paradox) and once $\text{yield}(q) \geq 2^{2n}$ a preimage is expected with high probability. For efficiency reasons, the preprocessing functions of most of the PuRF-based (and blockcipher-based) compression functions proposed so far are of a special type, namely linear, where a distinction can be made on the type of linearity.

In Theorem 3.2.3, we state a lower bound on $\text{yield}(q)$ for the general construction we are interested in (Definition 3.2.1) where the C^{pre} is (blockwise) linear; namely, $\text{yield}(q) \geq q^{t/c}$. Recall that t is the number of input blocks to the compression function and c the number of input blocks to the underlying primitive: In our case $t = 3$ and $c = 2$, so substituting these parameters leads to a collision attack of complexity $2^{2n/3}$ queries. But oftentimes we can achieve a significantly larger yield. Let us show formally why.

In the following, we consider general $\mathbb{F}_2$-linear preprocessing functions $C_1^{\mathsf{pre}}, C_2^{\mathsf{pre}} : \{0,1\}^{3n} \to \{0,1\}^{2n}$. Let $K_1 = \ker(C_1^{\mathsf{pre}})$ and $K_2 = \ker(C_2^{\mathsf{pre}})$ be the kernels of the linear maps C_1^{pre} and C_2^{pre}, respectively. We assume that these kernels have dimensions d_1 and d_2 (each at least n), respectively. That is, $\dim_{\mathbb{F}_2}(K_1) = d_1$ and $\dim_{\mathbb{F}_2}(K_2) = d_2$. Define $m = \dim_{\mathbb{F}_2}(K_1 \cap K_2)$; then we know that there are linear subspaces $U_1 \subset K_1$ and $U_2 \subset K_2$ (of respective dimensions $d_1 - m$ and $d_2 - m$) such that

$$(K_1 \cap K_2) \oplus U_i = K_i$$

for $i = 1,2$. Assume that we have already made the query zero string to both f^1 and f^2. If we further evaluate f^1 on $q-1$ (where $q \leq 2^{\min(d_1,d_2)-m}$) of the elements in $C_1^{\mathsf{pre}}(U_2)$ and f^2 on $q-1$ of the elements in $C_2^{\mathsf{pre}}(U_1)$, we obtain a yield of $2^m(q-1)^2$. Indeed, there are 2^m elements in the shared kernel. Besides, f^1-queries result in zero inputs for f^2, which already exists in the query list. Similarly, f^2-queries result in zero in f^1 inputs as well; moreover, they can be combined in $(q-1)^2$ ways to evaluate the compression function in accordance with the queries already made. So, once $q - 1 \geq 2^{(n-m)/2}$, collisions are expected with high probability (due to the birthday paradox) as the yield will contain at least 2^n output points (assuming uniformity of the output points). Because the maximum number of required queries is achieved for $m = 0$, which implies $d_1 = d_2 \geq n$, we need to ask at most $2^{n/2}$ queries to each of the underlying primitives to obtain a collision. Note that it is possible to ask $2^{n/2}$ queries in such a scenario as both $d_1 - m$ and $d_2 - m$ are at least $n > n/2$. As a result, we can conclude that, using linear C_1^{pre} and C_2^{pre}, it is not possible (in the information-theoretic setting) to achieve collision resistance beyond $2^{n/2}$ queries.

This brings us to the question: Is there a less expensive way to get non-linearity in C^{pre} (without aversely affecting the security bound) than by using field multiplication as presented in Construction 5.1.1? The answer is “maybe". In our analysis, we only require C^{pre} to satisfy two properties in order to get a useful collision resistance bound: (i) a good upper bound on $\text{yield}(q)$ (e.g., the one given in Proposition 5.1.4) and (ii) *the completion property* (Definition 5.1.3, see also Proposition 5.1.4). Our results are valid for any compression function with C^{pre} satisfying (i) and (ii). Therefore, it is well possible that a better C^{pre} (i.e., a more efficient one or one with better upper bound on $\text{yield}(q)$) exists. The reason we use the preprocessing suggested in Construction 5.1.1 is that it provides a satisfactory upper bound on $\text{yield}(q)$ (Proposition 5.1.4). In addition, it satisfies the completion

property (Proposition 5.1.4). We leave finding a better C^{pre} ('better' in the sense as discussed above) as an open problem.

The Need for Feedforwarding

Even if f^1 and f^2 are PuRFs *without* an inverse available to the adversary, a feedforward from the input of the compression function to the postprocessing function C^{post} is necessary to achieve non-trivial collision resistance. The attack we describe below is a variant of the well-known 'degrees of freedom' attack considered by Peyrin et al. [154], which itself is based on an older attack by Hohl et al. [78] (however this precursor exploits linearity of C^{pre} and C^{post} in addition to the lack of feedforward). Peyrin et al. [154] focus on achieving *optimal* collision resistance and, interestingly enough, they (correctly) claim that a feedforward is insufficient to prevent their generic attacks. Seurin and Peyrin [180] later express the stronger belief that a feedforward cannot improve the collision security of a PuRF-based scheme *at all*, when the calls to the underlying PuRFs are made in parallel. However, when collision resistance does not need not be optimal, we conclude that a feedforward *helps* and the claim of Seurin and Peyrin is incorrect for non-optimal collision resistance.

Suppose that no feedforward is present, specifically that C^{post} effectively only depends on the two outputs of f^1 and f^2. That is,

$$C^{\mathsf{post}}(a,b,c,y_1,y_2) = C^{\mathsf{post}}(y_1,y_2)\,.$$

As, on average, every C_2^{pre} output has 2^n preimages, there is (at least one) f^2 input with at least 2^n corresponding f^1 inputs. Query f^2 on this particular input and f^1 on $2^{n/2}$ arbitrarily chosen (distinct) corresponding inputs. Lacking any feedforward, a collision for f^1 will directly lead to a full collision (as the output of f^2 is essentially fixed). Noting that f^1 is a random function outputting n bits only, the birthday bound implies that the given number of queries suffices to expect a collision under f^1. If we cast this attack in the terminology of our proof of collision resistance, the lack of a feedforward allows us to create and exploit a degenerate partition (cf. Lemma 5.2.1).

The Need for Non-linear Postprocessing

Even if our preprocessing function C^{pre} is non-linear, a linear postprocessing function (even with feedforward) might still lead to an attack. Indeed, take for example the C^{pre} from Construction 5.1.1 and define C^{post} as follows

$$C^{\mathsf{post}}(a,b,c,y_1,y_2) = A\cdot\begin{pmatrix} a\\ b\\ c\\ y_1\\ y_2\end{pmatrix}\text{, where}\quad A=\begin{pmatrix}\omega_{11} & \omega_{12} & \omega_{13} & \omega_{14} & \omega_{15}\\ \omega_{21} & \omega_{22} & \omega_{23} & \omega_{24} & \omega_{25}\end{pmatrix}$$

is any matrix A over $\mathbb{F}_{2^n}$. Then the following attack illustrates that we can find collisions with only $2^{n/2}$ queries by creating collinear output points: Let the adversary choose any (a,b) and query f^1 to obtain $y_1 \leftarrow f^1(a,b)$. Note that the dimension of $\ker(A)$ (over $\mathbb{F}_{2^n}$) is three; hence for any fixed triple (a,b,y_1)

there exist unique c^* and ${y_2}^* \in \{0,1\}^n$ such that

$$\begin{pmatrix} a \\ b \\ c^* \\ y_1 \\ y_2^* \end{pmatrix} \in \ker(A) .$$

Thus, for arbitrary c and y_2, we have

$$A \cdot \begin{pmatrix} a \\ b \\ c \\ y_1 \\ y_2 \end{pmatrix} = A \cdot \left(\begin{pmatrix} a \\ b \\ c^* \\ y_1 \\ {y_2}^* \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ (c-c^*) \\ 0 \\ (y_2 - {y_2}^*) \end{pmatrix} \right) = A \cdot \begin{pmatrix} 0 \\ 0 \\ (c-c^*) \\ 0 \\ (y_2 - {y_2}^*) \end{pmatrix} = \begin{pmatrix} (c-c^*)\omega_{13} + (y_2 - {y_2}^*)\omega_{15} \\ (c-c^*)\omega_{23} + (y_2 - {y_2}^*)\omega_{25} \end{pmatrix} .$$

To describe the attack, we choose arbitrary pairs (a,b), query f^1 and calculate c^* for the obtained y_1. The image Z under the compression function of (a,b,c^*) can then be expressed with

$$Z = (y_2 - {y_2}^*) \begin{pmatrix} \omega_{15} \\ \omega_{25} \end{pmatrix} .$$

That is, it lies on a fixed line (independent of (a,b)) through the origin. Moreover, this image is a uniformly random point on this line (due to the randomness of the function f^2). As the line consists of only 2^n points, after querying $2^{n/2}$ random choices of (a,b) (each with corresponding c^*), we get a collision with high probability. Hence, adding the non-linear term $y_1 y_2$ is crucial for our design.

An obvious and efficient remedy to thwart such an attack would be to add the term ac in C^{post}. This has two obvious advantages: (i) It adds non-linearity in C^{post} and (ii) as it is already computed while calculating C^{pre}, we could gain some speed. Unfortunately, a similar attack still works in spite of the above trick. Let C^{post} be defined as follows

$$C^{\mathsf{post}}(a,b,c,y_1,y_2) = A \cdot \begin{pmatrix} a \\ b \\ c \\ y_1 \\ y_2 \\ ac \end{pmatrix} , \text{where} \quad A = \begin{pmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \omega_{14} & \omega_{15} & \omega_{16} \\ \omega_{21} & \omega_{22} & \omega_{23} & \omega_{24} & \omega_{25} & \omega_{26} \end{pmatrix}$$

is a matrix (over $\mathbb{F}_{2^n}$). Therefore the compression function output $Z \in \mathbb{F}_{2^n}^2$ can be expressed as:

$$Z = a \begin{pmatrix} \omega_{11} \\ \omega_{21} \end{pmatrix} + b \begin{pmatrix} \omega_{12} \\ \omega_{22} \end{pmatrix} + c \begin{pmatrix} \omega_{13} \\ \omega_{23} \end{pmatrix} + y_1 \begin{pmatrix} \omega_{14} \\ \omega_{24} \end{pmatrix} + y_2 \begin{pmatrix} \omega_{15} \\ \omega_{25} \end{pmatrix} + ac \begin{pmatrix} \omega_{16} \\ \omega_{26} \end{pmatrix} .$$

Let the adversary pick arbitrary (a,b) and query f^1 to obtain $y_1 \leftarrow f^1(a,b)$. Define

$$\begin{pmatrix} \omega'_{13} \\ \omega'_{23} \end{pmatrix} = \begin{pmatrix} \omega_{13} + a\omega_{16} \\ \omega_{23} + a\omega_{26} \end{pmatrix} \quad \text{and} \quad A' = \begin{pmatrix} \omega_{11} & \omega_{12} & \omega'_{13} & \omega_{14} & \omega_{15} \\ \omega_{21} & \omega_{22} & \omega'_{23} & \omega_{24} & \omega_{25} \end{pmatrix} .$$

Now let c^* and ${y_2}^* \in \{0,1\}^n$ be such that

$$-\left(a\begin{pmatrix}\omega_{11}\\ \omega_{21}\end{pmatrix}+b\begin{pmatrix}\omega_{12}\\ \omega_{22}\end{pmatrix}+y_1\begin{pmatrix}\omega_{14}\\ \omega_{24}\end{pmatrix}\right)=c^*\begin{pmatrix}\omega'_{13}\\ \omega'_{23}\end{pmatrix}+y_2^*\begin{pmatrix}\omega_{15}\\ \omega_{25}\end{pmatrix}.$$

The existence of such c^* and ${y_2}^* \in \{0,1\}^n$ is dependent on the corresponding determinant; yet with high probability it is non-zero (or a is chosen in such a way that the determinant becomes non-zero). Then

$$A\cdot\begin{pmatrix}a\\ b\\ c\\ y_1\\ y_2\\ ac\end{pmatrix}=A'\cdot\begin{pmatrix}a\\ b\\ c\\ y_1\\ y_2\end{pmatrix}=A'\cdot\left(\begin{pmatrix}a\\ b\\ c^*\\ y_1\\ {y_2}^*\end{pmatrix}+\begin{pmatrix}0\\ 0\\ (c-c^*)\\ 0\\ (y_2-{y_2}^*)\end{pmatrix}\right)=\begin{pmatrix}(c-c^*)\omega'_{13}+(y_2-{y_2}^*)\omega_{15}\\ (c-c^*)\omega'_{23}+(y_2-{y_2}^*)\omega_{25}\end{pmatrix}.$$

The rest of the attack works as in the linear C^{post} case, specifically, we query (a,b,c^*) at each step and iterate this procedure $2^{n/2}$ times.

A natural question is how to choose a non-linear postprocessing function C^{post} that allows for a rigorous security analysis, as well as efficient implementations. In Section 5.1.3, we explain why we choose to add y_1y_2 (see Construction 5.1.1) as the non-linear term in C^{post} and why some other less expensive alternatives turn out to be less attractive for us.

A Potential Drawback: Indistinguishability

All the attacks discussed so far consider resistance against conventional properties of the compression functions, in particular collision resistance. Instead, we could focus on another (still fairly standard) notion, namely indistinguishability. Here an adversary either gets access to a true random function (from $3n$ to $2n$ bits) or to our construction instantiated using ideal primitives (to which the adversary has no direct access). The adversary's goal is to distinguish between these two cases and, in the information-theoretic setting, all that is of interest is the number of queries to the $3n$-to-$2n$ bit interface. To distinguish our construction from a true random function, we show that the number of required queries is only $2^{n/2}$; this is lower than we would ideally hope for.

Consider a double-call DBL compression function (see Definition 3.2.1 with $t=3$ and $c=r=s=2$) $\mathsf{h}^{f^1,f^2}\colon \{0,1\}^{3n}\to\{0,1\}^{2n}$ with *any* preprocessing function $C^{\mathsf{pre}}=(C_1^{\mathsf{pre}},C_2^{\mathsf{pre}})$ for $C_1^{\mathsf{pre}},C_2^{\mathsf{pre}}\colon \mathbb{F}_{2^n}^3\to\mathbb{F}_{2^n}^2$ and the postprocessing function C^{post} given in Construction 5.1.1. We describe a distinguishing attack against h^{f^1,f^2} showing that with only $q=2^{n/2}$ queries to the construction h^{f^1,f^2} we can distinguish it from an ideal compression function (with high probability). The attack is based on the observation that a simultaneous collision on the outputs of f^1 and f^2 (i.e., there exist distinct triples (a,b,c) and (a',b',c') in $\mathbb{F}_{2^n}^3$ such that $f^1(a,b)=f^1(a',b')$ and $f^2(c,ac+b)=f^2(c',a'c'+b')$) leads to an abnormal and detectable behavior.

Let (a,b,c) and (a',b',c') be the input triples of h^{f^1,f^2} that satisfy

$$f^1\left(C_1^{\mathsf{pre}}(a,b,c)\right)=f^1\left(C_1^{\mathsf{pre}}(a',b',c')\right)=y_1\quad\text{and}\quad f^2\left(C_2^{\mathsf{pre}}(a,b,c)\right)=f^2\left(C_2^{\mathsf{pre}}(a',b',c')\right)=y_2\,.$$

Then, the following property is satisfied due to the structure of C^{post}:

$$\mathsf{h}^{f^1,f^2}(a,b,c)+\mathsf{h}^{f^1,f^2}(a',b',c')=\begin{pmatrix}(a+a')\omega_{11}\\(a+a')\omega_{21}\end{pmatrix}+\begin{pmatrix}(c+c')\omega_{12}\\(c+c')\omega_{22}\end{pmatrix}. \tag{5.1}$$

As noted before, regardless of the preprocessing function, every C_2^{pre} output has on average 2^n preimages and there exists at least one f^2 input with at least 2^n corresponding f^1 inputs. Arbitrarily choosing $2^{n/2}$ distinct ones (among this set) and querying h^{f^1,f^2} on the corresponding full inputs makes a collision in f^1 likely. As the f^2 input is fixed, this automatically becomes a simultaneous collision and (5.1) is satisfied. Crucially, this can be verified based on the inputs and outputs of h^{f^1,f^2} only; no direct access to the internals f^1 and f^2 is needed. For a random function, however, the probability for a single pair to satisfy (5.1) is only 2^{-2n}, so the probability to find a pair among the $2^{n/2}$ evaluations is at most 2^{-n}. Indeed, we can form $\binom{2^{n/2}}{2}\approx 2^n$ pairs out of $2^{n/2}$ queries; hence the probability of obtaining a pair that satisfies (5.1) is $2^n\cdot 2^{-2n}\approx 2^{-n}$.

Conceivably, better indistinguishability could be achieved by introducing further non-linearity in the postprocessing phase. Because our focus is primarily on achieving collision resistance, we do not explore this possibility further, nor do we attempt to pin down exact indistinguishability bounds for our construction. As a final remark, if we consider modes of operation, then bad randomness can typically be nullified by some suitable postprocessing (cf. [49, 56]) for the hash function.

5.1.3 Design Rationale for Pre and Postprocessing Functions

On the Choice of Szemerédi–Trotter Preprocessing Function

A typical information-theoretic (collision or preimage-finding) adversary can try to arrange its queries to the underlying primitives in such a way that it maximizes its yield. For our construction, the preprocessing function C^{pre} fully determines the relationship between the queries made to the primitive and the compression function evaluations this enables. Our search is therefore for preprocessing functions $C_1^{\mathsf{pre}}, C_2^{\mathsf{pre}}\colon \{0,1\}^{3n}\to\{0,1\}^{2n}$ such that yield(q) does not grow too fast as a function of q.

Ideally, if an adversary asks q queries to each of the primitives, this would only result in at most q full evaluations. However, without taking C^{pre} into account, an adversary asking q queries to each of the two primitives can combine them in q^2 ways. Although many combinations do not contribute to the yield (i.e., they are not in the image of C^{pre} and yield$(q)\ll q^2$), there is a potential here for the yield to be significantly larger than q. Indeed, when an adversary asks all 2^{2n} possible queries (for each of the primitives), it will be able to evaluate the compression function at all 2^{3n} possible inputs. Consequently, it is inevitable that for $q=2^{2n}$ the yield is $q^{3/2}$. The question is whether we can find a preprocessing function C^{pre} that has good behavior for $q<2^{2n}$ as well. As it turns out, we can do reasonably well by exploiting results from incidence geometry. With an eye on the minimum requirements on C^{pre} in order for the security proofs to go through, we introduce some additional terminology.

Definition 5.1.3. Let $(a,b),(c,d)\in\mathbb{F}_{2^n}^2$ denote the query pairs made to f^1 and f^2, respectively. We call a query pair (a,b)–(c,d) *compatible* if and only if $((a,b),(c,d))$ is in the image of C^{pre}. In addition, a query (a,b) is called (c,d)-compatible or vice versa if the pair (a,b)–(c,d) is compatible. For the

preprocessing function C^{pre} from Construction 5.1.1, a pair (a,b)–(c,d) is compatible if and only if $d = ac + b$ is satisfied. Finally, a preprocessing function C^{pre} satisfies the completion property if and only if (i) (a,b) and c (ii) (c,d) and a uniquely determine a compatible query pair (a,b)–(c,d) for any $a,b,c,d \in \mathbb{F}_{2^n}$.

Proposition 5.1.4. *C^{pre} from Construction 5.1.1 has the completion property and*

$$\mathrm{yield}(q) \leq q^{3/2} + q \,.$$

The proof of Proposition 5.1.4 relies on the finite field version of a theorem by Szemerédi and Trotter [197], which is given in Theorem 5.1.5.

Theorem 5.1.5 (Szemerédi–Trotter over finite fields)**.** *Let $\mathbb{F}$ be a finite field and P (respectively L) be a set of points (respectively lines) in $\mathbb{F}^2$. Let $I(P,L) = \{(p,\ell) \mid (p,\ell) \in P \times L \text{ and } p \in \ell\}$. Then*

$$|I(P,L)| \leq \min\left(|P||L|^{1/2} + |L|, |L||P|^{1/2} + |P|\right) \,.$$

Proof. We follow the proof given by Tao [198] on his web-page. Using the definition of $|I(P,L)|$, we can write it as

$$|I(P,L)| = \sum_{\ell \in L} |P \cap \ell| \,.$$

Cauchy–Schwarz inequality leads to

$$\sum_{\ell \in L} |P \cap \ell|^2 \geq \frac{|I(P,L)|^2}{|L|} \,.$$

Now, using a basic combinatorial argument, we observe that

$$\sum_{\ell \in L} \left(|P \cap \ell|^2 - |P \cap \ell|\right) = |\{(p,q,\ell) \in P \times P \times L : p \neq q; p,q \in \ell\}| \,,$$

where the right-hand side of the equation is at most $|P|^2$ as two distinct points are incident to at most one line. Therefore,

$$\sum_{\ell \in L} |P \cap \ell|^2 \leq |I(P,L)| + |P|^2 \,.$$

Finally, we compare the above with the inequality obtained from Cauchy–Schwarz:

$$\frac{|I(P,L)|^2}{|L|} \leq \sum_{\ell \in L} |P \cap \ell|^2 \leq |I(P,L)| + |P|^2 \,,$$

which implies

$$\left(|I(P,L)| - |P||L|^{1/2}\right) \leq \frac{|I(P,L)|\,|L|}{\left(|I(P,L)| + |P||L|^{1/2}\right)} = |L|\left(\frac{|I(P,L)|}{\left(|I(P,L)| + |P||L|^{1/2}\right)}\right) \leq |L| \,.$$

A dual argument (swapping the role of lines and points) finalizes the proof. □

Proof of Proposition 5.1.4. Firstly, we remark that the completion property can be algebraically verified: We need to show that an f^1-query pair (a,b) along with c and an f^2-query pair (c,d) with

an input block a uniquely determine a compatible query pair (a,b)–(c,d). Assume first that (a,b) and c are given. Then, by the definition of C_2^{pre} we obtain a (unique) $d = ac + b$ (using the finite field paradigms), which defines the query pair (a,b)–(c,d). Now suppose there exists another f^1-query pair (a',b') with an input block c' defining the same query pair (a,b)–(c,d). Then, necessarily, $(a,b) = (a',b')$ and $c = c'$; therefore $d' = a'c' + b' = d$. Now, let us show that it is also the case for f^2-query pair (c,d) and an input block a. It defines the query pair (a,b)–(c,d) for unique $b = d - ac$. Now assume that there exists yet another f^2-query pair (c',d') and an input block a' defining the same query pair (a,b)–(c,d). Then $(c,d) = (c',d')$ and $a = a'$; therefore $b' = d' - a'c' = b$.

To determine the yield(q), we interpret the output (a,b) of C_1^{pre} as the line $\ell : y = ax + b$ in $\mathbb{F}_{2^n}^2$ and the output of C_2^{pre} as a point on ℓ. This renders bounding the yield an immediate consequence of the above incidence theorem (originally due to Klein) that is a finite field version of a theorem (Theorem 5.1.5) of Szemerédi and Trotter over the reals. To finish the proof of Proposition 5.1.4, note that the sets $\mathcal{Q}[1]$ and $\mathcal{Q}[2]$ (the list of queries made to f^1 and f^2, respectively) correspond to the lines L and the points P, respectively, and $|I(\mathcal{Q}[2],\mathcal{Q}[1])|$ counts exactly the number of compression function inputs whose mapping can be completely determined by the given queries. Specifying $|\mathcal{Q}[1]| = |\mathcal{Q}[2]| = q$ yields the proposition statement. □

Non-Linear Matrix-Style Postprocessing

Our postprocessing is clearly inspired by the use of $\mathbb{F}_{2^n}$-matrices by Rogaway and Steinberger [173]; but with the crucial difference that we add non-linearity by introducing the term $y_1 y_2$. Omitting this non-linear term is fatal for security, as shown in Section 5.1.2, which contains a concrete collision attack on the linear-only version. In contrast, for our construction, the adversary's control is significantly reduced.

Now we describe the reasoning behind the choice of the non-linear term $y_1 y_2$. We first remark that allowing for high degree terms in y_1 and y_2 is beyond our analysis; this is simply because of the fact that our current choice leads to an easy geometric interpretation of the event of finding a collision whenever a fresh query is made to f^1 and f^2. Specifically, by considering only lines of the specific form (see Section 5.2 for the details) in the plane $\mathbb{F}_{2^n}^2$, we can analyze the collision resistance of our construction. Using higher degree terms in y_1 and y_2 requires the extension of our analysis to curves over $\mathbb{F}_{2^n}^2$ which is, for the moment, beyond reach using current proof technique.

Let us see why we do not prefer some simpler choices for C^{post}. Observe that our current choice requires one full finite field multiplication; therefore our search is for the operations with less overhead. Consider the postprocessing function $C^{\mathsf{post}} : \mathbb{F}_{2^n}^5 \to \mathbb{F}_{2^n}^2$ defined by

$$C^{\mathsf{post}}(a,b,c,y_1,y_2) = A \cdot \begin{pmatrix} a \\ b \\ c \\ y_1 \\ y_2 \\ \mathsf{F}(a,b,c,y_1,y_2) \end{pmatrix}, \text{ where } \quad A = \begin{pmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \omega_{14} & \omega_{15} & \omega_{16} \\ \omega_{21} & \omega_{22} & \omega_{23} & \omega_{24} & \omega_{25} & \omega_{26} \end{pmatrix}$$

is a matrix (over $\mathbb{F}_{2^n}$) and $\mathsf{F} : \mathbb{F}_{2^n}^5 \to \mathbb{F}_{2^n}$ is a non-linear function. Let us first consider the following choice: $\mathsf{F}(a,b,c,y_1,y_2) = a^2 + c^2$ (similar choices that contain only the input blocks lead to the

same issue). Compared to $\mathsf{F}(a,b,c,y_1,y_2) = y_1 y_2$ in Construction 5.1.1, this function leads to a more efficient scheme as squaring is, in general, less expensive than full finite field multiplication in binary extension fields. The digest Z is then expressed as follows:

$$Z = a\begin{pmatrix}\omega_{11}\\ \omega_{21}\end{pmatrix} + b\begin{pmatrix}\omega_{12}\\ \omega_{22}\end{pmatrix} + c\begin{pmatrix}\omega_{13}\\ \omega_{23}\end{pmatrix} + y_1\begin{pmatrix}\omega_{14}\\ \omega_{24}\end{pmatrix} + y_2\begin{pmatrix}\omega_{15}\\ \omega_{25}\end{pmatrix} + (a^2+c^2)\begin{pmatrix}\omega_{16}\\ \omega_{26}\end{pmatrix}.$$

Consider the following strategy. Let an adversary $\mathscr{A}$ fix (a,b) and obtain the corresponding $y_1 = f^1(a,b)$ and let

$$a\begin{pmatrix}\omega_{11}\\ \omega_{21}\end{pmatrix} + b\begin{pmatrix}\omega_{12}\\ \omega_{22}\end{pmatrix} + y_1\begin{pmatrix}\omega_{14}\\ \omega_{24}\end{pmatrix} + a^2\begin{pmatrix}\omega_{15}\\ \omega_{25}\end{pmatrix} = K \in \mathbb{F}_{2^n}^2.$$

Now observe, for a given $K \in \mathbb{F}_{2^n}^2$, that any $c^*, y_2^* \in \mathbb{F}_{2^n}$ satisfying

$$c^*\begin{pmatrix}\omega_{13}\\ \omega_{23}\end{pmatrix} + y_2^*\begin{pmatrix}\omega_{15}\\ \omega_{25}\end{pmatrix} = -K$$

leads to

$$A\cdot\begin{pmatrix}a\\ b\\ c\\ y_1\\ y_2\\ a^2+c^2\end{pmatrix} = A\cdot\left(\begin{pmatrix}a\\ b\\ c^*\\ y_1\\ {y_2}^*\\ a^2+{c^*}^2\end{pmatrix} + \begin{pmatrix}0\\ 0\\ (c-c^*)\\ 0\\ (y_2-{y_2}^*)\\ (c^2-{c^*}^2)\end{pmatrix}\right) = A\cdot\begin{pmatrix}0\\ 0\\ (c-c^*)\\ 0\\ (y_2-{y_2}^*)\\ (c^2-{c^*}^2)\end{pmatrix}.$$

To describe the attack, we follow the same reasoning as in the linear C^{post} case: We choose arbitrary pairs (a,b), query f^1 and calculate c^* (with the above property) for the obtained y_1. The image under the compression function of (a,b,c^*) then lies on a uniformly random point on the fixed line:

$$(y_2-{y_2}^*)\begin{pmatrix}\omega_{15}\\ \omega_{25}\end{pmatrix}.$$

As the line consists of only 2^n points, after querying $2^{n/2}$ arbitrary choices of (a,b) (each with corresponding c^*), we obtain a collision with high probability. The only requirement for the above attack is $\omega_{13}\omega_{25} \neq \omega_{15}\omega_{23}$ (or $\omega_{11}\omega_{24} \neq \omega_{14}\omega_{21}$, which holds with probability close to one for an arbitrarily chosen matrix A, if we start fixing (c,d) rather than (a,b)). A variant of the following attack is also applicable for many (efficient) non-linear functions that require only squaring and XOR.

We leave finding a more efficient postprocessing function C^{post} such that the resulting compression function still has provable properties open. As we detail here and previously, many natural choices turn out to be either insecure or inadequate in terms of provability.

5.1.4 Security Claims

In Theorems 5.1.6 and 5.1.8 (for collision and everywhere preimage resistance, respectively) we state our security claims for a large class of double-call DBL PuRF-based compression functions (see Definition 3.2.1) with parameters $t = 3$ and $c = s = r = 2$. Our analysis is valid for any double-call DBL PuRF-based compression function of the above class whenever C^{pre} satisfies the completion

property and C^{post} is the one given in Construction 5.1.1. Our compression function proposal (Construction 5.1.1) is a special instance of the designs considered in Theorems 5.1.6 and 5.1.8. Corresponding Corollaries 5.1.7 and 5.1.9 present the ramifications of Theorems 5.1.6 and 5.1.8 to Construction 5.1.1. The proof of Theorems 5.1.6 and 5.1.8 are given in Sections 5.2 and 5.3, respectively.

Theorem 5.1.6 (Query-complexity coll-security bound)**.** *Let h^{f^1,f^2} be a double-call DBL PuRF-based compression function (Definition 3.2.1) with parameters $t = 3$ and $c = r = s = 2$. Let C^{post} be the postprocessing function given in Construction 5.1.1 and let C^{pre} be (any) preprocessing function that satisfies the completion property. Let k, μ, γ and λ be arbitrary positive integers with $\lambda \geq 3$. Then*

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q) \leq \frac{\kappa Y}{2^n} + \frac{q\gamma^2}{2^{n-1}} + \frac{\binom{q}{\gamma}}{2^{(\gamma-1)n-1}} + 2^{2n}\left(\frac{Y}{2^n}\right)^{k+1} + \frac{\binom{q}{\mu}}{2^{(\mu-1)n-1}} + \frac{\binom{q}{\lambda}}{2^{(\lambda-2)n-1}} + \frac{q}{2^{n-1}}\,,$$

where $\kappa = k\lambda + \mu$ and $Y = \mathrm{yield}(q)$.

Corollary 5.1.7. *Let h^{f^1,f^2} be the compression function given in Construction 5.1.1. Then, for all $\delta > 0$ and $q \leq 2^{2n(1-\delta)/3}$ it holds (asymptotically in n) that*

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q) = o(1)\,.$$

Proof. For any $\delta > 0$, we show asymptotically (in n) that there exist positive constants k, λ, γ, μ and $\kappa = k\lambda + \mu$ such that each term in the statement of Theorem 5.1.6 vanishes given that $q = 2^{2n(1-\delta)/3}$. Using the trivial bound $\binom{a}{b} \leq a^b$ for positive integers a and b (where $a \geq b$), together with $\mathrm{yield}(q) \leq q^{3/2} + q$ (Proposition 5.1.4), we obtain

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q) \leq \frac{\kappa(q^{3/2}+q)}{2^n} + 2^{2n}\left(\frac{q^{3/2}+q}{2^n}\right)^{k+1} + \frac{q^\mu}{2^{(\mu-1)n-1}} + \frac{q^\lambda}{2^{(\lambda-2)n-1}} + \frac{q\gamma^2}{2^{n-1}} + \frac{q^\gamma}{2^{(\gamma-1)n-1}} + \frac{q}{2^{n-1}}\,.$$

Substituting $q = 2^{2n(1-\delta)/3}$ (and using the fact that $q^{3/2} \geq q$), we get

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q) \leq 2\kappa 2^{-\delta n} + 2^{2n+k+1-\delta(k+1)n} + \frac{2^{-2\delta\mu/3}}{2^{n(\mu/3-1)-1}} + \frac{2^{-2\delta\lambda/3}}{2^{n(\lambda/3-2)-1}} + \frac{2^{-2\delta/3}(\gamma^2+1)}{2^{n/3-1}} + \frac{2^{-2\delta\gamma/3}}{2^{n(\gamma/3-1)-1}}\,,$$

where

$$\kappa 2^{-\delta n+1} + \frac{2^{-2\delta/3}(\gamma^2+1)}{2^{n/3-1}} = o(1) \quad \text{as} \quad n \to \infty\,.$$

Moreover, the remaining terms vanish for $k > 2/\delta - 1, \mu, \gamma > 3, \lambda > 6, \kappa = k\lambda + \mu$. Hence, the claim follows. □

Theorem 5.1.8 (Query-complexity epre-security bound)**.** *Let h^{f^1,f^2} be a double-call DBL PuRF-based compression function (Definition 3.2.1) with parameters $t = 3$ and $c = r = s = 2$. Let C^{post} be the postprocessing function given in Construction 5.1.1 and let C^{pre} be (any) preprocessing function that satisfies the completion property. Then, for arbitrary positive integer $\kappa > 1$,*

$$\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q) \leq \frac{\kappa q}{2^{n-1}} + 2^{n+1}\binom{q}{\kappa}\left(\frac{1}{2^{n-1}}\right)^\kappa + \frac{q}{2^{n-1}}\,.$$

We remark that $\mathrm{yield}(q)$ no longer plays a role in the bound for $\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q)$; it essentially shows that the

yield-maximization is not the best strategy to find preimages. Finally we note that the requirements (listed in Table 5.1 on page 102) for our collision resistance analysis are sufficient (we simply require a subset of the conditions) for our preimage resistance result as well.

Corollary 5.1.9. *Let* h^{f^1,f^2} *be the compression function given in Construction 5.1.1. Then, for all* $\delta > 0$ *and* $q \leq 2^{n(1-\delta)}$ *it holds (asymptotically in n) that*

$$\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q) = o(1) \,.$$

Proof. The idea is the same as in Corollary 5.1.7: For any $\delta > 0$, we show asymptotically (in n) that there exists a positive constant κ such that $\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q)$ vanishes given that $q = 2^{n(1-\delta)}$. Substituting $q = 2^{n(1-\delta)}$ in the above theorem statement (and using $\binom{q}{\kappa} \leq q^{\kappa}$), we reach

$$\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q) \leq 2^{n+1}\left(2^{n(1-\delta)}\right)^{\kappa}\left(\frac{1}{2^{n-1}}\right)^{\kappa} + \frac{2^{n(1-\delta)}}{2^{n-1}} + \frac{\kappa 2^{n(1-\delta)}}{2^{n-1}} = 2^{n+\kappa+1-\delta n\kappa} + (\kappa+1)2^{1-\delta n} \,.$$

The last term vanishes as n gets large and so does the first term whenever $\kappa > 1/\delta$. □

Optimization of the Bounds

In the following, we deal with the optimization of the bounds obtained in Theorems 5.1.6 and 5.1.8. We begin with bounding the expression $\binom{a}{b}$ for $a > b$. By Stirling's approximation we have:

$$\binom{a}{b} \leq \frac{1}{\sqrt{2\pi}}\sqrt{\frac{a}{b(a-b)}}\frac{a^a}{b^b(a-b)^{a-b}} = \left(\frac{a}{b}\right)^b\sqrt{\frac{a}{2\pi b(a-b)}}\left(\frac{a}{a-b}\right)^{a-b} \,.$$

Observe that

$$\left(\frac{a}{a-b}\right)^{a-b} = \left(1 + b\left(\frac{1}{a-b}\right)\right)^{a-b} \leq e^b \,,$$

where the inequality simply follows from

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

Therefore, we attain

$$\binom{a}{b} \leq \left(\frac{ae}{b}\right)^b\sqrt{\frac{a}{2\pi b(a-b)}} \leq \left(\frac{ae}{b}\right)^b \,.$$

Now consider the bound given in Theorem 5.1.6 for arbitrary positive integers k, μ, γ and λ with $\lambda \geq 3$ and $\kappa = k\lambda + \mu$:

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q) \leq \frac{\kappa Y}{2^n} + \frac{q\gamma^2}{2^{n-1}} + \frac{\binom{q}{\gamma}}{2^{(\gamma-1)n-1}} + 2^{2n}\left(\frac{Y}{2^n}\right)^{k+1} + \frac{\binom{q}{\mu}}{2^{(\mu-1)n-1}} + \frac{\binom{q}{\lambda}}{2^{(\lambda-2)n-1}} + \frac{q}{2^{n-1}} \,.$$

From Proposition 5.1.4 we know that $Y \leq q^{3/2} + q$; moreover by using the inequality $\binom{a}{b} \leq (ae/b)^b$ obtained above, we get

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q) \leq \frac{\kappa q^{3/2}}{2^n} + \frac{\kappa q}{2^n} + \frac{q\gamma^2}{2^{n-1}} + \frac{(eq/\gamma)^\gamma}{2^{(\gamma-1)n-1}} + 2^{2n}\left(\frac{q^{3/2}+q}{2^n}\right)^{k+1} + \frac{(eq/\mu)^\mu}{2^{(\mu-1)n-1}} + \frac{(eq/\lambda)^\lambda}{2^{(\lambda-2)n-1}} + \frac{q}{2^{n-1}} \,.$$

We neglect the optimization of the terms

$$\frac{\kappa q}{2^n} + \frac{q}{2^{n-1}}$$

as they are dominated by term $\kappa q^{3/2}/2^n$. For the remaining terms, we note the following straightforward intuition [5]: Once the terms containing the same free parameters are chosen to be close-to-equal, then the overall bound becomes close-to-optimal. We start our analysis with the expression

$$\frac{q\gamma^2}{2^{n-1}} = \frac{(eq/\gamma)^\gamma}{2^{(\gamma-1)n-1}} ,$$

which after standard algebraic manipulations results in

$$\gamma = \frac{2n - \log_2(q) - 2\log_2(\gamma)}{n - \log_2(q) - \log_2(e) + \log_2(\gamma)} \leq \frac{2n - \log_2(q)}{n - \log_2(q) - \log_2(e)} .$$

Similarly, optimizing the expression (note that $\kappa = k\lambda + \mu$) by equating both terms in

$$\frac{\mu q^{3/2}}{2^n} + \frac{(eq/\mu)^\mu}{2^{(\mu-1)n-1}}$$

gives

$$\mu \leq \frac{2n - (3/2)\log_2(q)}{n - \log_2(q) - \log_2(e)} .$$

Finally, we study the expression:

$$\frac{k\lambda q^{3/2}}{2^n} + 2^{2n}\left(\frac{q^{3/2}+q}{2^n}\right)^{k+1} + \frac{(eq/\lambda)^\lambda}{2^{(\lambda-2)n-1}} .$$

To proceed, we suggest the following. We choose $k = k_0$ such that the term

$$2^{2n}\left(\frac{q^{3/2}+q}{2^n}\right)^{k_0+1}$$

is below a certain threshold (for a given n and an upper bound on q). Then, we use the same method for the remaining terms to get an optimized λ. More specifically,

$$\lambda \leq \frac{3n - (3/2)\log_2(q) - \log_2(k_0)}{n - \log_2(q)} .$$

We now illustrate the choice of k_0 for $n = 128$. We choose our threshold probability to be $1/2$; hence

$$2^{2n}\left(\frac{q^{3/2}+q}{2^n}\right)^{k_0+1} \leq 2^{2n}\left(\frac{q^{3/2}}{2^{n-1}}\right)^{k_0+1} < \frac{1}{2} \quad \Rightarrow \quad k_0 > \frac{2n+1}{n-1-(3/2)\log_2(q)} - 1 .$$

We select the smallest positive integer k_0 satisfying the above constraint for a given n and an upper bound on q. To illustrate, for $n = 128$ and $q \leq 2^{80}$, we obtain $k_0 = 36$. The graph of $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(q)$, for $n = 128$, $k_0 = 36$ and the optimized values of λ, γ and μ, as a function of $\log_2(q)$ is illustrated in

5. This technique is not the only way to obtain an optimized bound. Indeed, an alternative method is to look at the terms that contain the same parameter (say α) and to find the optimal value of α such that the sum achieves its smallest value. This can be done using standard techniques such as taking the derivative of the sum, setting it to zero and solving for α; yet the terms in question are transcendental functions, that do not allow for such a direct optimization.

Figure 5.1 on page 85. We note that an immediate application of the above technique to the bound given in Theorem 5.1.8 gives

$$\kappa \leq \frac{2n - \log_2(q)}{n - 1 - \log_2(q) - \log_2(e)} .$$

This in turn implies, for $n = 128$, that any adversary needs to ask at least (approximately) 2^{121} queries to the underlying primitives to find a preimage with high probability.

5.2 Proof of Collision Resistance (Theorem 5.1.6)

5.2.1 Overall Strategy

Let $\mathscr{A}$ be a collision-finding adversary that asks at most q queries to each of the public random functions f^1 and f^2 (without loss of generality we assume that the adversary asks exactly q queries to both). Our goal is to bound $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{h}}(\mathscr{A})$, in particular the probability of the event $\mathsf{coll}(\mathscr{Q})$, where $\mathscr{Q}$ is adaptively generated by $\mathscr{A}$ [6].

We break down the query list $\mathscr{Q}$ into $\mathscr{Q}[1]$ and $\mathscr{Q}[2]$; this makes a distinction between the query-response pairs of f^1 and f^2. We slightly abuse notation and use $\mathscr{Q}$ (and derived symbols such as $\mathscr{Q}_i$) interchangeably as a random variable (when it is the direct result of playing the collision game), or as a dummy variable (e.g., when we want to quantify over all possible instantiations), where the context makes the precise meaning clear. In all cases we can use the global parameter q for the number of f^1 and f^2 queries and $Y = \mathrm{yield}(q)$ to bound the yield.

To bound the probability of an adversary finding a collision, we first look at the probability that any specific query completes the collision: Fix i and consider the event $\mathsf{coll}(\mathscr{Q}_i) \wedge \neg\mathsf{coll}(\mathscr{Q}_{i-1})$. Here we call query i *fresh* and we say it *causes* a collision. For concreteness, suppose the i'th query is an f^1-query (a,b), then observe that it adds a new point to the yield set for every (a,b)-compatible pair (c,d) that was already in $\mathscr{Q}_{i-1}$. Now the i'th query can cause a collision in two different ways:

Case I For some preceding and compatible (c,d) the outcome was already part of the yield, that is to say, a duo of compatible and colliding pairs (a,b)–(c,d) and (a',b')–(c',d') is formed with the triple $\{(a',b'),(c,d),(c',d')\} \subseteq \mathscr{Q}_{i-1}$ (so in particular $(a,b) \neq (a',b')$).

Case II For two distinct preceding and compatible (c,d) and (c',d') the two freshly added points happen to collide. For this, a duo of distinct compatible and colliding pairs exists with $(a,b) = (a',b')$ (and necessarily $\{(c,d),(c',d')\} \subseteq \mathscr{Q}_{i-1}$).

In general, finding collisions via Case I is easier to achieve; assume there already exist compression function outputs in the yield. Because we add certain new points and we look for collisions among those already existing, we (intuitively) increase the chance of a collision compared to Case II, where we are only interested in collisions among the freshly added points. If there are only freshly added outputs or the yield contains a small number of elements, Case II might become easier to satisfy.

6. We can define the $(\mathscr{X},\mathscr{Y})$ random-system h (as in Section 3.3) mimicking the interaction of $\mathscr{A}$ with the uniformly random functions used in Construction 5.1.1. Let $\mathscr{X}$ be the set of pairs $(x,t) \in \mathbb{F}_{2^n}^2 \times \{1,2\}$, where t indicates which of the two uniformly random functions $\mathscr{A}$ is making queries to and x is the input to that uniform random function. Similarly, the set $\mathscr{Y}$ denotes the set of elements $(y,t) \in \mathbb{F}_{2^n} \times \{1,2\}$, where t (deterministic) indicates which of the two uniform random functions this output comes form and y is an output of that uniform random function (corresponding to the input x). It is not hard to see that the conditional probability distributions that define the random system h can be derived as in Example 3.3.2 (taking into account t).

When the i'th query is an f^2-query (c,d), an analogous split can be made and we fold into the two cases just described. We associate the events $\mathsf{coll}_I(\mathscr{Q})$ and $\mathsf{coll}_{II}(\mathscr{Q})$ with these two cases, where $\mathsf{coll}_I(\mathscr{Q})$ occurs if and only if for some i'th query Case I occurs. As the first collision to occur needs to fall into at least one of these two cases, it follows that

$$\mathsf{coll}(\mathscr{Q}) \equiv (\mathsf{coll}_I(\mathscr{Q}) \vee \mathsf{coll}_{II}(\mathscr{Q}))$$

or considering the i'th time that $\mathscr{A}$ makes its query we have

$$\mathsf{coll}(\mathscr{Q}_i) \wedge \neg\mathsf{coll}(\mathscr{Q}_{i-1}) \equiv (\mathsf{coll}_I(\mathscr{Q}_i) \wedge \neg\mathsf{coll}(\mathscr{Q}_{i-1})) \vee (\mathsf{coll}_{II}(\mathscr{Q}_i) \wedge \neg\mathsf{coll}(\mathscr{Q}_{i-1})) .$$

The probability of these events depends strongly on the number of compatible queries already in $\mathscr{Q}_{i-1}$; we denote this number by (random variable) n_i. Although we know (by design) that

$$\sum_{i=1}^{2q} n_i \le \mathrm{yield}(q) ,$$

a straightforward union bound fails to take this into account properly: Because potentially $n_i \approx q$, naive bounding of $\sum_{i=1}^{2q} n_i$ would be quadratic in q (which is typically much larger than $\mathrm{yield}(q)$). Dealing with this in the case of non-adaptive adversaries is straightforward (because such an adversary needs to commit to the n_i values in advance), but requires a more careful treatment in the case of adaptive adversaries. The tools developed in Section 3.3 serve for this purpose.

To bound the probability of $\mathsf{coll}_I(\mathscr{Q})$, we additionally condition under the event where there is not too many collinear output points. More precisely, the elements $Z \in \mathbb{F}_{2^n}^2$ that feature in $\mathrm{yieldset}^{\mathsf{h}}(\mathscr{Q})$, are considered as points in the two-dimensional vector space (over $\mathbb{F}_{2^n}$). For positive integer κ, $\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})$ is set if and only if $\mathscr{Q}$ gives rise to more than κ collinear output points. The reason to consider collinearity will become evident shortly.

A High-Level Overview of the Proof and the Road Map

To improve the readability of the proof of our collision resistance, we now present a high-level overview of it, as well as a road map that facilitates the tracking of the key points in the proof. We start with the observation, for any $\mathscr{Q}$, that

$$\mathsf{coll}(\mathscr{Q}) \equiv (\mathsf{coll}_I(\mathscr{Q}) \vee \mathsf{coll}_{II}(\mathscr{Q})) \equiv (\mathsf{coll}_I(\mathscr{Q}) \wedge \neg\mathsf{coll}_{II}(\mathscr{Q})) \vee \mathsf{coll}_{II}(\mathscr{Q}) .$$

The expression $(\mathsf{coll}_I(\mathscr{Q}) \wedge \neg\mathsf{coll}_{II}(\mathscr{Q}))$ is equivalent to

$$\big(\mathsf{coll}_I(\mathscr{Q}) \wedge \neg\mathsf{coll}_{II}(\mathscr{Q}) \wedge \neg\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})\big) \vee \big(\mathsf{coll}_I(\mathscr{Q}) \wedge \neg\mathsf{coll}_{II}(\mathscr{Q}) \wedge \mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})\big) .$$

Using the fact that

$$\big(\mathsf{coll}_I(\mathscr{Q}) \wedge \neg\mathsf{coll}_{II}(\mathscr{Q}) \wedge \neg\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})\big) \Rightarrow \big(\mathsf{coll}_I(\mathscr{Q}) \wedge \neg\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})\big)$$

and

$$\big(\mathsf{coll}_I(\mathscr{Q}) \wedge \neg\mathsf{coll}_{II}(\mathscr{Q}) \wedge \mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})\big) \Rightarrow \big(\neg\mathsf{coll}_{II}(\mathscr{Q}) \wedge \mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})\big)$$

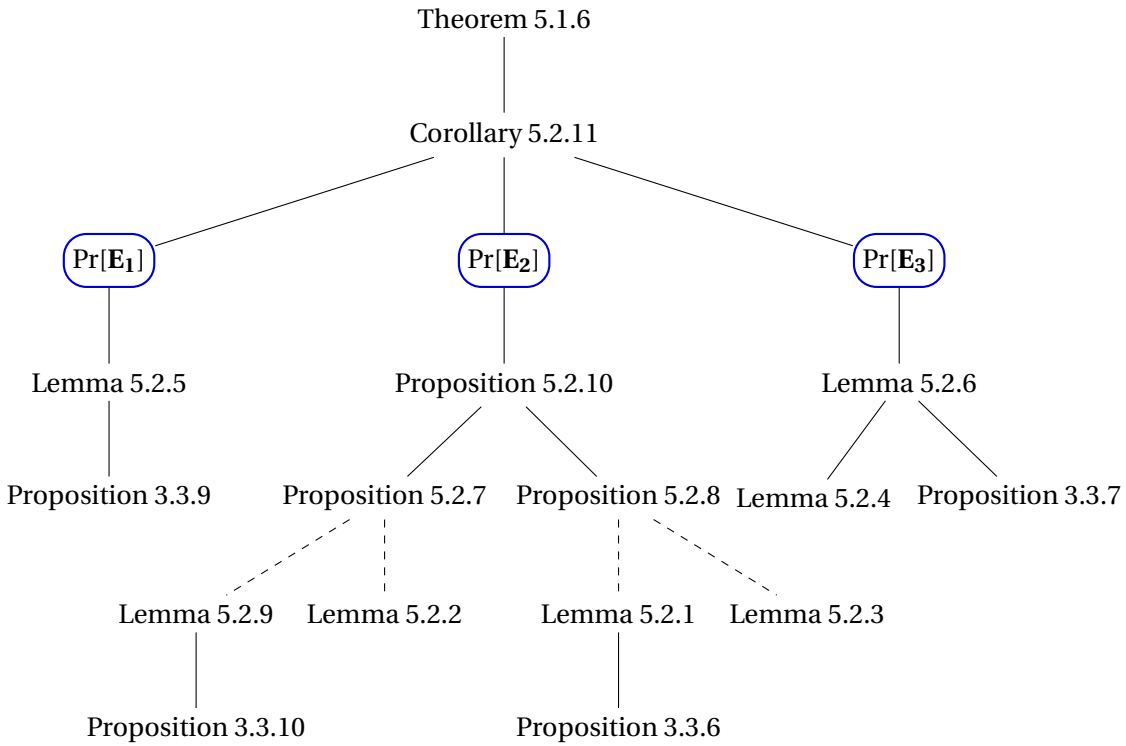

Figure 5.3 – A tree representation of the hierarchical relations of the statements used in the proof of Theorem 5.1.6. Probabilities do not correpond to a node in the tree; they are added for illustrative purposes to show the correspondence with the related statements. Solid lines show that the proof of the statement that appears in the parent node makes use of the statement shown in the child node. Dashed lines show that the child node is implied by the parent node.

we reach

$$\mathsf{coll}(\mathscr{Q}) \Rightarrow \underbrace{\left(\mathsf{coll}_I(\mathscr{Q}) \wedge \neg \mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})\right)}_{\mathbf{E_1}} \vee \underbrace{\left(\neg \mathsf{coll}_{II}(\mathscr{Q}) \wedge \mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})\right)}_{\mathbf{E_2}} \vee \underbrace{\mathsf{coll}_{II}(\mathscr{Q})}_{\mathbf{E_3}} . \tag{5.2}$$

The idea of our proof is to find separate upper bounds for the probability of the events $\mathbf{E_i}$ for $i = 1,2,3$ and then use the union bound to finalize the proof in Corollary 5.2.11 (i.e., $\sum_{i=1}^{3} \Pr[\mathbf{E_i}]$ provides the overall upper bound). Now we explain step by step where we provide these upper bounds in the subsequent sections and to which terms in Theorem 5.1.6 they correspond. Figure 5.3 illustrates the abstract overview of the road map used in the proof.

Let us start by looking at the event $\mathbf{E_1}$. An upper bound for $\Pr[\mathbf{E_1}]$ is given in Lemma 5.2.5 (which itself is based on Proposition 3.3.9). The bound that Lemma 5.2.5 gives corresponds to the term $\kappa Y/2^n$ in Theorem 5.1.6.

Similarly, an upper bound for $\Pr[\mathbf{E_3}]$ is established in Lemma 5.2.6: The probability of the event $\mathbf{E_3}$ is upper bounded by using an auxiliary flag; its probability of happening is bounded in Lemma 5.2.4.

Condition		Where used		Reference
			(Theorem 5.1.6)	(Section 5.2)
C1:	$\omega_{13}\omega_{25} \neq \omega_{15}\omega_{23}$	(**S**)	Non-degeneracy of $\mathscr{L}_{1:}$-lines	(5.3)
		(**N**)	Non-parallel $\mathscr{P}_{1:}$-partitions	Lemma 5.2.2
C2:	$\omega_{14}\omega_{25} \neq \omega_{15}\omega_{24}$	(**S**)	Non-degeneracy of $\mathscr{L}_{2:}$-lines	(5.4)
		(**N**)	Non-parallel $\mathscr{P}_{2:}$-partitions	Lemma 5.2.2
C3:	$(\omega_{11}, \omega_{21}) \neq 0$	(**N**)	Non-degeneracy of $\mathscr{P}_{1:}$-partitions	Lemma 5.2.1
C4:	$(\omega_{12}, \omega_{22}) \neq 0$	(**N**)	Non-degeneracy of $\mathscr{P}_{2:}$-partitions	Lemma 5.2.1
C5:	$(\omega_{15}, \omega_{25}) \neq 0$	(**N**)	Non-linearity of C^{post}	Construction 5.1.1

Table 5.1 – A summary of the properties of the entries of A (see Construction 5.1.1) used in the proof of Theorem 5.1.6. (**N**) denotes that the condition is necessary, whereas (**S**) denotes it is sufficient.

With the use of Proposition 3.3.7, an upper bound for $\Pr[\mathbf{E_3}]$ is then obtained. The terms

$$\frac{q\gamma^2}{2^{n-1}} + \frac{q^\gamma}{2^{(\gamma-1)n-1}}$$

from Theorem 5.1.6 correspond to the upper bound for $\Pr[\mathbf{E_3}]$.

Finally, we explain where the bounds for $\Pr[\mathbf{E_2}]$ (i.e., the remaining terms from Theorem 5.1.6) come from. We note that the overall Section 5.2.4 is devoted to bound $\Pr[\mathbf{E_2}]$. More specifically, we use Proposition 5.2.10 to establish an implication that leads to an upper bound for $\Pr[\mathbf{E_2}]$, which itself uses Propositions 5.2.7 and 5.2.8. Moreover, several auxiliary events, which are defined and investigated in Sections 5.2.2 and 5.2.4, are required to finalize the bound $\Pr[\mathbf{E_2}]$: The upper bound for the auxiliary events are given in Lemmas 5.2.1, 5.2.2, 5.2.3 and 5.2.9.

On the Matrix A Used in C^{post}

In the following, we consider a general matrix A (see Construction 5.1.1) over $\mathbb{F}_{2^n}$

$$A = \begin{pmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \omega_{14} & \omega_{15} \\ \omega_{21} & \omega_{22} & \omega_{23} & \omega_{24} & \omega_{25} \end{pmatrix}$$

for the proof of Theorem 5.1.6. The conditions on the entries of the matrix A required throughout, as well as where exactly they are used, are provided in Table 5.1. For comparison, we can check that the suggested matrix A

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

satisfies the requirements given in Table 5.1.

Now let us investigate how many matrices of the above form satisfy the conditions listed in Table 5.1 and see with what probability an arbitrarily selected matrix works for our construction. We start with the number of possibilities for the fifth column as it is the shared column for properties **C**1 and **C**2 (and observe that conditions **C**1 and **C**2 imply condition **C**5). Excluding the zero column, there are $((2^n)^2 - 1)$ choices for the fifth column. For each such choice, we have $((2^n)^2 - 2^n)$ possibilities for the third and the fourth columns (for conditions **C**1 and **C**2): These columns can be anything but the

multiples of the fifth column. Finally, we are left with $((2^n)^2-1)$ possibilities for columns one and two. All in all, there are

$$\left((2^n)^2-1\right)^3 \cdot \left((2^n)^2-2^n\right)^2 = (2^n-1)^5 \cdot (2^n)^2 \cdot (2^n+1)^3 \approx 2^{10n}$$

matrices for our purposes. Therefore, the probability that a randomly selected matrix A satisfies our criteria is close to one and there are plenty of matrices that are suitable for our construction.

Output Lines

As we assume that an adversary can only output a collision for which it has made all necessary queries, an f^1-query (a,b) can only complete a collision using an already present compatible f^2-query (c,d). Let (a,b) be an f^1-query and let (c,d) be a preceding (a,b)-compatible f^2-query with $y_2 = f^2(c,d)$. The output $C^{\mathsf{post}}(a,b,c,y_1,y_2)$ of the compression function on input (a,b,c) then lies on the line (in $\mathbb{F}_{2^n}^2$)

$$\mathscr{L}_{1:c,d,y_2;a}\colon \left\{ \begin{pmatrix} a\omega_{11}+c\omega_{12}+y_2\omega_{14} \\ a\omega_{21}+c\omega_{22}+y_2\omega_{24} \end{pmatrix} + y_1 \begin{pmatrix} \omega_{13}+y_2\omega_{15} \\ \omega_{23}+y_2\omega_{25} \end{pmatrix} \mid y_1 \in \mathbb{F}_{2^n} \right\}, \tag{5.3}$$

where we get the actual output point for (a,b,c) by setting $y_1 = f^1(a,b)$. The randomness of f^1 results in a random point on $\mathscr{L}_{1:c,d,y_2;a}$. In the following, we call the vector

$$\begin{pmatrix} \omega_{13}+y_2\omega_{15} \\ \omega_{23}+y_2\omega_{25} \end{pmatrix}$$

the *slope* of the line; and the vector

$$\begin{pmatrix} a\omega_{11}+c\omega_{12}+y_2\omega_{14} \\ a\omega_{21}+c\omega_{22}+y_2\omega_{24} \end{pmatrix}$$

is called the *offset*. Note that the line cannot be degenerate (see condition **C**1 in Table 5.1), i.e., it has non-zero slope, as

$$\begin{vmatrix} \omega_{13} & \omega_{15} \\ \omega_{23} & \omega_{25} \end{vmatrix} \neq 0 \quad \Rightarrow \quad \begin{pmatrix} \omega_{13}+y_2\omega_{15} \\ \omega_{23}+y_2\omega_{25} \end{pmatrix} \neq 0\,.$$

(An alternative sufficient condition for non-degeneracy is $(\omega_{15},\omega_{25}) = 0 \neq (\omega_{13},\omega_{23})$.) Similarly, let (c,d) be an f^2-query and let (a,b) be a preceding (c,d)-compatible f^1-query. The output of the compression function on (a,b,c) then lies on the line:

$$\mathscr{L}_{2:a,b,y_1;c}\colon \left\{ \begin{pmatrix} a\omega_{11}+c\omega_{12}+y_1\omega_{13} \\ a\omega_{21}+c\omega_{22}+y_1\omega_{23} \end{pmatrix} + y_2 \begin{pmatrix} \omega_{14}+y_1\omega_{15} \\ \omega_{24}+y_1\omega_{25} \end{pmatrix} \mid y_2 \in \mathbb{F}_{2^n} \right\}. \tag{5.4}$$

This time the output point is obtained by setting $y_2 = f^2(c,d)$. Again, the randomness of f^2 results in a random point on $\mathscr{L}_{2:a,b,y_1;c}$. We note that now non-degeneracy follows from the condition **C**2 given in Table 5.1. Now it is easy to see why we do not want too many collinear points: It would ease the collision-finding considerably. The reason is simply that, due to the above lines, the freshly generated compression function outputs lie on a well-defined (output) line. If there are too many

already existing digest values lying on the newly added output lines, then the probability of finding a collision will increase significantly. We quantify the effect of collinearity with the event $\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})$: For a positive integer κ, $\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})$ is set to true if and only if $\mathscr{Q}$ gives rise to more than κ collinear output points (we need $\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})$ both for the aforementioned events $\mathbf{E_1}$ and $\mathbf{E_2}$, see (5.2)).

5.2.2 Building Tools for the Proof: Partitions, Bunches and Some Auxiliary Events

Here we present some additional terminology required in the subsequent sections. First we discuss the *partitions* and *bunches*: These are the geometric objects defined via the output lines introduced above. Briefly (a more formal treatment follows shortly), a partition, which is defined separately for varying a and c values (corresponding to f^1- and f^2-queries), is a set of parallel (output) lines that partitions $\mathbb{F}_{2^n}^2$. A bunch, however, is defined again as a set of lines—this time for a fixed f^1 or f^2 query—yet it is different from a partition in the sense that it contains (possibly) a set of non-parallel lines.

Second, we introduce certain auxiliary events and provide upper bounds for their probabilities to occur. These events are important while we analyze collinearity; in particular the event $\Pr[\mathbf{E_2}]$ (introduced in (5.2)). Specifically, we are interested in four auxiliary events: degenerate partitions, parallel partitions, local collinearity and target local collinearity.

Degenerate partition occurs when a partition collapses to a single line rather than partitioning the output plane. Parallel partition, however, is the event where two or more different partitions happen to be the same. Finally, we are interested in the events local and target local collinearity; these events convey information about the collinearity of certain points (that are different from output points) in $\mathbb{F}_{2^n}^2$. The only difference between these two is that the target local collinearity requires these output points to be on a line with a prescribed slope.

Partitions and Bunches

Suppose that an f^2-query (c,d) resulted in $y_2 = f^2(c,d)$. By the completion property, we get, for each $a \in \mathbb{F}_{2^n}$, a unique b such that (a,b) is (c,d)-compatible. Now we recall that if we query $f^1(a,b)$, the resulting yield point lies on the line $\mathscr{L}_{1:c,d,y_2;a}$. From Equation (5.3) of $\mathscr{L}_{1:c,d,y_2;a}$, it follows that the slope of these lines is fixed (because (c,d) and y_2 are fixed) and independent of a; hence by ranging over all possible a values in $\mathbb{F}_{2^n}$ we achieve a set of (parallel) lines. This is what we call a partition (created via a values):

$$\mathscr{P}_{1:c,d,y_2} = \{\mathscr{L}_{1:c,d,y_2;a} \mid a \in \mathbb{F}_{2^n}\} .$$

In our proof of collision resistance, a partition conveys information about how the query $y_2 = f^2(c,d)$ might be used in *future* use. Similarly, we define the partition $\mathscr{P}_{2:a,b,y_1}$ defined via different c values in $\mathbb{F}_{2^n}$ as follows:

$$\mathscr{P}_{2:a,b,y_1} = \{\mathscr{L}_{2:a,b,y_1;c} \mid c \in \mathbb{F}_{2^n}\} .$$

The opposite notion to a partition is a bunch: It conveys information about how a current query $f^1(a,b)$ might lead to an immediate collision. For all preceding and (a,b)-compatible $(c_j,d_j) \in \mathscr{Q}$, for some integer $j \geq 1$, the bunch of interest is the collection of lines

$$\mathscr{B}_{1:(a,b)}(\mathscr{Q}) = \left\{\mathscr{L}_{1:c_j,d_j,y_{2:j};a} \mid y_{2:j} = f^2(c_j,d_j)\ ,\ (c_j,d_j,y_{2:j}) \in \mathscr{Q} \wedge (c_j,d_j) \text{ compatible with } (a,b)\right\} .$$

(We also write $\mathscr{B}_{1:i}$ if the i'th query is an f^1-query (a,b).) The idea is that the answer $y_1 = f^1(a,b)$ specifies a point on each of these lines to be added to the yield set; we refer to this as *realizing* the bunch. For the record, $\mathscr{B}_{2:(c,d)}(\mathscr{Q})$ is defined analogously:

$$\mathscr{B}_{2:(c,d)}(\mathscr{Q}) = \left\{ \mathscr{L}_{2:a_j,b_j,y_{1:j};c} \mid y_{1:j} = f^1(a_j,b_j)\ ,\ (a_j,b_j,y_{1:j}) \in \mathscr{Q} \wedge (a_j,b_j) \text{ compatible with } (c,d) \right\}\ .$$

Both partitions and bunches will be our central tools while we analyze collinearity, in particular the event $\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})$ and $\Pr[\mathbf{E_2}]$.

Degenerate Partitions

We have seen that a partition contains a set of parallel lines. If different choices of a lead to different lines, the lines compatible to (c,d) necessarily partition the output plane (justifying our terminology). It is possible however that regardless of the a values, we end up with identical lines (though with a different parametrization). In such a case, a partition collapses to a single line and we speak of a degenerate partition [7].

A degenerate partition causes problems in our proof, because it allows an adversary to create many collinear points (by ranging over a). Let $\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})$ denote the event that $\mathscr{Q}$ gives rise to a degenerate partition (either via different a or c values). The following proposition states an upper bound for $\Pr\left[\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})\right]$.

Lemma 5.2.1 (Degenerate partitions)**.** *Let $\mathscr{Q}$ be generated by an adaptive adversary, then*

$$\Pr\left[\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})\right] \leq \frac{q}{2^{n-1}}\ ,$$

and if

$$\omega_{11}\omega_{23} \neq \omega_{13}\omega_{21}\ ,\ \omega_{12}\omega_{24} \neq \omega_{14}\omega_{22}\ ,\ \omega_{11}\omega_{25} = \omega_{15}\omega_{21} \quad \textit{and} \quad \omega_{12}\omega_{25} = \omega_{15}\omega_{22}\ ,$$

then

$$\Pr\left[\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})\right] = 0\ .$$

Proof. Let us fix c,d and y_2 such that $y_2 = f^2(c,d)$. It follows from (5.3) that a partition $\mathscr{P}_{1:c,d,y_2}$, corresponding to (c,d) with $y_2 = f^2(c,d)$, is degenerate if and only if the vectors

$$\begin{pmatrix}\omega_{11}\\ \omega_{21}\end{pmatrix} \quad \text{and} \quad \begin{pmatrix}\omega_{13}+y_2\omega_{15}\\ \omega_{23}+y_2\omega_{25}\end{pmatrix}$$

are collinear. As both vectors are non-zero, collinearity is equivalent to the condition that

$$(\omega_{11}\omega_{23} - \omega_{13}\omega_{21}) + y_2(\omega_{11}\omega_{25} - \omega_{15}\omega_{21}) = 0\ .$$

If

$$\omega_{11}\omega_{23} - \omega_{13}\omega_{21} \neq 0 \quad \text{yet} \quad \omega_{11}\omega_{25} - \omega_{15}\omega_{21} = 0$$

7. Indeed, our attack from Section 5.1.2 against the feedforward-less version of C^{post} exploits degenerate partitions where all the output points lie on the same line.

this equation has no solution for y_2 and, consequently, no degenerate partition $\mathscr{P}_{1:c,d,y_2}$ can exist. (Note that the above argument is independent from the choice of c, d and y_2 and it holds for any given such triple.) Otherwise, there is at most one solution and, for a given f^2-query, it is returned with probability 2^{-n}. The case for f^1-queries follows from (5.4) by symmetry. Then,

$$(\omega_{12}\omega_{24} - \omega_{14}\omega_{22}) + y_1(\omega_{12}\omega_{25} - \omega_{15}\omega_{22}) = 0$$

has to be avoided. As there are in total $2q$ queries (q to each of f^1 and f^2), a union bound leads to the stated upper bound. We remark that the analysis of this event can also be performed by using Proposition 3.3.6; we only need to define $\mathbf{E}(\mathscr{Q}) = \mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})$ and proceed accordingly. □

Parallel Partitions

Now we define another bad event, *parallel partitions*, that can potentially help a collision-finding adversary create collinear points. We have seen that, once answered, a single f^2-query (c, d) determines a well-defined slope for the partition $\mathscr{P}_{1:c,d,y_2}$. If two or more distinct partitions (of the same type) have the same slope, we call the partitions parallel. Specifically, $\mu > 1$ parallel partitions exist if there exist distinct (c_i, d_i) for $i = 1, \ldots, \mu$ such that

$$f^2(c_1, d_1) = \cdots = f^2(c_\mu, d_\mu) = y_2 \,.$$

The case for the parallel partitions due to f^1-queries is defined analogously.

Parallel partitions can be exploited by an adversary to create collinear points. Assume there exist $\mu > 1$ parallel partitions (without loss of generality) $\mathscr{P}_{1:c_1,d_1,y_2}, \ldots, \mathscr{P}_{1:c_\mu,d_\mu,y_2}$. The idea is to pick a line, say, $\mathscr{L}_{1:c_1,d_1,y_2;a_1} \in \mathscr{P}_{1:c_1,d_1,y_2}$ and then to choose the same line from the remaining partitions $\mathscr{P}_{1:c_2,d_2,y_2}, \ldots, \mathscr{P}_{1:c_\mu,d_\mu,y_2}$ by selecting the correct a values that result in $\mathscr{L}_{1:c_1,d_1,y_2;a_1}$, i.e., $\mathscr{L}_{1:c_1,d_1,y_2;a_1} = \ldots = \mathscr{L}_{1:c_\mu,d_\mu,y_2;a_\mu}$. In turn, this process makes the corresponding points $f^1(a_i, b_i)$ (for $i = 2, \ldots, \mu$) lie on the line that was selected.

The number of parallel partitions is tightly related to a standard occupancy problem. Consequently, avoiding parallel partitions altogether is not realistic, yet we can put reasonable bounds on too much parallelism occurring. This suffices for our bounding of collinear points. We define $\mathsf{bad}_{\mathsf{pp}[\mu]}(\mathscr{Q})$ to be the event when $\mathscr{Q}$ results in at least μ parallel partitions (of identical type). The following lemma serves for stating an upper bound for $\Pr\left[\mathsf{bad}_{\mathsf{pp}[\mu]}(\mathscr{Q})\right]$.

Lemma 5.2.2 (Parallel partitions)**.** *Let $\mathscr{Q}$ be generated by an adaptive adversary, then*

$$\Pr\left[\mathsf{bad}_{\mathsf{pp}[\mu]}(\mathscr{Q})\right] \leq \frac{\binom{q}{\mu}}{2^{(\mu-1)n-1}}$$

for any integer $\mu > 0$.

Proof. Without loss of generality, let us consider the parallel partitions due to f^2-queries only (the other case is disjunct and similar). We begin with the base case $\mu = 2$. In order for two partitions (corresponding to $y_2 = f^2(c, d)$ and $y_2' = f^2(c', d')$) to be parallel, it is necessary and sufficient that

the slope vectors

$$\begin{pmatrix}\omega_{13}+y_2\omega_{15}\\ \omega_{23}+y_2\omega_{25}\end{pmatrix} \quad \text{and} \quad \begin{pmatrix}\omega_{13}+y_2'\omega_{15}\\ \omega_{23}+y_2'\omega_{25}\end{pmatrix}$$

are collinear. The latter is equivalent to

$$(y_2-y_2')(\omega_{13}\omega_{25}-\omega_{15}\omega_{23})=0\,.$$

Given that the determinant is non-zero (see condition **C**1 in Table 5.1), this boils down to the condition that $y_2 = y_2'$ [8]. Hence, the probability of having $\mu = 2$ parallel partitions can be bounded from above by a straightforward collision bound $\binom{q}{2}/2^n$. The case for arbitrary μ follows similarly from the above analysis; it requires in particular a μ-way collision among the y_2 values (explaining the stated bound after taking into account the analogous f^1-case as well). □

Local Collinearity

Now we discuss another auxiliary event that is used in our collinearity analysis. Suppose an f^1-query results in $y_1 = f^1(a,b)$. We associate with this query-response pair a point $(a,y_1) \in \mathbb{F}_{2^n}^2$. Let $\mathsf{bad}_{\mathsf{lc}[\lambda]}(\mathscr{Q})$ be the event that there exist at least λ pairs of f^1-queries (a_i,b_i) with distinct a_i values, such that the associated points $(a_i,y_{1:i})$ are collinear or, alternatively, that there exist at least λ pairs of f^2-queries (c_i,d_i) with distinct c_i values, such that the points $(c_i,y_{2:i})$ are collinear. We use the following lemma for bounding $\Pr\left[\mathsf{bad}_{\mathsf{lc}[\lambda]}(\mathscr{Q})\right]$.

Lemma 5.2.3 (Local collinearity)**.** *Let $\mathscr{Q}$ be generated by an adaptive adversary, then*

$$\Pr\left[\mathsf{bad}_{\mathsf{lc}[\lambda]}(\mathscr{Q})\right] \le \frac{\binom{q}{\lambda}}{2^{(\lambda-2)n-1}}$$

for any integer $\lambda > 0$.

Proof. Without loss of generality, let us consider local collinearity due to f^1-queries. Selecting λ pairs (a,b) out of q f^1-queries with distinct a values can be done in at most $\binom{q}{\lambda}$ different ways; and for any λ pairs of queries, the probability that the corresponding points lie on the same line is $1/2^{(\lambda-2)n}$: Once two of the y_1 values for the corresponding a values are specified, the remaining y_1 values are determined uniquely. Local collinearity due to f^2-queries is similar; applying a union bound gives the stated claim. □

Target Local Collinearity

For local collinearity, we are interested in any λ associated points being collinear, without worrying about which line they are on. However, in an upcoming case we are only interested in points all lying on a line with a pre-specified slope (the offset of the line is not fixed in advance). Let $\mathsf{bad}_{\mathsf{slc}[\gamma]}(\mathscr{Q})$ be the event that $\mathscr{Q}[1]$ or $\mathscr{Q}[2]$ leads to more than γ associated points collinear with pre-specified, non-vertical, slope (where the slopes for $\mathscr{Q}[1]$ and $\mathscr{Q}[2]$ might differ). Lemma 5.2.4 provides an upper bound for $\Pr\left[\mathsf{bad}_{\mathsf{slc}[\gamma]}(\mathscr{Q})\right]$.

8. Our attack in Section 5.1.2 against the linear version of C^{post}, where $\omega_{15} = \omega_{25} = 0$, indeed exploits parallel partitions.

Lemma 5.2.4 (Target local collinearity)**.** *Let $\mathcal{Q}$ be generated by an adaptive adversary, then*

$$\Pr\left[\mathsf{bad}_{\mathsf{slc}[\gamma]}(\mathcal{Q})\right] \leq \frac{\binom{q}{\gamma}}{2^{(\gamma-1)n-1}}$$

for any integer $\gamma > 0$.

Proof. The proof follows from Lemma 5.2.3 taking into account the fact that the fixed slope and a single point is sufficient to determine all the remaining points. □

5.2.3 Bounding Collisions: Focusing on $\Pr[\mathbf{E_1}]$ and $\Pr[\mathbf{E_3}]$

We now have all the tools required in the subsequent sections to finalize the proof of Theorem 5.1.6; hence starting with this subsection, we can proceed on the road map we provided in Figure 5.3. We begin our analysis with upper bounding the probabilities of the events $\mathbf{E_1}$ and $\mathbf{E_3}$ given in (5.2). Firstly, we provide an upper bound for

$$\Pr[\mathbf{E_1}] = \Pr[\mathsf{coll}_I(\mathcal{Q}) \wedge \neg\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathcal{Q})]$$

in Lemma 5.2.5. Secondly, we analyze

$$\Pr[\mathbf{E_3}] = \Pr[\mathsf{coll}_{II}(\mathcal{Q})]$$

in Lemma 5.2.6.

Bounding $\Pr[\mathbf{E_1}]$: We bound in the following lemma the probability of finding a collision relative to an existing yield point, provided that there are not too many collinear points yet. Recall that $\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathcal{Q})$ is set if and only if $\mathcal{Q}$ gives rise to more than κ collinear output points.

Lemma 5.2.5. *Let i be a positive integer that satisfies $i \leq q$ and Let $\mathcal{Q}_i$ be arbitrary query list that satisfies $\neg\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathcal{Q}_i)$ (for some positive integer κ). Then the probability that the i'th query causes a collision with an element in* $\mathrm{yieldset}^{\mathsf{h}}(\mathcal{Q}_{i-1})$ *can be upper bounded by $n_i\kappa/2^n$, where n_i denotes the number of elements in $\mathcal{Q}_{i-1}$ that are compatible with the i'th query. Moreover*

$$\Pr[\mathbf{E_1}] = \Pr[\mathsf{coll}_I(\mathcal{Q}) \wedge \neg\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathcal{Q})] \leq \frac{\kappa Y}{2^n}\,,$$

where $Y = \mathrm{yield}(q)$.

Proof. Each of the n_i compatible elements, together with the i'th query, defines a line such that the random answer to the i'th query determines which point is added to the yield set. The condition $\neg\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathcal{Q}_i)$ implies that on each of these lines, there are at most κ previous yield points. As the underlying primitive is a random function, the answer is fully random and for a given line, one of the previous yield points is hit with probability at most $\kappa/2^n$. A union bound over the n_i lines gives the local bound $n_i\kappa/2^n$. To obtain the major bound, we use $B_\Sigma = \kappa Y/2^n$ (Proposition 3.3.9) as $\sum_{i=1}^{2q} n_i \leq Y$. □

Bounding $\Pr[\mathbf{E_3}]$**:** We now bound the probability of finding an instantaneous collision with a fresh query, first given that $\neg\mathsf{bad}_{\mathsf{slc}[\gamma]}(\mathcal{Q}_i)$ holds. Recall that $\mathsf{bad}_{\mathsf{slc}[\gamma]}(\mathcal{Q})$ is the event that $\mathcal{Q}[1]$ or $\mathcal{Q}[2]$ leads to more than γ associated points collinear with a pre-specified, non-vertical slope (see Lemma 5.2.4 for an upper bound for $\mathsf{bad}_{\mathsf{slc}[\gamma]}(\mathcal{Q})$). Then we finalize our bound for $\Pr[\mathsf{coll}_{II}(\mathcal{Q})]$ using Proposition 3.3.7 along with Lemma 5.2.4 (cf. Figure 5.3).

Lemma 5.2.6. *Let i be a positive integer that satisfies $i \le q$ and let $\mathcal{Q}$ be generated by an adaptive adversary. Then*

$$\Pr\left[\mathsf{coll}_{II}(\mathcal{Q}_i) \wedge \neg\mathsf{coll}_{II}(\mathcal{Q}_{i-1}) \wedge \neg\mathsf{bad}_{\mathsf{slc}[\gamma]}(\mathcal{Q}_i)\right] \le \frac{\gamma^2}{2^n}$$

and

$$\Pr[\mathbf{E_3}] = \Pr[\mathsf{coll}_{II}(\mathcal{Q})] \le \frac{q\gamma^2}{2^{n-1}} + \Pr\left[\mathsf{bad}_{\mathsf{slc}[\gamma]}(\mathcal{Q})\right]$$

for any integer $\gamma > 0$.

Proof. We only bound the probability of an f^1-query (a,b) that causes an instantaneous collision; the argument for f^2-queries is analogous. Suppose (c,d) and (c',d') are both distinct and (a,b)-compatible, and already part of the query history so that $y_2 = f^2(c,d)$ and $y_2' = f^2(c',d')$. Suppose that $y_1 = f^1(a,b)$, then (by using the definition of output lines from (5.4) on page 103) the statement that the two resulting outputs Z and Z' collide ($Z = Z'$) is equivalent to

$$\begin{pmatrix}(c-c')\omega_{12} + (y_2 - y_2')\omega_{13}\\ (c-c')\omega_{22} + (y_2 - y_2')\omega_{23}\end{pmatrix} = y_1 \begin{pmatrix}(y_2 - y_2')\omega_{15}\\ (y_2 - y_2')\omega_{25}\end{pmatrix}. \tag{5.5}$$

This implies that, for any given duo of (a,b)-compatible pairs (c,d) and (c',d'), there is at most one solution for y_1 that would give rise to a collision. Moreover, (5.5) reduces to

$$(\omega_{13}\omega_{25} - \omega_{15}\omega_{23})(y_2 - y_2')^2 + (\omega_{12}\omega_{25} - \omega_{22}\omega_{15})(c-c')(y_2 - y_2') = 0\,.$$

As $y_2 = y_2{}'$ in conjunction with (5.5) (and the completion property) would imply $(c,d) = (c',d')$, we can simplify it even further to (note the condition **C**1 from Table 5.1 for non-zero denominator)

$$(y_2 - y_2') = (c-c')\frac{\begin{vmatrix}\omega_{12} & \omega_{15}\\ \omega_{22} & \omega_{25}\end{vmatrix}}{\begin{vmatrix}\omega_{13} & \omega_{15}\\ \omega_{23} & \omega_{25}\end{vmatrix}}\,.$$

This is equivalent to saying that the points (c, y_2) and (c', y_2') are on a line with pre-specified slope (and not vertical). Given $\neg\mathsf{bad}_{\mathsf{slc}[\gamma]}(\mathcal{Q}_i)$, we know that there exist at most γ collinear points with a prescribed slope; hence we can create instantaneous collisions only by using one of $\binom{\gamma}{2}$ pairs generated among those γ points. As for each pair there is a unique y_1 and the probability of hitting the correct y_1 is $1/2^n$, we obtain the first claim (here we use the trivial upper bound $\binom{\gamma}{2} \le \gamma^2$). The second, however, follows from a union bound (over all f^1- and f^2-queries), along with Lemma 5.2.4 and Proposition 3.3.7. □

We finalize our overall collision resistance bound with the analysis of

$$\Pr[\mathbf{E_2}] = \Pr[\neg \mathsf{coll}_{II}(\mathscr{Q}) \wedge \mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})]$$

(cf. Figure 5.3), which is carried out in Section 5.2.4.

5.2.4 Bounding Overall Collinearity: Bounding $\Pr[\mathbf{E_2}]$

We now turn our attention to bounding the probability of event $\mathbf{E_2}$, i.e., of 'too much' collinearity within the yield set given that $\mathsf{coll}_{II}(\mathscr{Q})$ does not occur. The main difficulty in establishing a good bound for $\Pr[\mathbf{E_2}]$ is knowing how to properly separate the randomness of the f^1- and f^2-queries. After all, each f^1-query serves simultaneously to add points to the yield set directly (based on preceding compatible f^2-queries), as well as to set up possible future f^2-queries.

In a non-adaptive setting, the problem can be simplified considerably by rearranging the queries, so that all f^1-queries are asked first. However, even then problems remain: Although any *individual* yield point is distributed uniformly random in the output plane, two points depending on the same f^1-query are clearly dependent. Thus, we cannot just look at the much easier problem of collinearity among, say, $Y = \mathrm{yield}(q)$ points randomly distributed in the output plane.

To overcome these technical difficulties (in the adaptive setting), we use a method that we refer to as *bunching*. Now, fix i and suppose the i'th query is an f^1-query (a,b). Recall that for the f^1-query (a,b) having n_i compatible preceding f^2-queries (c_j,d_j) (with $y_{2:j} = f^2(c_j,d_j)$ for $j = 1,\ldots,n_i$), we define the bunch $\mathscr{B}_{1:i}$ consisting of the lines $\mathscr{L}_{1:c_j,d_j,y_{2:j};a}$. The answer $y_1 = f^1(a,b)$ adds a single point to the yield set for each compatible f^2-query (c_j,d_j). These n_i new points lie on the lines $\mathscr{L}_{1:c_j,d_j,y_{2:j};a}$, thereby realizing the bunch $\mathscr{B}_{1:i}$. We refer to the set of freshly added points inside a bunch as a constellation, denoted by

$$\mathscr{C}_{1:i}(\mathscr{Q}) = \{\mathsf{h}^{f^1,f^2}(a,b,c_j) \mid d_j = ac_j + b \ \wedge\ (c_j,d_j) \in \mathscr{Q}_{i-1} \ \wedge\ (c_j,d_j) \text{ compatible with } (a,b)\}\,.$$

In order to determine maximum collinearity within the yield set, we estimate (i) the probability of too much collinearity occurring within a single constellation (Proposition 5.2.8) and (ii) the probability of too many constellations being collinear (Lemma 5.2.9). Here, a set of constellations is collinear if and only if we can choose a point from each constellation in the set such that all these selected yield points are collinear. If we know that at most λ points are collinear within a single constellation, and at most k constellations are collinear, we can conclude that at most $\kappa = k\lambda$ points are collinear overall. This is formalized in the proposition below, taking into account an additional technicality (due to parallel partitions). Recall that $\mathsf{bad}_{\mathsf{int}[\lambda]}(\mathscr{Q})$ and $\mathsf{bad}_{\mathsf{pp}[\mu]}$ denote the events local collinearity and parallel partitions, respectively. Proposition 5.2.7 shows that $\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})$ can be broken down into several (monotone) sub-events, including $\mathsf{bad}_{\mathsf{int}[\lambda]}(\mathscr{Q})$ and $\mathsf{bad}_{\mathsf{pp}[\mu]}$.

Proposition 5.2.7. *Let k,λ, and μ be fixed positive integers and let $\kappa = k\lambda + \mu$. Let $\mathsf{bad}_{\mathsf{int}[\lambda]}(\mathscr{Q})$ be the event that there exists a constellation containing more than λ collinear points. Let $\mathsf{bad}_{\mathsf{ext}[k]}(\mathscr{Q})$ be the event that there exists a line ℓ passing through more than k constellations whose bunches do not contain ℓ. Then (for arbitrary $\mathscr{Q}$)*

$$\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q}) \ \Rightarrow\ \big(\mathsf{bad}_{\mathsf{int}[\lambda]}(\mathscr{Q}) \ \vee\ \mathsf{bad}_{\mathsf{ext}[k]}(\mathscr{Q}) \ \vee\ \mathsf{bad}_{\mathsf{pp}[\mu]}(\mathscr{Q})\big)\,.$$

Proof. Suppose conversely that

$$\neg\left(\mathsf{bad}_{\mathsf{int}[\lambda]}(\mathscr{Q}) \vee \mathsf{bad}_{\mathsf{ext}[k]}(\mathscr{Q}) \vee \mathsf{bad}_{\mathsf{pp}[\mu]}(\mathscr{Q})\right)$$

and that κ points are collinear, on some line ℓ. We know that there are at most μ partitions parallel to this line. Each parallel partition can only contribute to one point on ℓ. Remove these partitions and we know that there are at most k constellations collinear to this line and that each constellation contains at most λ points on the line. Thus the maximum number on the line ℓ is $k\lambda+\mu$, implying $\kappa \le k\lambda+\mu$. □

Bounding Collinearity Within a Single Constellation

Recall that $\mathsf{bad}_{\mathsf{int}[\lambda]}(\mathscr{Q})$ is the event that after all the queries, there exists a constellation (induced by $\mathscr{Q}$) containing more than λ collinear points. We can decompose this event into two auxiliary events we introduced earlier.

Proposition 5.2.8. *For arbitrary $\mathscr{Q}$, if (integer) $\lambda \ge 3$ then*

$$\left(\neg\mathsf{coll}_{II}(\mathscr{Q}) \wedge \mathsf{bad}_{\mathsf{int}[\lambda]}(\mathscr{Q})\right) \Rightarrow \left(\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q}) \vee \mathsf{bad}_{\mathsf{lc}[\lambda]}(\mathscr{Q})\right) .$$

Proof. For simplicity, we start our analysis with the case $\lambda = 3$; the case of an arbitrary λ follows easily. Let $\mathscr{Q}$ be such that $\neg\mathsf{coll}_{II}(\mathscr{Q}) \wedge \mathsf{bad}_{\mathsf{int}[3]}(\mathscr{Q})$ holds. Suppose that the event $\mathsf{bad}_{\mathsf{int}[3]}(\mathscr{Q})$ is caused by collinearity within $\mathscr{C}_{2:j}(\mathscr{Q})$, where (c,d) is the j'th query made by the adversary to f^2 such that $y_2 = f^2(c,d)$. (The case of collinearity within some $\mathscr{C}_{1:i}(\mathscr{Q})$ is analogous.) Then there exist distinct (c,d)-compatible f^1-queries $(a_1,b_1),(a_2,b_2)$, and (a_3,b_3) in $\mathscr{Q}_{j-1}$, where we simplify the indexing from e.g., a_{i_1} to a_1. Notice that $a_i \ne a_{i'}$ for $i \ne i'$ (otherwise we would have $b_i = b_{i'}$ for $i \ne i'$ by the completion property). For $i = 1,2,3$, let $y_{1:i} = f^1(a_i,b_i)$ and let the yield point (t_i,u_i) be equal to the compression function evaluated at input (a_i,b_i,c). These three output points (t_i,u_i) are all distinct due to $\neg\mathsf{coll}_{II}(\mathscr{Q})$ and they lie on the same line if and only if

$$(u_2-u_1)(t_3-t_1) = (u_3-u_1)(t_2-t_1) .$$

Expanding the explicit form of C^{post}, we obtain

$$\big(\omega_{21}+B_2(\omega_{23}+y_2\omega_{25})\big)\big(\omega_{11}+B_3(\omega_{13}+y_2\omega_{15})\big) = \big(\omega_{21}+B_3(\omega_{23}+y_2\omega_{25})\big)\big(\omega_{11}+B_2(\omega_{13}+y_2\omega_{15})\big) ,$$

where $B_i = (y_{1:i}-y_{1:1})/(a_i-a_1)$ for $i = 2,3$. The above equation simplifies to

$$(B_2-B_3)\left(\begin{vmatrix}\omega_{11} & \omega_{13}\\ \omega_{21} & \omega_{23}\end{vmatrix} + y_2\begin{vmatrix}\omega_{11} & \omega_{15}\\ \omega_{21} & \omega_{25}\end{vmatrix}\right) = 0 .$$

If $B_2 \ne B_3$, then we obtain a unique y_2 that only depends on the matrix A defining C^{post}. By inspection, it is precisely the y_2 that causes degenerate partitions in event $\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})$ (its probability is bounded in Lemma 5.2.1). The event $B_2 = B_3$, however, occurs if and only if

$$y_{1:3} = y_{1:1} + \frac{a_3-a_1}{a_2-a_1}(y_{1:2}-y_{1:1}) ,$$

which is equivalent to the points $(a_1,y_{1:1}),(a_2,y_{1:2})$ and $(a_3,y_{1:3})$ being collinear. This corresponds to

the local collinearity $\mathsf{bad}_{\mathsf{lc}[3]}(\mathcal{Q})$; its probability is bounded in Lemma 5.2.3. This concludes the proof of

$$\left(\neg\mathsf{coll}_{II}(\mathcal{Q}) \wedge \mathsf{bad}_{\mathsf{int}[\lambda]}(\mathcal{Q})\right) \Rightarrow \left(\mathsf{bad}_{\mathsf{dp}}(\mathcal{Q}) \vee \mathsf{bad}_{\mathsf{lc}[\lambda]}(\mathcal{Q})\right)$$

for $\lambda = 3$. For larger λ, observe that unless $\mathsf{bad}_{\mathsf{dp}}(\mathcal{Q})$ occurs, for any triple of points local collinearity needs to hold, which in turn implies $\mathsf{bad}_{\mathsf{lc}[\lambda]}(\mathcal{Q})$. □

Bounding Collinearity Between Constellations

To bound collinearity between constellations, we first consider collinearity with a given line ℓ. More precisely, given a line ℓ in the output plane, we are interested in bounding the probability that at least k constellations are incident to ℓ. For a line ℓ, integer k and query history $\mathcal{Q}$, let $\mathsf{bad}_{\ell-\mathsf{hit}[k]}(\mathcal{Q})$ be the event that at least k constellations are incident to ℓ, restricted to those constellations whose corresponding bunch does *not* contain ℓ. Recall that $\mathsf{bad}_{\mathsf{ext}[k]}(\mathcal{Q})$ is the event that there exists a line ℓ passing through more than k constellations whose bunches do not contain ℓ.

Lemma 5.2.9. *Let ℓ be given (also to the adversary) and let $\mathcal{Q}$ be generated adaptively. Then*

$$\Pr\left[\mathsf{bad}_{\ell-\mathsf{hit}[k]}(\mathcal{Q})\right] \leq \left(\frac{Y}{2^n}\right)^{k+1} \quad \textit{and} \quad \Pr\left[\mathsf{bad}_{\mathsf{ext}[k]}(\mathcal{Q})\right] \leq 2^{2n}\left(\frac{Y}{2^n}\right)^{k+1} .$$

Proof. Let $\mathsf{ctr}_{\ell-\mathsf{hit}}(\mathcal{Q})$ be the number of constellations that are incident to ℓ, again restricted to those constellations whose corresponding bunch does not contain ℓ. Clearly, the event $\mathsf{bad}_{\ell-\mathsf{hit}[k]}(\mathcal{Q})$ is equivalent to $\mathsf{ctr}_{\ell-\mathsf{hit}}(\mathcal{Q}) \geq k$. Note that for any i, we have $\mathsf{ctr}_{\ell-\mathsf{hit}}(\mathcal{Q}_i) - \mathsf{ctr}_{\ell-\mathsf{hit}}(\mathcal{Q}_{i-1}) \in \{0,1\}$ as constellation i can be counted at most once (specifically if it is incident to ℓ). Let $\mathsf{hit}_{\ell-\mathsf{hit}}(i)$ be the event that the bunch $\mathscr{B}_i$ upon realization is incident to ℓ. Suppose that $\ell \notin \mathscr{B}_i$ and that $\mathscr{B}_i$ consists of n_i lines (each containing an output point). Because ℓ intersects each line in a bunch in at most one point, we obtain that

$$0 < \Pr\left[\mathsf{hit}_{\ell-\mathsf{hit}}(i)\right] \leq \frac{n_i}{2^n} .$$

Due to yield restrictions, $\sum_{i=1}^{2q} n_i \leq Y$. The lemma statement follows from applying Proposition 3.3.10 with $B_\Sigma = Y/2^n$. The statement for $\Pr\left[\mathsf{bad}_{\mathsf{ext}[k]}(\mathcal{Q})\right]$ follows from the union bound over all lines ℓ. □

Proposition 5.2.10. *Let k, λ, and μ be fixed positive integers with $\lambda \geq 3$ and $\kappa = k\lambda + \mu$. Then, for arbitrary $\mathcal{Q}$*

$$\left(\neg\mathsf{coll}_{II}(\mathcal{Q}) \wedge \mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathcal{Q})\right) \Rightarrow \mathsf{flag}(\mathcal{Q}) ,$$

where

$$\mathsf{flag}(\mathcal{Q}) \equiv \left(\mathsf{bad}_{\mathsf{ext}[k]}(\mathcal{Q}) \vee \mathsf{bad}_{\mathsf{pp}[\mu]}(\mathcal{Q}) \vee \mathsf{bad}_{\mathsf{lc}[\lambda]}(\mathcal{Q}) \vee \mathsf{bad}_{\mathsf{dp}}(\mathcal{Q})\right) .$$

Moreover,

$$\Pr[\mathbf{E_2}] \leq \Pr[\mathsf{flag}(\mathcal{Q})] \leq \Pr[\mathsf{bad}_{\mathsf{ext}[k]}(\mathcal{Q})] + \Pr[\mathsf{bad}_{\mathsf{pp}[\mu]}(\mathcal{Q})] + \Pr[\mathsf{bad}_{\mathsf{lc}[\lambda]}(\mathcal{Q})] + \Pr[\mathsf{bad}_{\mathsf{dp}}(\mathcal{Q})] .$$

Proof. For any $\mathscr{Q}$, we have (using Proposition 5.2.7):

$$\begin{aligned}
\left(\neg\mathsf{coll}_{II}(\mathscr{Q}) \wedge \mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})\right) &\Rightarrow \neg\mathsf{coll}_{II}(\mathscr{Q}) \wedge \left(\mathsf{bad}_{\mathsf{int}[\lambda]}(\mathscr{Q}) \vee \mathsf{bad}_{\mathsf{ext}[k]}(\mathscr{Q}) \vee \mathsf{bad}_{\mathsf{pp}[\mu]}(\mathscr{Q})\right) \\
&\Rightarrow \left(\neg\mathsf{coll}_{II}(\mathscr{Q}) \wedge \mathsf{bad}_{\mathsf{int}[\lambda]}(\mathscr{Q})\right) \vee \mathsf{bad}_{\mathsf{ext}[k]}(\mathscr{Q}) \vee \mathsf{bad}_{\mathsf{pp}[\mu]}(\mathscr{Q}) \\
\text{(Proposition 5.2.8)} \quad &\Rightarrow \mathsf{bad}_{\mathsf{dp}}(\mathscr{Q}) \vee \mathsf{bad}_{\mathsf{lc}[\lambda]}(\mathscr{Q}) \vee \mathsf{bad}_{\mathsf{ext}[k]}(\mathscr{Q}) \vee \mathsf{bad}_{\mathsf{pp}[\mu]}(\mathscr{Q}) \\
&\equiv \mathsf{flag}(\mathscr{Q}) \,.
\end{aligned}$$

Hence, the claim follows after using the union bound. □

5.2.5 Finishing the Proof

The following corollary concludes what we have discussed so far (cf. Figure 5.3) and finishes the proof of Theorem 5.1.6 with the help of earlier obtained bounds (Lemmas 5.2.1, 5.2.2, 5.2.3 5.2.5, 5.2.6 and 5.2.9).

Corollary 5.2.11. *Let $\mathscr{Q}$ be generated by an adaptive adversary, then*

$$\Pr[\mathsf{coll}(\mathscr{Q})] \le \Pr[\mathsf{coll}_I(\mathscr{Q}) \wedge \neg\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})] + \Pr[\mathsf{coll}_{II}(\mathscr{Q})] + \Pr\left[\mathsf{flag}(\mathscr{Q})\right] \,,$$

where $\Pr\left[\mathsf{flag}(\mathscr{Q})\right]$ *can be upper bounded by*

$$\Pr\left[\mathsf{flag}(\mathscr{Q})\right] \le \Pr\left[\mathsf{bad}_{\mathsf{ext}[k]}(\mathscr{Q})\right] + \Pr\left[\mathsf{bad}_{\mathsf{pp}[\mu]}(\mathscr{Q})\right] + \Pr\left[\mathsf{bad}_{\mathsf{lc}[\lambda]}(\mathscr{Q})\right] + \Pr\left[\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})\right] \,.$$

Here, the last term $\Pr\left[\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})\right]$ *is optional (depending on the matrix A).*

Proof. The proof follows from the decomposition of the collision event given in (5.2)

$$\mathsf{coll}(\mathscr{Q}) \Rightarrow \underbrace{\left(\mathsf{coll}_I(\mathscr{Q}) \wedge \neg\mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})\right)}_{\mathbf{E_1}} \vee \underbrace{\left(\neg\mathsf{coll}_{II}(\mathscr{Q}) \wedge \mathsf{bad}_{\mathsf{cl}[\kappa]}(\mathscr{Q})\right)}_{\mathbf{E_2}} \vee \underbrace{\mathsf{coll}_{II}(\mathscr{Q})}_{\mathbf{E_3}} \,,$$

together with the use of union bound. □

5.3 Proof of Everywhere Preimage Resistance (Theorem 5.1.8)

Here, assuming familiarity with the notions introduced in Section 5.2, we prove Theorem 5.1.8. Moreover, we assume that the conditions on the matrix A provided in Table 5.1 are satisfied. Let Z be the target digest to be inverted and let $\mathscr{A}$ be a preimage-finding adversary that asks at most q queries to each of the public random functions f^1 and f^2. Our goal is to bound $\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(\mathscr{A})$, in particular the probability of the event $\mathsf{epre}_Z(\mathscr{Q})$. Similar to the collision resistance proof, we first bound the probability of finding a preimage on the i'th query (either to f^1 or f^2), conditioned on an auxiliary event not having occurred yet (Lemma 5.3.1). Then, we finalize the proof by bounding the probability of the auxiliary event (Lemma 5.3.2).

Now, for a set of queries $\mathscr{Q}$, let $\mathsf{bad}^{(1)}_{\kappa}(\mathscr{Q})$ be the event that there exists a bunch $\mathscr{B}_{1:(a,b)}$ (obtained from queries in $\mathscr{Q}$ only) having more than $\kappa > 1$ lines passing through Z. Similarly, let $\mathsf{bad}^{(2)}_{\kappa}(\mathscr{Q})$ be the event that there exists a bunch $\mathscr{B}_{2:(c,d)}$ (obtained from queries in $\mathscr{Q}$ only) that has more than κ

lines passing through Z. We define

$$\mathbf{F}(\mathscr{Q}) \equiv \mathsf{bad}_\kappa(\mathscr{Q}) \equiv \left(\mathsf{bad}_\kappa^{(1)}(\mathscr{Q}) \vee \mathsf{bad}_\kappa^{(2)}(\mathscr{Q})\right) .$$

Lemma 5.3.1. *Let C^{post} be the postprocessing function from Construction 5.1.1 (with non-degenerate lines only) and C^{pre} be any preprocessing function that satisfies the completion property. Then*

$$\Pr\left[\mathsf{epre}_Z(\mathscr{Q}_i) \mid \neg\mathsf{epre}_Z(\mathscr{Q}_{i-1}) \wedge \neg\mathsf{bad}_\kappa(\mathscr{Q}_i)\right] \leq \frac{\kappa}{2^n}$$

for any positive integers $\kappa > 1$ and $i \leq q$.

Proof. Consider the i'th query; assume that it is an f^2-query, so $y_2 \leftarrow f^2(c,d)$ (the f^1-case is analogous). Due to the condition $\neg\mathsf{bad}_\kappa(\mathscr{Q}_i)$, we know that there are at most κ lines from the bunch associated to (c,d) that go through Z. Next, it follows from (5.4) that for each of these lines, there is a unique y_2^* that leads to the target preimage Z. As the probability that $y_2 = y_2^*$ is $1/2^n$, a union bound (over at most κ lines) results in the desired probability. □

Lemma 5.3.2. *Let C^{post} be the postprocessing function from Construction 5.1.1 (with non-degenerate lines only) and C^{pre} be any preprocessing function that satisfies the completion property. Then*

$$\Pr\left[\mathsf{bad}_\kappa(\mathscr{Q})\right] \leq 2^{n+1}\binom{q}{\kappa}\left(\frac{1}{2^{n-1}}\right)^\kappa + \Pr\left[\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})\right]$$

for any positive integer $\kappa > 1$.

Proof. Without loss of generality, we concentrate on $\mathsf{bad}_\kappa^{(2)}(\mathscr{Q})$ (the analysis of the event $\mathsf{bad}_\kappa^{(1)}(\mathscr{Q})$ is symmetric); for a fixed pair (c,d) consider the event $Z \in \mathscr{L}_{2:a,b,y_1;c}$ for any (c,d)-compatible f^1-query pair (a,b) chosen by the adversary. Using (5.4) and exploiting the non-degeneracy of the output lines, we obtain that $Z \in \mathscr{L}_{2:a,b,y_1;c}$ if and only if

$$Z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} a\omega_{11} + c\omega_{12} + y_1\omega_{13} \\ a\omega_{21} + c\omega_{22} + y_1\omega_{23} \end{pmatrix} + y_2 \begin{pmatrix} \omega_{14} + y_1\omega_{15} \\ \omega_{24} + y_1\omega_{25} \end{pmatrix}$$

for some $y_2 \in \mathbb{F}_{2^n}$. This holds if and only if the vectors

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} - \begin{pmatrix} a\omega_{11} + c\omega_{12} + y_1\omega_{13} \\ a\omega_{21} + c\omega_{22} + y_1\omega_{23} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \omega_{14} + y_1\omega_{15} \\ \omega_{24} + y_1\omega_{25} \end{pmatrix}$$

are collinear. This condition reduces, after several standard algebraic manipulations, to

$$y_1^2\begin{vmatrix}\omega_{13} & \omega_{15}\\ \omega_{23} & \omega_{25}\end{vmatrix} + y_1\left(\begin{vmatrix}\omega_{14} & \omega_{13}\\ \omega_{24} & \omega_{23}\end{vmatrix} - c\begin{vmatrix}\omega_{12} & \omega_{15}\\ \omega_{22} & \omega_{25}\end{vmatrix} - a\begin{vmatrix}\omega_{11} & \omega_{15}\\ \omega_{21} & \omega_{25}\end{vmatrix} + \begin{vmatrix} z_1 & \omega_{15}\\ z_2 & \omega_{25}\end{vmatrix}\right) - \begin{vmatrix} z_1 & \omega_{14}\\ z_2 & \omega_{24}\end{vmatrix} + a\begin{vmatrix}\omega_{11} & \omega_{14}\\ \omega_{21} & \omega_{24}\end{vmatrix} + c\begin{vmatrix}\omega_{12} & \omega_{14}\\ \omega_{22} & \omega_{24}\end{vmatrix} = 0 .$$

Therefore, there exist at most two y_1 values (because $\omega_{13}\omega_{25} \neq \omega_{15}\omega_{23}$ due to non-degeneracy of the output lines; see condition **C**1 in Table 5.1) satisfying $Z \in \mathscr{L}_{2:a,b,y_1;c}$ for any (c,d)-compatible f^1-query pair (a,b).

In general, once κ different (c,d)-compatible f^1-query pairs are specified (along with the target

digest Z), κ quadratic equations need to hold, each having at most two solutions. As choosing κ distinct (c,d)-compatible f^1-query pairs can be done in at most $\binom{q}{\kappa}$ ways and the f^1 outputs are uniformly and independently drawn over $\{0,1\}^n$, we get the probability

$$\binom{q}{\kappa}\left(\frac{1}{2^{n-1}}\right)^{\kappa}$$

in order for the bunch $\mathscr{B}_{2:(c,d)}(\mathscr{Q})$ to have at least κ lines containing Z. We remark here that this holds under the assumption that there exist no degenerate partitions. A set of lines would collapse to a single line otherwise; thus, once a single line contains Z, then so does the entire degenerate partition. Note also that the above analysis is independent of d and y_2; only c values are counted. As $\mathscr{A}$ can choose c values in 2^n possible ways, we can use a union bound over all c values to get an overall bound. Moreover, considering the symmetric event $\mathsf{bad}^{(1)}_{\kappa}(\mathscr{Q})$ in turn gives

$$\Pr\left[\mathsf{bad}_{\kappa}(\mathscr{Q})|\neg\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})\right] \leq 2^{n+1}\binom{q}{\kappa}\left(\frac{1}{2^{n-1}}\right)^{\kappa} .$$

The claim follows from

$$\Pr\left[\mathsf{bad}_{\kappa}(\mathscr{Q})\right] \leq \Pr\left[\mathsf{bad}_{\kappa}(\mathscr{Q})|\neg\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})\right] + \Pr\left[\mathsf{bad}_{\mathsf{dp}}(\mathscr{Q})\right] .$$

□

We conclude the proof of Theorem 5.1.8 by summing up various bounds we obtain from Lemmas 5.3.1 and 5.3.2.

An Attack (Almost) Matching the Bound

Theorem 5.1.8 guarantees that an adversary limited (asymptotically in n) to $\mathscr{O}(2^{n(1-\delta)})$ queries (for any $\delta > 0$) has a disappearing advantage to find preimages; it is clearly suboptimal for a DBL compression function construction. Unfortunately, this is inherent to the construction and not an artifact of the proof: Indeed the proof itself is almost constructive in giving an attack with advantage at least $q/2^n$ as well, as exploited below.

Claim 5.3.3. *Let h^{f^1,f^2} be the compression function given in Construction 5.1.1 and let Z be an arbitrary target digest. Then, a preimage for Z under h^{f^1,f^2} can be found with reasonable probability after $\mathscr{O}(2^n)$ queries.*

Proof. Consider the following preimage-finding adversary $\mathscr{A}$ on input target digest Z. Let $\mathscr{A}$ randomly pick an f^2-query (c,d) and obtain its response $y_2 = f^2(c,d)$. This defines a partition $\mathscr{P}_{1:c,d,y_2}$. Unless the partition is degenerate (which happens with negligible probability), there is a unique a value such that the line $\mathscr{L}_{1:c,d,y_2;a}$ contains Z. Choose this a and the (unique) b such that (a,b) is (c,d)-compatible and query $f^1(a,b)$. The probability (over f^1) of hitting Z is $1/2^n$ as the output point is randomly assigned for the generated line. Hence after iterating this procedure 2^n times, a preimage is expected with reasonable probability. Note that the attack is (mildly) adaptive, has virtually no overhead (so the time-complexity matches the query-complexity and memory consumption is negligible) and parallelizes trivially. □

5.4 Blockcipher-Based Instantiation

When instantiating (the primitives used by) the compression function from Construction 5.1.1 with (ideal) blockciphers, there are several ways to proceed. We only discuss two, an insecure and a possibly (we leave the security proof for this construction as a future work) secure way. Note that the original PuRF-based proof is no longer valid once the underlying primitives are replaced by ideal blockciphers, mainly due to the availability of decryption queries, as exploited by the following attack.

5.4.1 Straightforward Adaptation

Let the double-call DBL blockcipher-based compression function h^{E^1,E^2} (Definition 2.3.8 with $m = \kappa = n$ and $s = 2n$) be defined by the same C^{pre} and C^{post} as in Construction 5.1.1 by setting $f^1(a,b) = E^1_b(a)$ and $f^2(c,d) = E^2_d(c)$. This leads to an insecure instantiation, as illustrated by the following attack.

Claim 5.4.1. *For the compression function h^{E^1,E^2} defined by the same C^{pre} and C^{post} as in Construction 5.1.1, collisions can be found with high probability using $\mathcal{O}(2^{n/2})$ queries and $\mathcal{O}(2^{n/2})$ time (asymptotically in n).*

Proof. We sketch two possible attacks by making use of the decryption oracle to generate all yield points on the very same line (hence limiting the possible number of compression function outputs to 2^n). The trick is suggested by our collision resistance proof (in the PuRF setting), where we can easily violate the specific conditions required in the proofs of two of the auxiliary degenerate events, i.e., degenerate partitions and parallel partitions (in Lemmas 5.2.1 and 5.2.2, respectively). In the PuRF scenario, it is indeed hard to satisfy the corresponding bad events, but for the given blockcipher-based instantiation we have the freedom to call decryption oracles. Take for instance the y_2 condition from Lemma 5.2.1 that leads to a degenerate partition.

If $\omega_{11}\omega_{25} \neq \omega_{15}\omega_{21}$ then we can simply ask the decryption of y_2 (with any key) to cause the condition. This allows an adversary to adaptively construct compatible (a,b)–(c,d)-pairs in such a way that the corresponding output points are collinear. This reduces the collision search to a line (of containing 2^n points). As the output points are uniformly distributed on this line, the birthday bound applies and a collision is expected after $\mathcal{O}(2^{n/2})$ steps.

If $\omega_{11}\omega_{25} = \omega_{15}\omega_{21}$, however, we cannot exploit degenerate partitions. Yet, it is still possible to attack the *plain* blockcipher-based instantiation; this time using parallel partitions. First note that we can easily create $\mu > 1$ parallel partitions (say $\mathscr{P}_{1:c,d,y_2}$, without loss of generality) by using a single ciphertext as we are just aiming for collisions in the ciphertexts, which are trivial to generate (i.e., choose a single y_2 and call D^2 μ times with μ different keys).

Turning the above observation into a collision attack is again based on creating collinear output points: We first fix both the partition $\mathscr{P}_{1:c,d,y_2}$ and a line $\mathscr{L}_{1:c,d,y_2;a} \in \mathscr{P}_{1:c,d,y_2}$. Then, we pick a random $d' \neq d$, query $D^2_{d'}(y_2)$ and obtain the corresponding unique $c' = D^2_{d'}(y_2)$ (now we are in a partition parallel to $\mathscr{P}_{1:c,d,y_2}$). Next, we select the a' value that corresponds to the line $\mathscr{L}_{1:c,d,y_2;a}$ in the newly generated partition (note $\mathscr{L}_{1:c,d,y_2;a} = \mathscr{L}_{1:c',d',y_2;a'}$). Finally, we obtain the unique corresponding b' and query $E^1_{b'}(a')$. The result lies on $\mathscr{L}_{1:c,d,y_2;a}$ thanks to the way we have chosen our queries. As the

output points created this way will all be collinear yet random (as the outputs of E^1 are random), collisions are expected with high probability after repeating the above procedure $2^{n/2}$ times (due to the birthday paradox). □

5.4.2 "DM"plified Version

In order to thwart the above-mentioned weaknesses, we suggest (without a security proof) using the underlying blockciphers in the Davies–Meyer mode, i.e., we feedforward the plaintext into the ciphertexts via (field) addition. Note that there is no need to change C^{pre}; the only modification is required in C^{post}.

Construction 5.4.2. *Let $E^1, E^2 \in \mathrm{Block}(n,n)$ be fixed randomly and independently sampled blockciphers. Define the double-call DBL blockcipher-based compression function $\mathsf{h}^{E^1,E^2}\colon \{0,1\}^{3n} \to \{0,1\}^{2n}$ (Definition 2.3.8 with $m = \kappa = n$ and $s = 2n$) by using the preprocessing function $C^{\mathsf{pre}}\colon \mathbb{F}_{2^n}^3 \to \left(\mathbb{F}_{2^n}^2\right)^2$, where*

$$C^{\mathsf{pre}} = \left(C_1^{\mathsf{pre}}, C_2^{\mathsf{pre}}\right), \quad C_1^{\mathsf{pre}}(a,b,c) = (a,b) \quad \textit{and} \quad C_2^{\mathsf{pre}}(a,b,c) = (c, ac+b)$$

and the postprocessing $C^{\mathsf{post}}\colon \mathbb{F}_{2^n}^5 \to \mathbb{F}_{2^n}^2$,

$$C^{\mathsf{post}}(a,b,c,y_1,y_2) = A \cdot \begin{pmatrix} a \\ c \\ a+y_1 \\ c+y_2 \\ (a+y_1)(c+y_2) \end{pmatrix}, \textit{ where } \quad A = \begin{pmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \omega_{14} & \omega_{15} \\ \omega_{21} & \omega_{22} & \omega_{23} & \omega_{24} & \omega_{25} \end{pmatrix}$$

is the same matrix given in Section 5.1.1.

5.5 Practical Considerations and Comparison

By design, our construction consists of a number of XOR operations (to implement the matrix multiplication) plus two multiplications in the finite field $\mathbb{F}_{2^n}$: one during the preprocessing and one during the postprocessing. In the following, we briefly present two ways of optimizing the performance of our construction in the iteration (we focus on the blockcipher-based instantiation and aim to see the performance of it; note that we do not make security claims for this case). Depending on the platform (and relative constraints), we may benefit from either method. A more in-depth performance analysis of our design is left mainly unexplored.

Firstly, we focus on reducing the overhead of the two (full) finite field multiplications; they consist of the major computational overhead compared to existing compression function constructions. Secondly, we attempt to run several key-scheduling units simultaneously to reduce the effect of multiple (and sequential) re-keying. The second approach is particularly important for the architectures that support the AES instruction-set (AES-NI) [9]. Indeed, as exploited by the recent comprehensive performance comparison (by Bos et al. [35]) of AES-driven DBL hash functions on modern architectures that support the AES-NI, reducing the cost of the key-scheduling is a key factor to obtain an efficient scheme.

9. AES key-scheduling requires roughly 1.5 times more cycles to compute than encryption when AES-NI is used [35].

Let us define $(a, b||c) \leftarrow (M_i, v_{i-1})$ (see Definition 2.2.1 for the notation) and look at the overall overhead by using the proposed matrix A from Section 5.1. It is not too difficult to see that we can run simultaneously the full finite field multiplications $y_1 y_2$ and ac that appear at the i'th and $(i+1)$'st iterations, respectively. Furthermore, at each compression function evaluation, two blockcipher calls can be made independently. The remaining overhead is simply a few XOR instructions required for the matrix multiplication (together with some loading and storing) that we can safely assume to be negligible. All in all, the performance in the Merkle–Damgård iterated hash function (when used with our compression function) is expected to be determined by two parallel blockcipher calls (along with the corresponding key-scheduling units) plus two parallel (full) finite field multiplications.

Here we describe our second optimization where our major goal is to reduce the effect of frequent key-scheduling. Let M_i denote the message to be compressed at the i'th iteration and let v_{i-1} be the state value that is output from the $(i-1)$'st compression function evaluation. We treat $v_{i-1} = (v_{i-1}^1 || v_{i-1}^2)$ (for $v_{i-1}^1, v_{i-1}^2 \in \{0,1\}^n$) and define $(a, b, c) \leftarrow (v_{i-1}^1, M_i, v_{i-1}^2)$. As the input block b, which is assigned to the message block M_i, is directly fed to the key-schedule of the first blockcipher call, we can run several key-schedules (only for E^1) in parallel for multiple iterations; hence we might reduce the overall cost.

Previous Proposals

The only published—and not badly broken—hash functions that use a double-call DBL blockcipher-based compression function in the same class as ours (making two parallel calls to an n-bit key, n-bit block blockcipher) are MDC-2 [37] and MJH [105]. Although MDC-2 only has minimal overhead (a small number of XOR operations) and calls the same blockcipher twice, its compression function only provides a collision resistance of $\Theta(2^{n/2})$ queries. The construction thus relies crucially on the iteration to improve its security. Even then, even if MDC-2 remains essentially unbroken in the iteration [90], the best security proof implies that we need $\Omega(2^{3n/5})$ queries [192] to find a collision. Similar to MDC-2, MJH can provide non-trivial collision resistance only in the iteration; currently the best proof shows that we need $\Omega(2^{2n/3-\log n})$ queries to find a collision. Its computational overhead consists of only a small number of XOR operations and a shift; moreover, the two blockcipher calls have the advantage of using the same key (allowing shared key-schedule).

When allowing for two calls to a blockcipher with $2n$-bit key, several proposals achieve (close to) optimal collision resistance [65, 76, 104, 106]. Some of these constructions, such as Abreast-DM, are built using a single blockcipher that is called twice in parallel with minimal overhead (a few XOR operations). Usually $2n$-bit key blockciphers are, both in software and hardware, more costly than the n-bit key ones as more rounds are needed to properly disperse the key. Nonetheless, the added cost is likely less than a full finite field multiplication.

There are also DBL compression functions that make only a single call to a blockcipher with $2n$-bit keys. Lucks [112] (see also [150]) gives a compression function that achieves near-optimal collision resistance in the iteration, and Stam [190, 191] gives a compression function (QPB-DBL) with near-optimal collision resistance. Lee and Steinberger [108] prove that we can replace the $3n$-to-n bit PuRF in the original construction with, instead, a cascade of two $2n$-to-n bit PuRFs while maintaining near-optimal collision resistance. This Lee–Steinberger–Stam construction has as overhead of two full finite field multiplications, so the computational cost is similar to that of our construction.

However, neither the PuRF calls nor the finite field multiplications can be made in parallel (yielding a latency double that of our construction).

Finally, there are DBL constructions (Definition 3.2.1) based on a $2n$-to-n bit primitive that make more than two calls (per compression function evaluation). An example is Nandi et al.'s construction [129] that makes *three* calls (and with only two XOR operations as overhead). Furthermore, several schemes proposed by Peyrin et al. [154] and Knudsen and Preneel [93] (with only several XOR operations as overhead) fall into this category. For Peyrin et al.'s compression functions, collision resistance has been established to be $\Theta(q^3/2^{2n})$ in the information-theoretic setting [180]; for the Knudsen–Preneel schemes, precise bounds for collision resistance are still open (see Chapter 6 for the details).

A Quick Glance at the Performance of Our Design Using AES-NI

Finally, following the methodology in [35] and instantiating the underlying blockciphers with AES-128 (with appropriate domain separation to obtain two distinct blockciphers), we can achieve a clear benchmark with almost all existing multi block-length hash function designs by using their framework. We first introduce some background material.

AES AES is a member of the Rijndael blockcipher suite [51]. It was standardized by the US National Institute of Standards and Technology (NIST) after a public competition similar to the one currently ongoing for SHA-3. AES operates on an internal state of 128 bits while supporting 128-, 192-, and 256-bit keys. The internal state is organized in a 4×4 array of 16 bytes, which is transformed by a round function N_r times. The number of rounds is $N_r = 10$ for the 128-bit key, $N_r = 12$ for the 192-bit key, and $N_r = 14$ for the 256-bit key variants. In order to encrypt, the internal state is initialized, then the first 128-bits of the key are XORed into the state, after which the state is modified $N_r - 1$ times according to the round function, followed by a slightly different final round (for the exact details see the AES specification [51]).

The AES Instruction Set (AES-NI) In the last decade, use of the *single instruction, multiple data* (SIMD) paradigm has become a general trend in computer architecture design. It enhances the speed of software implementations by off-loading the computational work to special units that operate on larger data types, thus improving overall throughput. In 1999, Intel introduced the streaming SIMD extensions (SSE), a SIMD instruction set extension to the x86 architecture.

One of the latest additions to these extensions is the AES instruction set [70] available in the 2010 Intel Core processor family based on the 32nm Intel micro-architecture named Westmere. This instruction set will also be supported by AMD in their next-generation CPU "Bulldozer". This instruction set consists of six new instructions. At the same time, a new instruction for performing carry-less multiplication was released in the `CLMUL` instruction set extension. We can summarize the new instructions as follows [70]:

- `AESENC` and `AESDEC` perform a single round of encryption, respectively decryption.
- `AESENCLAST` and `AESDECLAST` perform the last round of encryption, respectively decryption.
- `AESKEYGENASSIST` is used for generating the round keys used for encryption.
- `AESIMC` is used for converting the encryption round keys to a form usable for decryption using the

Algorithm	Primitive	Speed
Abreast-DM [102]	AES-256	11.21
Hirose-DBL [76]	AES-256	9.82
MDC-2 [37]	AES-128	10.00
MJH [105]	AES-128	7.45
QPB-DBL [190]	AES-256	14.12
Our Construction	AES-128	17.40

Table 5.2 – The performance comparison of certain DBL compression function designs that compress 128-bit message blocks and output 256-bit digest. The primitive employed and the achieved speed (in cycles per byte) using the AES instructions are shown in the second and third column, respectively.

Equivalent Inverse Cipher.

- `PCLMULQDQ` performs carry-less multiplication of two 64-bit operands to an 128-bit output.

Based on preliminary analysis, our (not very optimized) implementation achieves 17.40 cycles/byte (on an Intel Core i5 650 (3.20GHz)) using C intrinsics and the recent AES instruction set (AES-NI) extensions. A comparison of our performance results with some of those given in [35] is presented in Table 5.2. We leave a more detailed exploration of the performance of our construction as an open problem.

6 On the Security of Knudsen–Preneel Compression Functions

This chapter offers a new security analysis of the Knudsen–Preneel (KP) construction [91–93] (see also the background from Section 2.3.5) when the underlying compression functions are modeled as public random functions (PuRFs). Recall that Knudsen–Preneel compression functions make use of an $[r,k,d]$ linear error-correcting code over $\mathbb{F}_{2^e}$ (for $e>1$) to build a compression function from underlying blockciphers operating in the Davies–Meyer mode. From a security point of view, Knudsen and Preneel show that, in the complexity-theoretic setting, finding collisions takes time at least $2^{(d-1)n/2}$ for their constructions [93, Theorem 4]. Preimage resistance, however, is simply conjectured to be the square of the collision resistance.

In this chapter, we study both the KP-conjectured preimage resistance and the claimed collision resistance bounds: We describe new attacks, taking into account both query- *and* time-complexity (and to a lesser extent space-complexity). The relevant complexities (ignoring constants and factors that are polynomial in n) of our attacks, as well as the KP-parameters in question, are summarized in Table 6.1. We note that our attacks are not specific to the cases where the underlying PuRFs are $2n \to n$ or $3n \to n$. For instance, as we detail in Appendix C, they also work against the compression functions suggested by Knudsen and Preneel with PuRFs that compress $5n$-bit to n-bit [1].

Our secondary result is a preimage-resistance security proof for the Knudsen–Preneel compression functions. In Theorem 6.3.14, we determine a lower bound on the query-complexity of preimage-finding attacks, including attacks mounted by computationally unbounded adversaries. The theorem gives a concrete bound, and a related corollary (Corollary 6.3.15) provides an asymptotic assessment of the bound that is easier to grasp. In particular, it shows that the query-complexity of our new preimage attack is essentially optimal (up to the factors that are polynomial in n).

Because the lower bounds on the query-complexity serve as 'best case' lower bounds for the complexity of real-world attacks, we can conclude that our new preimage-finding attack is *optimal* whenever the time-complexity of our attack matches its query-complexity. This happens for 9 out of the 16 schemes: for the seven MDS schemes with $d=3$, and for codes $[8,5,3]_4$ and $[12,9,3]_4$. For the remaining seven schemes, we leave a gap between the information-theoretic lower bound and the real-life upper bound (i.e., the attack that we are able to achieve). For our optimized collision attacks, we can show that for 12 out of 16 suggested parameters, we can mount a collision attack whose time-complexity matches its query-complexity (ignoring constants and factors that are logarithmic

1. This parameter mimics MD4 and MD5 compression function.

Code	Compression Function	Collision Resistance			Preimage Resistance		
		New Results		KP Claim	New Results		KP Conjecture
		Our Attack Complexity		KP (Time) Lower	Our Attack Complexity		KP (Time) Lower
		Query	Time	Bound	Query	Time	Bound
$[r,k,d]_{2^e}$		$2^{kn/(3k-r)}$	Sec. 6.4	$2^{(d-1)n/2}$	$2^{rn/k}$	Sec. 6.3	$2^{(d-1)n}$
$[5,3,3]_4$	$(5+1)n \to 5n$	$2^{3n/4}$	$2^{3n/4}$	2^n	$2^{5n/3}$	$2^{5n/3}$	2^{2n}
$[8,5,3]_4$	$(8+2)n \to 8n$	$2^{5n/7}$	$2^{5n/7}$	2^n	$2^{8n/5}$	$2^{8n/5}$	2^{2n}
$[12,9,3]_4$	$(12+6)n \to 12n$	$2^{3n/5}$	$2^{3n/5}$	2^n	$2^{4n/3}$	$2^{4n/3}$	2^{2n}
$[9,5,4]_4$	$(9+1)n \to 9n$	$2^{5n/6}$	$2^{5n/6}$	$2^{3n/2}$	$2^{9n/5}$	$2^{11n/5}$	2^{3n}
$[16,12,4]_4$	$(16+8)n \to 16n$	$2^{3n/5}$	$2^{4n/5}$	$2^{3n/2}$	$2^{4n/3}$	$2^{7n/3}$	2^{3n}
$[6,4,3]_{16}$	$(6+2)n \to 6n$	$2^{2n/3}$	$2^{2n/3}$	2^n	$2^{3n/2}$	$2^{3n/2}$	2^{2n}
$[8,6,3]_{16}$	$(8+4)n \to 8n$	$2^{3n/5}$	$2^{3n/5}$	2^n	$2^{4n/3}$	$2^{4n/3}$	2^{2n}
$[12,10,3]_{16}$	$(12+8)n \to 12n$	$2^{5n/9}$	$2^{5n/9}$	2^n	$2^{6n/5}$	$2^{6n/5}$	2^{2n}
$[9,6,4]_{16}$	$(9+3)n \to 9n$	$2^{2n/3}$	$2^{2n/3}$	$2^{3n/2}$	$2^{3n/2}$	2^{2n}	2^{3n}
$[16,13,4]_{16}$	$(16+10)n \to 16n$	$2^{13n/23}$	$2^{20n/23}$	$2^{3n/2}$	$2^{16n/13}$	2^{2n}	2^{3n}
$[4,2,3]_8$	$(4+2)n \to 4n$	×	×	2^n	2^{2n}	2^{2n}	2^{2n}
$[6,4,3]_8$	$(6+6)n \to 6n$	$2^{2n/3}$	$2^{2n/3}$	2^n	$2^{3n/2}$	$2^{3n/2}$	2^{2n}
$[9,7,3]_8$	$(9+12)n \to 9n$	$2^{7n/12}$	$2^{7n/12}$	2^n	$2^{9n/7}$	$2^{9n/7}$	2^{2n}
$[5,2,4]_8$	$(5+1)n \to 5n$	×	×	$2^{3n/2}$	$2^{5n/2}$	2^{3n}	2^{3n}
$[7,4,4]_8$	$(7+5)n \to 7n$	$2^{4n/5}$	$2^{4n/5}$	$2^{3n/2}$	$2^{7n/4}$	$2^{9n/4}$	2^{3n}
$[10,7,4]_8$	$(10+11)n \to 10n$	$2^{7n/11}$	$2^{9n/11}$	$2^{3n/2}$	$2^{10n/7}$	2^{2n}	2^{3n}

Table 6.1 – Knudsen–Preneel constructions (cf. [93, Table V and VIII]) based on a $2n$-to-n and $3n$-to-n bit primitive (PuRF), their (incorrect) collision resistance claim, (incorrect) preimage resistance conjecture and our findings, are summarized. Non-MDS (for $\mathbb{F}_{2^2}$) and Watanabe-resistant parameters (for $\mathbb{F}_{2^3}$) are given in italics. The symbol × shows that our techniques are not applicable to the corresponding codes. Our attacks serve as an upper bound on the level of collision and preimage resistance.

in n). We leave the information-theoretic security proof for collision resistance as an open problem.

From a practical point of view, the time-complexities of our attacks are better than the previously best known attack (Knudsen and Preneel for preimages [93], Watanabe for collisions [205]) in every case but two, specifically when the code is either $[4,2,3]_8$ or $[5,2,4]_8$; in both cases we match the original preimage attack. Moreover, we show that this is optimal for $[4,2,3]_8$. A disadvantage of our attacks is their relatively high space-complexity, which is typically higher than that of previous attacks.

From a theoretical point of view, our attacks go well *below* the conjectured (for finding preimages) and the proven (for finding collisions) time-complexity lower bound. This demonstrates the incorrectness of the bounds claimed (for collision resistance) and conjectured (for preimage resistance) by Knudsen and Preneel. As a result, we conclude that with the possible exception of two of the proposed parameter sets (i.e., $[4,2,3]_8$ and $[5,2,4]_8$), the Knudsen–Preneel compression functions do not achieve the security level they were designed for.

The Preimage Attack in Detail We begin with the simple observation that $(0^a \,||\, x_1) \oplus (0^a \,||\, x_2)$ yields a string of the form $(0^a \,||\, X)$ (cf. the proof of Theorem 3.2.3). More generally, any linear combination of strings with the same pattern of fixed zero bits will yield a string of the same form. By restricting PuRF queries to strings with the same (blockwise) pattern we can optimize the *yield* of these queries (in particular yield(q), see Definition 2.3.1), or the maximum number of KP compression function

evaluations an adversary can compute for a given number of queries. This observation allows us to reduce the query-complexity of a preimage-finding attack to the (proven) bare minimum and, notably, also allows the attack to be non-adaptive.

When mounting our reduced-query attack against a KP construction with parameters $[r,k,d]_{2^e}$, the result is r lists with partial preimages (under each of the PuRF) and a full preimage is expected to 'hide' among these lists. That is to say, when we consider all possible combinations of partial preimages, some will correspond to a codeword and others will not. To reduce the time-complexity of our attack, we need to be able to find a codeword efficiently among all possibilities.

Finding full preimages from the lists of partial preimages is the core innovation of our attack. (Section 6.3 gives the details.) We exploit the fact that codewords in the *dual* code can be used to express relations among PuRF-inputs that correspond to a preimage. We also leverage known techniques for solving the generalized birthday problem (see e.g., [41, 47, 179, 200]) in order to prune the partial-preimage lists and, consequently, find a preimage for the compression function faster (than a naive approach or that of KP).

Here is a concrete example to give some quick, initial intuition. For the $[5,3,3]_4$ code given by Knudsen and Preneel [93, Section C], we observe that the (four) inputs to the PuRFs satisfy the equation $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$ (cf. Example 6.1.3). This relationship extends to the lists of partial preimages, so we begin by enumerating all solutions to this equation using the elements of the first four lists. This takes only quadratic time in the size of the lists because we can look for collisions $x_1 \oplus x_2 = x_3 \oplus x_4$ instead. We finalize the preimage search by checking whether or not any of these solutions can be extended in a way that their fifth component x_5 is in the fifth final list of partial preimages. If so, we have found a preimage.

Upon closer inspection, the relation used in this example corresponds to a codeword in the dual code. We will see that using a dual codeword of minimal distance allows the 'merge' stage of the attack (i.e., enumerating all solutions) to be the most efficient. Thus the minimum *dual* distance plays an important role in expressing the complexity of our attack. Furthermore, for maximum distance separable (MDS) codes, we show that one merge stage is sufficient, and we can finalize the preimage search straight away (as in the example above). For non-MDS codes a second stage of merging might be necessary to improve the overall complexity of the preimage-finding attack. In the non-MDS case expressing algebraically the time-complexity becomes a bit difficult; nevertheless, it is always easy to evaluate, given the generator matrix (of the linear code) that defines the compression function.

Finally, we note that our *attacks* should carry over to the blockcipher setting without much extra work; adapting the security proofs rigorously is less straightforward. The attacks also carry over to the iterative setting, provided the adversary gets to choose the initial vector; for a fixed IV we do not believe that our attacks work very effectively. However, a faster preimage attack on the compression function, as we present it, can be used in the Lai–Massey meet-in-the-middle attack [102].

The Collision Attacks in Detail For our improved collision attacks, we mainly follow in the footsteps of our preimage attack, yet with some subtle differences, e.g., to incorporate the idea underpinning Watanabe's collision attack (Algorithm 6.4.4). To this end, the *mise en place* in Section 6.4.1 provides a detailed mathematical characterization of the preprocessing function of the Knudsen–Preneel compression functions. As a first simple result, this allows us in Section 6.4.2 to revise the

attack of Watanabe in a way that slightly reduces the time requirements, yet significantly increases the number of collisions it can produce. More precisely, after an initial effort of $d2^n$, we can generate (up to) $2^{c(k-d)n}$ collisions in constant time (for $k > d$).

In our revised version of Watanabe's attack (see Section 6.4.2), we also arrive at a new, symbiotic collision-finding attack (Algorithm 6.4.9) with time-complexity $2^{dn/(d+1)}$. The attack is a combination of the ideas of Watanabe's attack and our preimage attack and works whenever $d \leq k$ (so the same two codes $[4,2,3]_8$ and $[5,2,4]_8$ as in Watanabe's case are excluded from our attack). Even more important is that if the inequality is strict; in other words, if $d < k$, the adversary can create further collisions (like our revised Watanabe attack) in *constant* time (up to $2^{c(k-d)n}$ collisions).

In Section 6.4.3 we introduce a novel parametrized information-theoretic collision attack (Algorithm 6.4.12). It turns out that the new symbiotic attack and the standard yield-based information-theoretic collision attack (Proposition 6.2.1) are both on opposite ends of the spectrum of parametrized attacks, yet optimality is typically achieved somewhere in the middle—with $[4,2,3]_8$ and $[5,2,4]_8$ as sole exceptions—yielding query-complexity $2^{kn/(3k-r)}$.

Our final contribution (presented in Sections 6.4.5 and 6.4.6) is a reduced-time variant of our optimized (parametrized) attack above. For this, we use the same ideas as in our preimage attack, but with a significant twist: This time we use the dual code of the *shortened* code to search for collisions. As a result, for 12 out of 16 (KP) suggested parameters, we can mount a collision attack whose time-complexity matches its query-complexity (ignoring constants and factors that are logarithmic in n). Even better, only for $[4,2,3]_8$ and $[5,2,4]_8$ are we unable to surpass the time-complexity of any prior attack we are aware of, for the rest we set new records.

6.1 The Knudsen–Preneel Hash Functions

Knudsen and Preneel [91, 92] introduce a family of hash functions employing linear error-correcting codes (we use the journal version [93] as our frame of reference). Although their work ostensibly targets blockcipher-based designs, the main technical thread of their work develops a transform that extends the range of an 'ideal' compression function (blockcipher-based, or not) in a manner that delivers some target level of security.

In fact, the KP transform is a special instance of a blockwise-linear scheme (Definition 3.2.2), in which the inputs to the underlying PuRFs are determined by a linear error-correcting code over a binary field with extension degree $e > 1$, i.e., $\mathbb{F}_{2^e}$, and with $\mathsf{C}^{\mathsf{post}}$ being the identity matrix over $\mathbb{F}_2^{rb \times rb}$ (corresponding to concatenating the PuRF outputs). The extension field $\mathbb{F}_{2^e}$ is represented as a sub-ring of the matrix ring (of dimension equaling the extension degree) over the base field. We formalize this by an injective ring homomorphism $\varphi : \mathbb{F}_{2^e} \to \mathbb{F}_2^{e \times e}$ and let $\bar{\varphi} : \mathbb{F}_{2^e}^{r \times k} \to \mathbb{F}_2^{re \times ke}$ be the component-wise application of φ and subsequent identification of $(\mathbb{F}_2^{e \times e})^{r \times k}$ with $\mathbb{F}_2^{re \times ke}$ (we use $\bar{\varphi}$ for matrices over $\mathbb{F}_{2^e}$ of arbitrary dimensions).

Definition 6.1.1 (Knudsen–Preneel transform)**.** Let $[r,k,d]$ be a linear error-correcting code over $\mathbb{F}_{2^e}$ with generator matrix $G \in \mathbb{F}_{2^e}^{k \times r}$. Let $\varphi : \mathbb{F}_{2^e} \to \mathbb{F}_2^{e \times e}$ be an injective ring homomorphism and let b be a positive divisor of e such that $ek > rb$. Then the Knudsen–Preneel compression function $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ equals $\mathsf{h} = \mathsf{BL}^b(\mathsf{C}^{\mathsf{pre}}, \mathsf{C}^{\mathsf{post}})$ (see Definition 3.2.2) with $\mathsf{C}^{\mathsf{pre}} = \bar{\varphi}(G^T)$ and $\mathsf{C}^{\mathsf{post}} = I_{rb}$.

If $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$, then $\mathsf{h}_n : \{0,1\}^{kcn} \to \{0,1\}^{rn}$ with $c = e/b$ is defined for all n for which b divides n. (As b is uniquely determined, given e and c, we often omit it.) Below is an example (Example 6.1.3) to help illustrate this formalism. For use of h in an iterated hash function, note that per invocation (of h) we can compress $(ek/b - r)$ message blocks (hence the requirement $ek > rb$ ensures actually compression is taking place).

Remark 6.1.2. In our definition of $\mathsf{KP}^b([r,k,d]_e)$, the given parameters do not uniquely determine the matrix $\mathsf{C}^{\mathsf{pre}}$ (and can lead to different compression functions). We list the three freedoms:

1. Choice of a linear error-correcting code of the given parameters $[r,k,d]_{2^e}$.
2. Choice of a generator matrix G for the chosen code.
3. Choice of an injective homomorphism φ.

We concentrate on the case $(b,e) \in \{(1,2),(2,4),(1,3)\}$ and then in particular on the 16 parameter sets given by Knudsen and Preneel. Our results—and those of Knudsen and Preneel—are largely unaffected by these three choices, with a possible exception for non-MDS codes, as we will see later. (Nevertheless, the actual cost of implementing the compression function depends on the choice of $\mathsf{C}^{\mathsf{pre}}$.) For completeness, Table 6.2 contains the systematic (see Section 3.1) generator matrices we consider, as given by Magma's BKLC (Best Known Linear Codes) routine.

Example 6.1.3. *Consider the compression function* $\mathsf{h} = \mathsf{KP}^1([5,3,3]_4)$*, in particular* h_n *(see Definition 3.2.2). This builds a* $6n \to 5n$ *compression function that uses five (independent) PuRFs* f^i*, for* $i = 1,\ldots,5$*, each mapping* $2n \to n$*. The relevant parameters are* $e = 2$*,* $c = 2$ *and* $b = 1$*. The preprocessing function* C^{pre} *of* h_n *is defined by a generator matrix* G *of the code* $[5,3,3]_4$*. We use the following* φ *as proposed by Knudsen and Preneel [93]:*

$$\varphi(0) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad \varphi(1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \varphi(w) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \quad \textit{and} \quad \varphi(w^2) = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix},$$

where $\mathbb{F}_{2^2} = \mathbb{F}_2(w)$ *and* $p(x) = x^2 + x + 1$ *is the minimal polynomial of* w*. Moreover, we choose the* G *as given in [93]. Hence, we reach*

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & w \\ 0 & 0 & 1 & 1 & w^2 \end{pmatrix} \quad \textit{and} \quad \mathsf{C}^{\mathsf{pre}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Therefore, given $W \in \{0,1\}^{6n}$*,* h_n *computes the digest* $Z \in \{0,1\}^{5n}$ *as follows:*

1. *Compute* $X \leftarrow (\mathsf{C}^{\mathsf{pre}} \otimes I_n) \cdot W$*;*
2. *For* $X = (x_1 || \ldots || x_5)$*, where* $x_i \in \{0,1\}^{2n}$ *(*$i = 1,\ldots,5$*), compute* $y_i = f^i(x_i)$*;*
3. *For* $Y = (y_1 || \ldots || y_5)$*, where* $y_i \in \{0,1\}^n$ *(*$i = 1,\ldots,5$*), output* $Z = (I_5 \otimes I_n) \cdot Y$*; equivalently* $Z = y_1 || \ldots || y_5$*.*

More intuitively, for an input $W \in \{0,1\}^{6n}$ *where* $W = w_1 || \ldots || w_6$ *and* $w_i \in \{0,1\}^n$ *for* $i = 1,\ldots,6$*, the*

Code	P^T
$\mathbb{F}_{2^2}-[5,3,3]$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & w & w^2 \end{pmatrix}$
$\mathbb{F}_{2^2}-[8,5,3]$	$\begin{pmatrix} 0 & w^2 & 0 & w & w^2 \\ w^2 & w & w^2 & 1 & 0 \\ w^2 & 1 & w & 0 & w^2 \end{pmatrix}$
$\mathbb{F}_{2^2}-[12,9,3]$	$\begin{pmatrix} w^2 & 1 & w & 0 & 0 & 0 & w^2 & w^2 & w^2 \\ w^2 & 0 & 1 & 1 & 1 & w & 0 & w & w \\ w^2 & w^2 & 0 & w^2 & 1 & w^2 & w^2 & w^2 & 1 \end{pmatrix}$
$\mathbb{F}_{2^2}-[9,5,4]$	$\begin{pmatrix} 0 & w & 1 & w & 0 \\ 1 & w^2 & 0 & 1 & w^2 \\ 1 & 1 & w^2 & 0 & 1 \\ w & w & w & w^2 & w^2 \end{pmatrix}$
$\mathbb{F}_{2^2}-[16,12,4]$	$\begin{pmatrix} 1 & w^2 & 0 & 1 & w^2 & w & w & w^2 & 1 & 0 & w^2 & 1 \\ 0 & w & w^2 & 1 & w & 1 & 0 & 1 & w & 1 & w^2 & w \\ w^2 & 1 & w & 1 & 0 & 1 & w & 1 & w^2 & w & 0 & 1 \\ w^2 & 0 & 1 & w^2 & w & w & w^2 & 1 & 0 & w^2 & 1 & 1 \end{pmatrix}$
$\mathbb{F}_{2^4}-[6,4,3]$	$\begin{pmatrix} w^8 & w^{12} & w^{14} & w^9 \\ w^{12} & w^{14} & w^9 & 1 \end{pmatrix}$
$\mathbb{F}_{2^4}-[8,6,3]$	$\begin{pmatrix} w^{10} & w^7 & w^8 & w^{12} & w^{14} & w^9 \\ w^7 & w^8 & w^{12} & w^{14} & w^9 & 1 \end{pmatrix}$
$\mathbb{F}_{2^4}-[12,10,3]$	$\begin{pmatrix} w^7 & w^{10} & w^3 & w^3 & w^{10} & w^7 & w^8 & w^{12} & w^{14} & w^9 \\ w^{10} & w^3 & w^3 & w^{10} & w^7 & w^8 & w^{12} & w^{14} & w^9 & 1 \end{pmatrix}$
$\mathbb{F}_{2^4}-[9,6,4]$	$\begin{pmatrix} w^4 & w^8 & w^6 & w^{11} & w^2 & w^{14} \\ w^{10} & w^3 & w^4 & w^7 & w^6 & w^{14} \\ w^8 & w^6 & w^{11} & w^2 & w^{14} & 1 \end{pmatrix}$
$\mathbb{F}_{2^4}-[16,13,4]$	$\begin{pmatrix} w^{14} & w^2 & w^{11} & w^6 & w^8 & w^4 & w^{12} & w^4 & w^8 & w^6 & w^{11} & w^2 & w^{14} \\ w^6 & w^7 & w^4 & w^3 & w^{10} & w^{13} & w^{13} & w^{10} & w^3 & w^4 & w^7 & w^6 & w^{14} \\ w^2 & w^{11} & w^6 & w^8 & w^4 & w^{12} & w^4 & w^8 & w^6 & w^{11} & w^2 & w^{14} & w^1 \end{pmatrix}$
$\mathbb{F}_{2^3}-[4,2,3]$	$\begin{pmatrix} w^3 & w^4 \\ w^4 & 1 \end{pmatrix}$
$\mathbb{F}_{2^3}-[6,4,3]$	$\begin{pmatrix} w^5 & w^5 & w^3 & w^4 \\ w^5 & w^3 & w^4 & 1 \end{pmatrix}$
$\mathbb{F}_{2^3}-[9,7,3]$	$\begin{pmatrix} 1 & w^4 & w^3 & w^5 & w^5 & w^3 & w^4 \\ w^4 & w^3 & w^5 & w^5 & w^3 & w^4 & w^1 \end{pmatrix}$
$\mathbb{F}_{2^3}-[5,2,4]$	$\begin{pmatrix} w^4 & w^3 \\ w^2 & w^3 \\ w^3 & 1 \end{pmatrix}$
$\mathbb{F}_{2^3}-[7,4,4]$	$\begin{pmatrix} w^4 & w^6 & w^4 & w^3 \\ w & w & w^2 & w^3 \\ w^6 & w^4 & w^3 & 1 \end{pmatrix}$
$\mathbb{F}_{2^3}-[10,7,4]$	$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 \\ 1 & w^2 & w^4 & w^6 & w & w^3 & w^5 \end{pmatrix}$

Table 6.2 – The codes (over $\mathbb{F}_{2^e}$ for $e \in \{2,3,4\}$) and the leading compression functions suggested by Knudsen–Preneel. Here, the generator matrix G is of the form $G = [I_k|P]$ and the table contains P^T. The Magma command BKLC($GF(2^e), r, k$) gives the resulting code $[r,k,d]_{2^e}$ where d is the best known value for the given parameters (matching with the codes given by KP).

inputs to the underlying PuRFs are formed as follows:

$$x_1 = (w_1 \,||\, w_2)\ ,\ x_2 = (w_3 \,||\, w_4)\ ,\ x_3 = (w_5 \,||\, w_6)\ ,\ x_4 = (w_1 \oplus w_3 \oplus w_5 \,||\, w_2 \oplus w_4 \oplus w_6)\ ,$$
$$x_5 = (w_1 \oplus w_3 \oplus w_4 \oplus w_6 \,||\, w_2 \oplus w_3 \oplus w_5 \oplus w_6).$$

The digest is obtained by the concatenation of the corresponding PuRF outputs y_i for all $i \in \{1,\ldots,5\}$.

Knudsen–Preneel's Security Claims

Knudsen and Preneel concentrate on the collision resistance of their compression function in the complexity-theoretic model. Under a fairly generous (but plausible) assumption, they essentially show that if $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$, then finding collisions in h_n takes time at least $2^{(d-1)n/2}$. For preimage resistance, Knudsen and Preneel do not give a corresponding theorem or an assumption, yet they do *conjecture* it to be the square of the collision resistance; that is, they conjecture that finding a preimage takes at least time $2^{(d-1)n}$. We first state the KP assumption for finding collisions, then we continue with their security claims both for collision and preimage resistance.

KP Assumption [93]: Assume that a collision for a Knudsen–Preneel compression function $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ has been found; say for h_n with the pair (W,W'). Thus, $\mathsf{h}_n(W) = \mathsf{h}_n(W')$. We call an underlying PuRF f^i *active*, if the colliding pair (W,W') results in $x_i \neq x_i'$ for the inputs of the corresponding PuRF f^i. Let N be the number of active PuRFs of which $N - \bar{t}$ can be attacked independently (i.e., collision search for $N - \bar{t}$ PuRFs can be mounted independently) and let $\bar{t}_{\min}$ be the minimum of all such $\bar{t}$ values. Then, it is assumed that collision-finding against h_n takes at least $2^{\bar{t}_{\min} n/2}$ time.

Proposition 6.1.4 (Knudsen–Preneel lower bounds [93])**.** *Let* $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *be given and consider* h_n *(with b dividing n). Then*

1. *(Conjectured) Finding preimages takes time at least* $2^{(d-1)n}$.
2. *(Under KP Assumption) Finding collisions takes time at least* $2^{(d-1)n/2}$.

In the proof of the above collision resistance claim, Knudsen and Preneel show (incorrectly) that $\bar{t}_{\min} = d-1$; their idea is the following. Because a collision requires the compression function inputs to be different, there has to be at least one active PuRF in the systematic portion. Specifically, there has to be an $i \in \{1,\ldots,k\}$ such that f^i is active. Consequently, there must exist at least $d-1$ active PuRFs in the non-systematic portion as the minimum distance of the underlying code is d.

Knudsen and Preneel's erroneous conclusion is that whatever the number of active PuRFs in the systematic portion, the number of active PuRFs in the non-systematic part remains at least $d-1$. Therefore, based on their assumption, the claimed lower bound holds as the PuRFs corresponding to the non-systematic portion cannot be attacked independently. However, as we detail in Section 6.4, it turns out that $\bar{t}_{\min} = 0$; simply note that whenever $d \leq k$, we can find d active PuRFs in the systematic portion, all of which can be attacked independently. Thus, the KP assumption leads to a lower bound of only one. Our collision attacks, as well as the one by Watanabe, are based on this simple observation. Regarding preimage resistance, this observation does not lead to a meaningful interpretation, and indeed Section 6.4 shows that the preimage resistance is independent of the number of active PuRFs.

Knudsen and Preneel also present two attacks, one for finding preimages [93, Proposition 3] and one for finding collisions [93, Proposition 4]. We only describe the preimage-finding attack in detail (see Algorithm 6.1.5); the collision-finding attack operates on the same principle. For all proposed parameters, the preimage attack runs (asymptotically in n) in time $\mathcal{O}(2^{(r-k)n})$ implying that for MDS codes, with $r-k = d-1$, it matches the conjectured lower bound.

Algorithm 6.1.5 (Knudsen–Preneel Preimage Attack)**.**

Input: $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$*, a systematic generator matrix* $G = (I_k|P)$*, a block-size* n *with* $b|n$ *(for* b n' *=*n*) and target digest* $Z \in \{0,1\}^{rn}$*.*

Output: *A preimage* $W \in \{0,1\}^{ken/b}$ *such that* $\mathsf{h}_n(W) = Z = z_1 || \ldots || z_r$ *for* $z_1, \ldots, z_r \in \{0,1\}^n$*.*

1. QUERY PHASE. *For* $i = 1, \ldots, k$*, let* $\mathcal{Q}[i] \subseteq \{0,1\}^{cn}$ *such that* $|\mathcal{Q}[i]| = 2^{rn/k}$*. Query* f^i *on all* $x_{i,j} \in \mathcal{Q}[i]$*. Keep a list* L_i *of all partial preimages* $x_{i,j} \in \mathcal{Q}[i]$ *satisfying* $f^i(x_{i,j}) = z_i$*.*
2. MERGE PHASE. *Create* $\tilde{L} = L_1 \times \ldots \times L_k$*.*
3. FINALIZATION. *For all* $X \in \tilde{L}$ *create the unique* W *corresponding to it and check whether it results in* $\mathsf{h}_n(W) = Z$*. If so, output* W*.*

Proposition 6.1.6 (Knudsen–Preneel attacks [93])**.** *Let* $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *be given and consider* h_n *(with* b *dividing* n*). Then (asymptotically in* n*)*

1. *Preimage-finding requires* $\mathcal{O}(\max(2^{n(r-k)}, 2^{rn/k}))$ *time and* $\mathcal{O}(2^{(r-k)n/k})$ n*-bit blocks of memory.*
2. *Collision-finding requires* $\mathcal{O}(\max(2^{n(r-k)/2}, 2^{(r+k)n/(2k)}))$ *time and* $\mathcal{O}(2^{(r-k)n/2k})$ n*-bit blocks of memory.*

Proof. For the preimage attack on h_n, given the target digest $Z \in \{0,1\}^{rn}$ (by construction $Z = y_1||\ldots||y_r$), the aim is to find $W \in \{0,1\}^{ken/b}$ such that $\mathsf{h}_n(W) = Z$ holds. Note that (due to restricting to a systematic generator matrix for $\mathscr{C}$) the first k different PuRFs $f^1, \ldots, f^k$ have mutually independent inputs. This allows us to find partial preimages for $y_1, \ldots, y_k$ independently. More precisely, (in QUERY PHASE) for all $i \in \{1, \ldots, k\}$, $2^{(r-k)\frac{n}{k}+n}$ (arbitrary) queries are asked to each f^i and those whose answers hit the target partial image y_i are stored in the lists L_i. As each target value is expected to be hit $2^{(r-k)\frac{n}{k}}$ times, the lists contain about $2^{(r-k)\frac{n}{k}}$ partial preimages. This step can be performed in $k2^{(r-k)\frac{n}{k}+n}$ PuRF evaluations with a memory requirement of $ek2^{(r-k)n/k}$ n-bits.

Now, in the MERGE PHASE, all the lists are combined in all possible ways to query the remaining PuRFs $f^{k+1}, \ldots, f^r$. Notice that as $f^1, \ldots, f^k$ correspond to the systematic portion of the code, any tuple $(x_1, \ldots, x_k)$ of partial preimages *uniquely* defines a tuple of queries $(x_{k+1}, \ldots, x_r)$ for the remaining $r-k$ PuRFs. Following this remark (and the fact that the elements in k lists are independent), as there are roughly $2^{(r-k)\frac{n}{k}}$ elements in each list, the overall number of possible combinations is $2^{(r-k)n}$. This is sufficient to expect a preimage for the remaining $(r-k)$ PuRFs, however to find it potentially requires $2^{(r-k)n}$ PuRF evaluations (and negligible additional memory).

For the collision attack, the same technique can be employed, but the number of possible combinations only needs to be $2^{(r-k)n/2}$ (in order to expect a collision for the remaining $(r-k)$ PuRFs). Consequently, in the first step of the attack, the attacker only needs to ask $2^{(r-k)\frac{n}{2k}+n} = 2^{(r+k)n/2k}$ queries to each f^i and after some bookkeeping the claim follows. □

6.2 Yield-based Information-Theoretic Attacks

The following proposition is a consequence of the yield-based information-theoretic attacks presented in Theorem 3.2.3. Briefly, it provides (a non-adaptive) information-theoretic attack against Knudsen–Preneel compression functions: Preimages can be expected after $2^{rn/k}$ queries and collisions can be expected after $2^{rn/(2k)}$ queries to the underlying primitives.

Proposition 6.2.1. *Let* $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *be given. Consider* h_n *with* b *dividing* n. *If* $[r,k,d]_{2^e}$ *is MDS or* $[r,k,d] \in \{[8,5,3],[12,9,3],[9,5,4],[16,12,4]\}$ *is given with a generator matrix* G *for* $[r,k,d]_4$, *as given by Magma's BKLC (Best Known Linear Code) routine (see Table 6.2); then preimages can be expected after* $2^{rn/k}$ *queries and collisions can be expected after* $2^{rn/(2k)}$ *queries.*

Proof. Let $[r,k,d]_{2^e}$ be an MDS code. For simplicity (and without loss of generality), let us assume that its generator matrix is systematic. We first investigate the claim on preimage resistance. The yield maximizing adversary from the proof of Theorem 3.2.3 asks at most q queries to the PuRFs corresponding to the systematic portion of the compression function. Hence, the expected number of partial preimages per PuRF there is $q/2^n$. Now consider the input x_i of any PuRF f^i from the non-systematic part (i.e., $i \in \{k+1,\ldots,r\}$). Our claim is that x_i is dependent on x_j for all $j \in \{1,\ldots,k\}$. This is equivalent to say that the column vector v_i in the generator matrix can be written as

$$v_i = \sum_{j=1}^{k} e_j v_j \quad \text{for} \quad e_j \neq 0 \text{ and } e_j \in \mathbb{F}_{2^e},$$

otherwise the condition that $[r,k,d]_{2^e}$ being an MDS is violated. Assume the contrary; then there exists at least one $j_0 \in \{1,\ldots,k\}$ such that $e_{j_0} = 0$ and

$$v_i - \sum_{\forall\, j \in \{1,\ldots,k\} - j_0} e_j v_j = e_{j_0} v_{j_0} = 0 \quad \text{where} \quad e_j \neq 0 \quad \text{for} \quad j \in \{1,\ldots,k\} - j_0\,.$$

Then the vectors v_i and v_j for $j \in \{1,\ldots,k\} - j_0$ are linearly dependent; hence the $k \times k$ sub-matrix containing these vectors is of rank strictly less than k, which is a contradiction. Hence, all of the inputs from the systematic part will contribute to evaluate the PuRF f^i. All in all, $(q/2^n)^k(1/2^n)$ inputs will hit the targeted value y_i. As there exist $r-k$ PuRFs in total in the non-systematic portion, the total expected number of preimages is $(q/2^n)^k(1/2^n)^{r-k} = q^k/2^{rn}$. So, preimages can be expected after $2^{rn/k}$ queries.

Similarly for collisions, the expected number of partial collisions per PuRF is $q^2/2^n$ for the systematic part; this yields a total of $\left(q^2/2^n\right)^k$ partially-colliding pairs. By the above, we have again that x_i is dependent on x_j for all $j \in \{1,\ldots,k\}$. Hence, $\left(q^2/2^n\right)^k(1/2^n)$ of the colliding pairs are expected to collide also under f^i. Running this argument iteratively for all $i \in \{k+1,\ldots,r\}$, we finally attain

$$\left(\frac{q^2}{2^n}\right)^k \left(\frac{1}{2^n}\right)^{r-k} = \frac{q^{2k}}{2^{rn}}$$

colliding pairs. So, after $q = 2^{rn/2k}$ queries collisions are output with a high probability.

Now suppose $[r,k,d] \in \{[8,5,3],[12,9,3],[9,5,4],[16,12,4]\}$ is given with a generator matrix G for $[r,k,d]_4$ (as given by Magma's BKLC routine, see Table 6.2). We prove our claim for the code $[8,5,3]_4$, the remaining cases are analogous. Moreover, we show the preimage attack in detail, the collision attack follows the same principle using the techniques in the preimage attack and the case for MDS codes. Assume that at most $2^{8n/5}$ queries are made to the underlying PuRFs corresponding to the systematic portion of the code. As above, we expect to end up with $2^{8n/5}/2^n = 2^{3n/5}$ partial preimages for f^i for all $i \in \{1,\ldots,5\}$. Consider the PuRF f^6 and its corresponding input x_6 obtained from the generator matrix given in Table 6.2. Clearly, x_6 is dependent only on x_2, x_4 and x_5.

Combining all possible preimage candidates corresponding to the PuRFs f^2, f^4 and f^5, we expect to obtain $(2^{3n/5})^3(1/2^n) = 2^{4n/5}$ partial preimages (hitting y_2, y_4, y_5 and y_6 in conjunction). We repeat the same analysis for f^7 and its corresponding input x_7. This time the relevant inputs from the systematic portion are x_1, x_2, x_3 and x_4. We have now $(2^{3n/5})^2$ additional candidates that were not used so far (i.e., x_1 and x_3) to be combined with $2^{4n/5}$ partial preimages that we have obtained earlier. All in all,

$$(2^{3n/5})^2 2^{4n/5}\left(\frac{1}{2^n}\right) = 2^n$$

preimage candidates are expected to hit the target digest besides y_8. As there is no freedom left from the systematic portion, we must complete the attack with the remaining 2^n candidates, which is sufficient to end up, with a high probability, with a single preimage output. Note that a similar attack work for the collision resistance with $q = 2^{8n/10}$ queries. By running a similar argument for the remaining KP-suggested non-MDS codes, we complete the proof. □

6.3 Revisiting the Preimage Resistance

6.3.1 Practical Preimage Attack Against $\mathsf{KP}^1([5,3,3]_4)$ in $\mathcal{O}(2^{5n/3})$ Time

Section 6.2 contains a theoretical attack with a minimal number of queries. This already allows us to turn Knudsen–Preneel's preimage attack from adaptive into non-adaptive one. In this section, we address reducing the time-complexity, while still keeping our attack non-adaptive.

Let $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ and n be given, where $b|n$ (and $bn' = n$, $c = e/b$ as before). Consider a non-adaptive preimage-finding adversary $\mathscr{A}$ against h_n, that tries to find a preimage for $Z \in \{0,1\}^{rn}$. For each $i = 1,\ldots,r$, $\mathscr{A}$ commits to query lists $\mathscr{Q}[i] \subseteq V_i = \{0,1\}^{cn}$ which, after querying, will result in a list of *partial* preimages

$$L_i = \{x_i \in \mathscr{Q}[i] | f^i(x_i) = z_i\}\,.$$

Because f^i is presumed random, we can safely assume that L_i is a set of approximately $|\mathscr{Q}[i]|/2^n$ randomly drawn elements of $\mathscr{Q}[i]$. Finding a preimage then becomes equivalent to finding an element $X = (x_1,\ldots,x_r)$ in the range of C^{pre} for which $x_i \in L_i$ for all $i = 1,\ldots,r$, $X \in \prod_{i=1}^r L_i$. Due to the linearity of C^{pre} at hand, the range-check itself is efficient for any given X, so a naive approach would be to simply exhaustively search $\prod_{i=1}^r L_i$. This would take time $\prod_{i=1}^r |L_i|$.

An improvement can already be obtained by observing that, when a systematic matrix G is used to generate the code, any element in $\bigoplus_{i=1}^k V_i$ can uniquely (and efficiently) be extended to some X in the range of C^{pre}. This lies at the heart of Knudsen and Preneel's adaptive attack and it can be adapted to the non-adaptive setting: for all elements in $\prod_{i=1}^k L_i$ compute its unique completion X and check whether for the remaining $i = k+1,\ldots,r$ the resulting $x_i \in L_i$. This reduces the time-complexity to $\prod_{i=1}^k |L_i|$; still, we can do better. Before describing our preimage attack in its full generality, we present an example of it applied to the compression function $\mathsf{h} = \mathsf{KP}^1([5,3,3]_4)$.

Claim 6.3.1. *For the compression function* $\mathsf{h} = \mathsf{KP}^1([5,3,3]_4)$*, preimages in* h_n *can be found in* $\mathcal{O}(2^{5n/3})$ *time (asymptotically in n) with a memory requirement of* $\mathcal{O}(2^{4n/3})$ *n-bit blocks.*

Proof. We refer to Example 6.1.3 for the details of G, φ and $\mathsf{C}^{\mathsf{pre}}$. Let target digest $Z = z_1||\ldots||z_5$ be given where $z_i \in \{0,1\}^n$ for $i = 1,\ldots,5$. Our aim is to find the PuRF inputs $x_i = (x_i^1||x_i^2) \in \{0,1\}^{2n}$

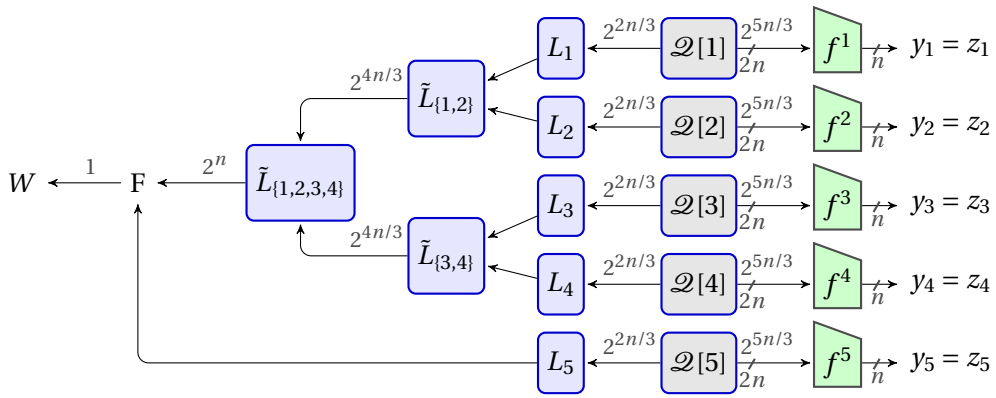

Figure 6.1 – Our preimage attack on $\mathsf{h}_n = \mathsf{KP}^1([5,3,3]_4)$ illustrated. The (unlabeled) inputs to $f^1,\ldots,f^5$ correspond to $(x_1^1,x_1^2),\ldots,(x_5^1,x_5^2)$. Here, F denotes the FINALIZATION phase.

with $x_i^1, x_i^2 \in \{0,1\}^n$ such that $f^i(x_i) = z_i$ holds for all $i = 1,\ldots,5$, and $X = (\mathsf{C}^{\mathsf{pre}} \otimes I_{n'}) \cdot W$ for some compression function input W (where X is comprised of the five x_i, i.e., $X = (x_1,\ldots,x_5)$). In this case, W is a preimage for Z. An outline of the attack is given in Figure 6.1, we proceed with the details.

The attack starts with what we call the QUERY PHASE. Specifically, for each $i = 1,\ldots,5$ and for all

$$x_i^1, x_i^2 \in 0^{n/6} \times \{0,1\}^{5n/6} ,$$

we query $f^i(x_i)$ and keep a list L_i of pairs that hit the target digest z_i. We set the first $n/6$ bits to zero for both x_i^1 and x_i^2 and exhaust all possibilities for the rest. As a result, a total of $2^{5n/3}$ queries are made (in $2^{5n/3}$ time) per PuRF, resulting in $|L_i| \approx 2^{5n/3}/2^n = 2^{2n/3}$ (as each query has probability 2^{-n} to hit its target). Because any triple $(x_1,x_2,x_3) \in L_1 \times L_2 \times L_3$ uniquely determines a preimage candidate W, finding a preimage is equivalent to finding an element $X \in L_1 \times \ldots \times L_5$ in the range of C^{pre}. To do this efficiently, we first identify the tuples (x_1,x_2,x_3,x_4) in the lists that can be complemented (not necessarily using $x_5 \in L_5$) to an element in the range. From the generator matrix G it can be seen that this complementation is possible if and only if $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$. So let us define

$$L_{\{1,2,3,4\}} = \{(x_1,x_2,x_3,x_4) \in L_1 \times L_2 \times L_3 \times L_4 \mid x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0\} .$$

We can construct $L_{\{1,2,3,4\}}$ efficiently by using a standard technique related to the generalized birthday problem [200]. It starts with the MERGE PHASE, where we create the lists $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ defined by

$$\tilde{L}_{\{1,2\}} = \{((x_1,x_2), x_1 \oplus x_2) \mid (x_1,x_2) \in L_1 \times L_2\} \quad \text{and} \quad \tilde{L}_{\{3,4\}} = \{((x_3,x_4), x_3 \oplus x_4) \mid (x_3,x_4) \in L_3 \times L_4\} ;$$

both are sorted on their second components. In the JOIN PHASE, we look for the collisions in their second components. As $|L_i| \approx 2^{2n/3}$, creating either $\tilde{L}$ takes (asymptotically in n) about $\mathcal{O}(n2^{4n/3})$ time and $\mathcal{O}(2^{4n/3})$ memory. (In general, the smallest $\tilde{L}$ is sorted and stored, and the other is used for collision check.) As $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ both have roughly $2^{4n/3}$ elements and they need to collide on $2n$ bits, of which $n/3$ bits are set to zero, the expected number of collisions is about (see Figure 6.1)

$$\frac{(2^{4n/3})^2}{2^{(2-1/3)n}} = 2^n = |L_{\{1,2,3,4\}}| .$$

We now have the collision list $L_{\{1,2,3,4\}}$ and all that needs to be done is to check, for each of its elements, whether the corresponding $x_5 \in L_5$. If this is the case, then (x_1, x_2, x_3) produces a valid preimage. This last step we call the FINALIZATION phase. It is clear that it cannot take much longer than it took to create $L_{\{1,2,3,4\}}$. Moreover, the expected number of preimages output is one. Note that $|L_{\{1,2,3,4\}}| \approx 2^n$ and $|L_5| \approx 2^{2n/3}$. Again, we need to check the correspondence on $2n$ bits, of which $n/3$ are set to zero. Hence, we do expect to find $2^{(1+2/3)n}/2^{(2-1/3)n} = 1$ preimage. Summing up the stepwise time and memory-complexities gives the desired result. □

Remark 6.3.2. The above attack takes a direct approach to finding all solutions of $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$ in the MERGE and JOIN PHASE of our preimage-finding algorithm. Even though this approach significantly improves the currently best known attack in terms of time-complexity, its memory-complexity is not the best we can achieve. Indeed, we can reduce the memory-complexity (asymptotically in n) to $\mathcal{O}(2^{2n/3})$ by applying the techniques proposed in the literature [41, 47, 200] (see Section 6.3.5 for the revised algorithm together with its application to other KP-suggested parameters).

6.3.2 Generic Attack Against MDS Schemes

In this section, we generalize our attack on the compression function $\mathsf{KP}^1([5,3,3]_4)$ to other Knudsen–Preneel compression functions. Note that the attack in Section 6.3.1 consists of four steps:

1. QUERY PHASE to generate the lists of partial preimages;
2. MERGE PHASE where two sets of lists are each merged exhaustively;
3. JOIN PHASE where collisions between the two merged lists are selected resulting in fewer partial preimages that are, however, preimage of a larger part of the target digest; and
4. FINALIZATION where the remaining partial preimages are filtered for being a full preimage.

As all Knudsen–Preneel compression functions are blockwise-linear, we are always able to run a non-adaptive attack. The question is whether we are always able to *efficiently* find a full preimage, given the lists of partial preimages L_i. For the $[5,3,3]_4$ example we could significantly reduce our workload because we found an easy-to-verify relation that the inputs to the PuRFs, thus in particular the elements in L_i, had to satisfy. The question is whether we were lucky to have found this relation, or whether there is a deeper underlying reason for it. Additionally, it is not immediately obvious that, even if we can somehow efficiently *merge* and *join*, there is an efficient way to *finalize*. We will proceed to describe the reason the attack worked and show that this naturally leads to a generalization to all Knudsen–Preneel schemes based on MDS codes. The slightly more complicated non-MDS case is discussed in Section 6.3.3.

The Core Observation From a high-level, our approach is simple: We first identify an index set $\mathscr{I} \subseteq \{1,\ldots,r\}$ defining a subspace $\bigoplus_{i\in\mathscr{I}} V_i$ for which the range of C^{pre} (when restricted to this subspace), is not surjective. By (blockwise-linear) construction, C^{pre} then maps to a subspace of $\bigoplus_{i\in\mathscr{I}} V_i$ of at most dimension $(|\mathscr{I}|-1)cn$ (over $\mathbb{F}_2$). As a consequence, we are able to prune significantly the total collection of candidate preimages in $\prod_{i\in\mathscr{I}} L_i$, keeping only those elements that are possibly in the range of C^{pre} restricted to $\bigoplus_{i\in\mathscr{I}} V_i$. In the following, we show how to *efficiently* find an index set $\mathscr{I}$, and how to *efficiently* prune.

It turns out that an important parameter determining the runtime of our preimage attack is $d^\perp$, the minimum distance of the dual code. Let χ be the function that maps $h \in \mathbb{F}_{2^e}^r$ to the set of indices of

non-zero entries in h. Thus, $\chi(h) \subseteq \{1,\dots,r\}$ and $|\chi(h)|$ equals the Hamming weight of the codeword. If $h \in \mathscr{C}^\perp$, then for $\mathscr{I} = \chi(h)$ we have precisely the property that allows us to prune $\prod_{i\in\mathscr{I}} L_i$ for partial preimages. The following proposition develops the key result for understanding our attack and the role the dual code plays in it. The interpretation follows the proposition.

Proposition 6.3.3. *Let* $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *and* $M \in \mathbb{F}_2^{e\times cr}$ *be given. Suppose that* $M = \bar{\varphi}(h^T)$ *for some* $h \in \mathbb{F}_{2^e}^r$*, then for all positive integers* $n' = n/b$ *it holds that*

$$(M \otimes I_{n'}) \cdot (\mathsf{C}^{\mathsf{pre}} \otimes I_{n'}) \cdot W = 0$$

for all $W \in \{0,1\}^{ken'}$ *if and only if*

$$h \in \mathscr{C}^\perp .$$

Proof. Let $h \in \mathbb{F}_{2^e}^r$ and $W \in \{0,1\}^{ken'}$ be given. Let $M = \bar{\varphi}(h^T)$ and recall that $\mathsf{C}^{\mathsf{pre}} = \bar{\varphi}(G^T)$ where G is a generator of $\mathscr{C}$. Then

$$\begin{aligned}(M \otimes I_{n'}) \cdot (\mathsf{C}^{\mathsf{pre}} \otimes I_{n'}) \cdot W &= (\bar{\varphi}(h^T) \otimes I_{n'}) \cdot (\bar{\varphi}(G^T) \otimes I_{n'}) \cdot W \\ &= ((\bar{\varphi}(h^T) \cdot \bar{\varphi}(G^T)) \otimes I_{n'}) \cdot W \\ &= (\bar{\varphi}((Gh)^T) \otimes I_{n'}) \cdot W .\end{aligned}$$

The statement that $(\bar{\varphi}((Gh)^T) \otimes I_{n'}) \cdot W = 0$ for all $W \in \{0,1\}^{ken'}$ is equivalent to the statement that $\bar{\varphi}((Gh)^T) = 0$. As φ is injective, this in turn is equivalent to $(Gh)^T = 0$. By definition, it holds that $Gh = 0$ if and only if $h \in \mathscr{C}^\perp$. □

In essence, this proposition tells us that if we are given a codeword $h \in \mathscr{C}^\perp$ and an element $X \in \mathbb{F}_2^{rcn}$ (to be input to the PuRFs), then X can only be in the range of C^{pre} if

$$(\bar{\varphi}(h^T) \otimes I_{n'}) \cdot X = 0 .$$

As the only parts of X relevant for this check are those lining up with the non-zero entries of h, we obtain that $\mathscr{I} = \chi(h)$ is what we are looking for. Indeed, an element $X \in \prod_{i\in\chi(h)} L_i$ can be completed to an element in the range of C^{pre} if and only if

$$(\bar{\varphi}(h^T) \otimes I_{n'}) \cdot (X + 0) = 0$$

(where we write $X + 0$ for embedding into the larger $\bigoplus_{i=1}^r V_i$). Efficient creation of

$$L_h = \left\{ X \in \prod_{i\in\chi(h)} L_i \mid (\bar{\varphi}(h^T) \otimes I_{n'}) \cdot (X + 0) = 0 \right\}$$

is done adapting standard techniques [41, 47, 179, 200] by splitting the codeword in two and looking for all collisions in the corresponding entries. Suppose that $h = h_0 + h_1$ with $\chi(h_0) \cap \chi(h_1) = \emptyset$, and define, for $j = 0, 1$

$$\tilde{L}_{h_j} = \left\{ (X_j, (\bar{\varphi}({h_j}^T) \otimes I_{n'}) \cdot (X_j + 0)) \mid X_j \in \prod_{i\in\chi(h_j)} L_i \right\} .$$

Then L_h consists of the elements $X_0 + X_1$ for which $(X_0, Y_0) \in \tilde{L}_{h_0}, (X_1, Y_1) \in \tilde{L}_{h_1}$, and $Y_0 = Y_1$. By sorting $\tilde{L}_{h_0}$ and $\tilde{L}_{h_1}$, the time-complexity of creating L_h is then roughly the maximum cardinality of

Algorithm 6.3.4 (Preimage attack against MDS-based schemes)**.**
Input: $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$, *block-size* n *with* $b|n$ *(for* $b\ n' = n$*) and target digest* $Z \in \{0,1\}^{rn}$.
Output: *A preimage* $W \in \{0,1\}^{ken/b}$ *such that* $\mathsf{h}_n(W) = Z = z_1||\ldots||z_r$.

1. QUERY PHASE. *Define*
$$\mathscr{X} = (\{0\}^{\frac{n}{b}-\frac{rn}{ek}} \times \{0,1\}^{\frac{rn}{ek}})^e$$
and, for $i = 1,\ldots,r$ *let* $\mathscr{Q}[i] = \mathscr{X} \subset V_i$. *Query* f^i *on all* $x_i \in \mathscr{Q}[i]$. *Keep a list* L_i *of all partial preimages* $x_i \in \mathscr{Q}[i]$ *satisfying* $f^i(x_i) = y_i = z_i$.

2. (FIRST) MERGE PHASE. *Find a non-zero codeword* $h \in \mathscr{C}^\perp$ *of minimum Hamming weight* $d^\perp$. *Let* $h = h_0 + h_1$ *with* $\chi(h_0) \cap \chi(h_1) = \emptyset$ *and of Hamming weights* $\lfloor d^\perp/2 \rfloor$ *and* $\lceil d^\perp/2 \rceil$, *respectively. Create, for* $j = 0,1$
$$\tilde{L}_{h_j} = \left\{ \left(X_j, (\bar{\varphi}({h_j}^T) \otimes I_{n'}) \cdot (X_j + 0)\right) \mid X_j \in \prod_{i \in \chi(h_j)} L_i \right\}$$
both sorted on their second component.

3. (FIRST) JOIN PHASE. *Create* L_h *consisting exactly of those elements* $X_0 + X_1$ *for which* $(X_0, Y_0) \in \tilde{L}_{h_0}, (X_1, Y_1) \in \tilde{L}_{h_1}$, *and* $Y_0 = Y_1$.

4. FINALIZATION. *For all* $X \in L_h$ *create the unique* W *corresponding to it and check whether it results in* $x_i \in L_i$ *for all* $i = 1,\ldots,r$. *If so, output* W.

the three sets $\tilde{L}_{h_0}, \tilde{L}_{h_1}$, and L_h involved. It therefore clearly pays to minimize the Hamming weights of h_0 and h_1, which is done by choosing a codeword $h \in \mathscr{C}^\perp$ of minimum distance $d^\perp$ and splitting it (almost) evenly (i.e., with $|\chi(h_0)| = \lfloor d^\perp/2 \rfloor$ and $|\chi(h_1)| = \lceil d^\perp/2 \rceil$). For an MDS code, we know that $d^\perp = k+1$. As a result, if h attains this, the map C^{pre} is injective when restricted to $\bigoplus_{i \in \chi(h)} V_i$ (or else the minimum distance would be violated). Hence, we know that all possible preimages given *all* the lists L_i are represented by the partial preimages contained in L_h. We can finalize by simply checking for all elements in $\tilde{L}_h$ whether its unique completion to $X \in \bigoplus_{i=1}^r V_i$ corresponds to $x_i \in L_i$ for all $i = 1,\ldots,r$. The complete preimage-finding algorithm is given in Algorithm 6.3.4.

Reinterpreting the Example Let us revisit our preimage attack example on $\mathsf{h} = \mathsf{KP}^1([5,3,3]_4)$ to see how it fits within the general framework. In the preimage attack example on $\mathsf{h} = \mathsf{KP}^1([5,3,3]_4)$, we more or less "magically" came up with the relation $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$. We can now appreciate that this constraint is really imposed by the dual codeword $h = (1\ 1\ 1\ 1\ 0) \in \mathbb{F}_{2^2}^5$. Thus our example corresponds to Algorithm 6.3.4 with $\chi(h_0) = \{1,2\}$ and $\chi(h_1) = \{3,4\}$ (leading to an even division). Note that we can also perform the attack based on other dual codewords of minimum distance, for instance $h = \left(1\ w\ w^2\ 0\ 1\right)$. In general, finding a minimum distance codeword might be more involved, but the dimensions are sufficiently small to allow for an exhaustive search.

Analysis of the Attack We proceed with the analysis of the generic preimage attack by providing the justifications of our claims and the overall time- and memory-complexities. We initially maintain $d^\perp$ in the expressions for future use (when discussing non-MDS codes). Moreover, we use α to denote the value r/k to keep the statement simple. We note that for $d = 3$ the overall time-complexity

Code		Query Complexity	Cardinalities related to our attack				Complexity Time	Memory
$[r,k,d]_{2^e}$	$d^\perp$	$\lvert\mathcal{Q}[i]\rvert$	$\lvert L_i\rvert$	$\lvert\tilde{L}_{h_0}\rvert$	$\lvert\tilde{L}_{h_1}\rvert$	$\lvert L_h\rvert$	Theorem 6.3.5	
$[5,3,3]_4$	4	$2^{5n/3}$	$2^{2n/3}$	$2^{4n/3}$	$2^{4n/3}$	2^n	$2^{5n/3}$	$2^{4n/3}$
$[6,4,3]_{16}$	5	$2^{3n/2}$	$2^{n/2}$	2^n	$2^{3n/2}$	2^n	$2^{3n/2}$	2^n
$[8,6,3]_{16}$	7	$2^{4n/3}$	$2^{n/3}$	2^n	$2^{4n/3}$	2^n	$2^{4n/3}$	2^n
$[12,10,3]_{16}$	11	$2^{6n/5}$	$2^{n/5}$	2^n	$2^{6n/5}$	2^n	$2^{6n/5}$	2^n
$[9,6,4]_{16}$	7	$2^{3n/2}$	$2^{n/2}$	$2^{3n/2}$	2^{2n}	2^{2n}	2^{2n}	$2^{3n/2}$
$[16,13,4]_{16}$	14	$2^{16n/13}$	$2^{3n/13}$	$2^{21n/13}$	$2^{21n/13}$	2^{2n}	2^{2n}	$2^{21n/13}$
$[4,2,3]_8$	3	2^{2n}	2^n	2^n	2^{2n}	2^n	2^{2n}	2^n
$[6,4,3]_8$	5	$2^{3n/2}$	$2^{n/2}$	2^n	$2^{3n/2}$	2^n	$2^{3n/2}$	2^n
$[9,7,3]_8$	8	$2^{9n/7}$	$2^{2n/7}$	$2^{8n/7}$	$2^{8n/7}$	2^n	$2^{9n/7}$	$2^{8n/7}$
$[5,2,4]_8$	3	$2^{5n/2}$	$2^{3n/2}$	$2^{3n/2}$	2^{3n}	2^{2n}	2^{3n}	$2^{3n/2}$
$[7,4,4]_8$	5	$2^{7n/4}$	$2^{3n/4}$	$2^{3n/2}$	$2^{9n/4}$	2^{2n}	$2^{9n/4}$	$2^{3n/2}$
$[10,7,4]_8$	8	$2^{10n/7}$	$2^{3n/7}$	$2^{12n/7}$	$2^{12n/7}$	2^n	2^{2n}	$2^{12n/7}$

Table 6.3 – An overview of the list cardinalities and computational complexity of preimage attacks on the Knudsen–Preneel compression functions based on MDS codes.

simplifies to $\mathcal{O}(2^{\alpha n})$, asymptotically in n, which coincides with the query-complexity. The proof of Theorem 6.3.5 (together with that of Theorem 6.3.7) is given in Section 6.3.4. Table 6.3 contains an overview for the various cardinalities and complexities for all the compression functions that are based on MDS codes.

Theorem 6.3.5. *Let* $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *be given and let* $d^\perp$ *be the minimum distance of the dual code of* $\mathscr{C}$*. Suppose* $\mathscr{C}$ *is MDS and consider the preimage attack described in Algorithm 6.3.4 run against* h_n *by using* $q = 2^{\alpha n}$ *queries for* $\alpha = r/k$ *(*$\lvert\mathcal{Q}[i]\rvert = 2^{rn/k}$*). Then the expected number of preimages is equal to one and the expectations for the internal list sizes are:*

$$\lvert L_i\rvert = 2^{(\alpha-1)n}\,, \quad \lvert L_h\rvert = 2^{(d^\perp(\alpha-1)-\alpha)n}\,, \quad \lvert\tilde{L}_{h_0}\rvert = 2^{\lfloor\frac{d^\perp}{2}\rfloor(\alpha-1)n}\,, \quad \lvert\tilde{L}_{h_1}\rvert = 2^{\lceil\frac{d^\perp}{2}\rceil(\alpha-1)n}\,.$$

The average case time-complexity of the algorithm is a small multiple of

$$\max\left(q, \lvert\tilde{L}_{h_1}\rvert, \lvert L_h\rvert\right)\,,$$

whereas the memory-complexity is $\lvert\tilde{L}_{h_0}\rvert$ *(expressed in the number of* cn*-bit blocks where* $e = bc$*). For* $d = 3$ *and substituting* $d^\perp = k+1$ *the time-complexity simplifies to* $\mathcal{O}(2^{\alpha n})$*, asymptotically in* n*, which is optimal (up to a small constant).*

In our attack, we set the number of queries as suggested by a yield-based bound (Proposition 6.2.1). Hence, as long as this first querying phase dominates, we know that our attack is optimal, as in the case, for example, against $\mathsf{KP}^1([5,3,3]_4)$ (see also Table 6.3). Note that the memory requirements of the preimage-finding algorithm given in Algorithm 6.3.4 can be further reduced using the techniques introduced in [47, 200]. We investigate a more space-efficient variant of our attack (without violating the time-complexity) in Section 6.3.5.

Algorithm 6.3.6 (Preimage attack against non-MDS-based schemes)**.**

Input: $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$*, block-size* n *with* $b|n$ *and target digest* $Z \in \{0,1\}^{rn}$*.*

Output: *A preimage* $W \in \{0,1\}^{ken/b}$ *such that* $\mathsf{h}_n(W) = Z = z_1||\ldots||z_r$*.*

1. QUERY PHASE. *As in Algorithm 6.3.4.*
2. FIRST MERGE PHASE. *As in Algorithm 6.3.4.*
3. FIRST JOIN PHASE. *As in Algorithm 6.3.4.*
4. SECOND MERGE PHASE. *Find a codeword* $h' \in \mathscr{C}^\perp \backslash \mathbb{F}_{2^e} h$ *of minimum Hamming weight (possibly exceeding* $d^\perp$*). Let* $h' = h'_0 + h'_1$ *with*

$$\chi(h'_0) \cap \chi(h'_1) = \emptyset \ , \ \chi(h'_1) \cap \chi(h) = \emptyset$$

and of Hamming weights yet to be determined. Create

$$\tilde{L}_{h'_0} = \left\{ \left(X_0, (\bar{\varphi}({h'_0}^T) \otimes I_{n'}) \cdot (X_0 + 0) \right) \mid X_0 \in L_h + \prod_{i \in \chi(h'_0) \backslash \chi(h)} L_i \right\}$$

$$\tilde{L}_{h'_1} = \left\{ \left(X_1, (\bar{\varphi}({h'_1}^T) \otimes I_{n'}) \cdot (X_1 + 0) \right) \mid X_1 \in \prod_{i \in \chi(h'_1)} L_i \right\} .$$

5. SECOND JOIN PHASE. *Create* $L_{h'}$ *consisting exactly of those elements* $X_0 + X_1$ *for which* $(X_0, Y_0) \in \tilde{L}_{h'_0}, (X_1, Y_1) \in \tilde{L}_{h'_1}$*, and* $Y_0 = Y_1$*.*
6. FINALIZATION. *For all* $X \in L_{h'}$ *create the unique* W *corresponding to it and check whether it results in* $x_i \in L_i$ *for all* $i = 1, \ldots, r$*. If so, output* W*.*

6.3.3 Generic Attack Against Non-MDS Schemes

For non-MDS codes, we try to mount the preimage attack given by Algorithm 6.3.4, but in the FINALIZATION we encounter a problem. As $d^\perp < k+1$ for non-MDS codes, the map C^{pre} restricted to $\bigoplus_{i \in \chi(h)} V_i$ is no longer injective and we can no longer reconstruct a unique W corresponding to some $X \in L_h$. There are two possible solutions to this problem. One is to simply merge as yet unused lists L_i into L_h until reconstruction does become unique.

However, a more efficient approach is to perform a second stage of merging and joining. In Algorithm 6.3.6, we simply paste in extra MERGE and JOIN phases in order to maintain the low complexity. We have only included one extra merge-join phase for non-MDS codes. For the parameters proposed by Knudsen and Preneel, this always suffices. For other parameters, possibly extra merge-join phases are required before full rank is achieved, we did not investigate this. We present concrete preimage-finding attack against $\mathsf{h} = \mathsf{KP}^1([8,5,3]_{2^2})$ at the end of this subsection.

Analysis of the Attack Although the addition of one extra round of MERGEing and JOINing sounds relatively simple, the analysis of it is slightly tedious, mainly because the first joining creates some asymmetry between the cardinality of the lists (that was not present for the MDS case). We note that in Theorem 6.3.7, the value j for which T_1 attains its minimum really only has two choices; but its algebraic optimization would not ease readability and would obscure the underlying meaning. Nevertheless, for the codes suggested by Knudsen and Preneel, it is always easy to determine the

Code		Cardinalities related to our attack						Complexity	
$[r,k,d]_{2^e}$	$d^\perp$	$\lvert\mathcal{Q}[i]\rvert$	$\lvert\tilde{L}_{h_0}\rvert$	$\lvert\tilde{L}_{h_1}\rvert$	$\lvert L_h\rvert$	$\max\left(\lvert\tilde{L}_{h'_0}\rvert,\lvert\tilde{L}_{h'_1}\rvert\right)$	$\lvert L_{h'}\rvert$	**Time**	**Memory**
$[8,5,3]_4$	4	$2^{8n/5}$	$2^{6n/5}$	$2^{6n/5}$	$2^{4n/5}$	$2^{7n/5}$	2^n	$2^{8n/5}$	$2^{6n/5}$
$[12,9,3]_4$	7	$2^{4n/3}$	2^n	$2^{4n/3}$	2^n	$2^{4n/3}$	2^n	$2^{4n/3}$	2^n
$[9,5,4]_4$	4	$2^{9n/5}$	$2^{8n/5}$	$2^{8n/5}$	$2^{7n/5}$	$2^{11n/5}$	2^{2n}	$2^{11n/5}$	$2^{8n/5}$
$[16,12,4]_4$	11	$2^{4n/3}$	$2^{5n/3}$	2^{2n}	$2^{7n/3}$	$2^{7n/3}$	2^{2n}	$2^{7n/3}$	$2^{5n/3}$

Table 6.4 – An overview of the list cardinalities and computational complexity of preimage attacks on the Knudsen–Preneel compression functions based on *non*-MDS codes.

resulting complexities due to the small parameters in question.

Because it is less clear from the theorem what the actual cardinalities will end up being (and consequently which step will be dominating), Table 6.4 summarizes the relevant quantities for the four non-MDS based compression functions $\mathsf{KP}^1([r,k,d]_4)$ suggested by Knudsen and Preneel [93]. Only for the $[9,5,4]_4$ code does the second stage dominate the overall runtime.

Theorem 6.3.7. *Let $[r,k,d] \in \{[8,5,3],[12,9,3],[9,5,4],[16,12,4]\}$ be given with a generator matrix G for $[r,k,d]_4$ (as given by Magma's BKLC routine); let $d^\perp$ be the minimum distance of the dual code of $\mathscr{C}$. For $\mathsf{h} = \mathsf{KP}^1([r,k,d]_4)$ consider the preimage attack described in Algorithm 6.3.6 run against h_n by using $q = 2^{\alpha n}$ queries for $\alpha = r/k$ ($\lvert\mathcal{Q}[i]\rvert = 2^{rn/k}$). Then, the expected number of preimages is equal to one, and the expectations for the internal list sizes for the first merge-join are as before (see Theorem 6.3.5) and for the second merge-join phase they are*

$$\max(\lvert\tilde{L}_{h'_0}\rvert,\lvert\tilde{L}_{h'_1}\rvert) \le 2^{T_1 n}, \quad \min(\lvert\tilde{L}_{h'_0}\rvert,\lvert\tilde{L}_{h'_1}\rvert) \le 2^{T_2 n}, \quad \lvert L_{h'}\rvert \le 2^{(r-k-2)n},$$

where

$$T_1 = \min_{j\in\{0,\ldots,k-d^\perp+1\}} \left(\max\{(k-d^\perp+2-j)(\alpha-1),((j+d^\perp)(\alpha-1)-\alpha)\}\right) \quad \textit{and} \quad T_2 = k(\alpha-1)+\alpha-2-T_1.$$

The expected time-complexity of the algorithm is a small constant multiple of

$$\max\left(q,\lvert\tilde{L}_{h_1}\rvert,\lvert L_h\rvert,\lvert\tilde{L}_{h'_0}\rvert,\lvert\tilde{L}_{h'_1}\rvert,\lvert L_{h'}\rvert\right),$$

whereas the expected memory-complexity is $\max\left(\lvert\tilde{L}_{h_0}\rvert,\min(\lvert\tilde{L}_{h'_0}\rvert,\lvert\tilde{L}_{h'_1}\rvert)\right)$ *(expressed in cn-bit blocks).*

Choice of Code Our attacks against the four non-MDS codes are based on the generator matrix given by Magma's BKLC routine. It is conceivable that different, non-equivalent codes perform differently under our attack. Most importantly, they might not have the same $d^\perp$, which will certainly change some of the cardinalities involved in our attack. Although this does not automatically mean the attack becomes faster or slower, it is certainly a possibility. We note that there is a trivial bound $d^\perp \le k$ (or else the code would be MDS), but in none of the four cases did we achieve this bound.

Additionally, even for non-equivalent codes with identical dual distance, slight variations in the runtime are conceivable. In the analysis upon which Theorem 6.3.7 is based, we conservatively assume

that, in the second merge, all remaining lists take part in the merge. If the second case dominates, however, a second codeword of minimum weight, or a low-weight codeword with significant overlap with the first, might reduce the listed complexity.

Finding Preimages for $\mathsf{h} = \mathsf{KP}^1([8,5,3]_{2^2})$ in $\mathcal{O}(2^{8n/5})$ Time

To illustrate our preimage attack on non-MDS schemes, we use $\mathsf{h} = \mathsf{KP}^1([8,5,3]_4)$ as an example. As in the $\mathsf{KP}^1([5,3,3]_4)$ case (Section 6.3.1), we are able to mount a preimage attack with almost optimal time-complexity.

Claim 6.3.8. *For the compression function* $\mathsf{h} = \mathsf{KP}^1([8,5,3]_4)$ *preimages can be found, asymptotically in* n*, using* $\mathcal{O}(2^{8n/5})$ *time and* $\mathcal{O}(2^{6n/5})$ *memory (expressed in* $2n$*-bits).*

Proof. In this case, we are building a $10n \to 8n$ compression function that uses eight calls to respective $2n \to n$ ideal primitives. In this attack, given the target digest $Z = z_1||\ldots||z_8$, we aim to find PuRF inputs $x_i = (x_i^1||x_i^2) \in \{0,1\}^{2n}$ such that $f^i(x_i) = z_i$ for all $i = 1,\ldots,8$, and $X = (\mathsf{C}^{\mathsf{pre}} \otimes I_{n'}) \cdot W$ for some compression function input W (where X is comprised of the eight x_i values). In this case W is a preimage for Z.

Below is a systematic generator matrix for the code $[8,5,3]_{2^2}$ that we use as a frame reference (note that the choice of this matrix does not affect the runtime of our attack as long as $d^\perp = 4$):

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & w^2 & w^2 \\ 0 & 1 & 0 & 0 & 0 & w^2 & w & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & w^2 & w \\ 0 & 0 & 0 & 1 & 0 & w & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & w^2 & 0 & w^2 \end{pmatrix}.$$

By using this G and φ from Example 6.1.3 we can construct the corresponding $\mathsf{C}^{\mathsf{pre}}$.

In the QUERY PHASE, we ask queries (to each PuRF) of a special form and keep the partial preimage lists to be used in the later stages of the algorithm. More precisely, we choose the queries of the form

$$x_i^1, x_i^2 \in 0^{n/5} \times \{0,1\}^{4n/5}$$

to evaluate $f_i(x_i^1, x_i^2)$ and keep a list L_i of pairs that hit the target partial digest z_i for all $i = 1,\ldots,8$. As a result, $2^{8n/5}$ elements are queried per PuRF (in $\mathcal{O}(2^{8n/5})$ time), which produces an approximate partial preimage size of $|L_i| \approx 2^{3n/5}$ (as each partial digest is assumed to be hit with a probability of 2^{-n}). Next, we make use of the dual codeword

$$h = \begin{pmatrix} 0 & w^2 & 0 & w & w^2 & 1 & 0 & 0 \end{pmatrix} \in \mathbb{F}_{2^2}^8$$

in the FIRST MERGE-JOIN PHASE so as to efficiently prune the irrelevant preimage candidates. This is done in accordance with the techniques detailed in Sections 6.3.2 and 6.3.3. As a result, we achieve the following relations (defined by the above h) between the inputs of the PuRFs f^2, f^4, f^5 and f^6:

$$x_2^2 \oplus x_4^1 \oplus x_4^2 \oplus x_5^2 = x_6^1 \quad \text{and} \quad x_2^1 \oplus x_2^2 \oplus x_4^1 \oplus x_5^1 \oplus x_5^2 = x_6^2 .$$

The merging is performed with the conditions $\chi(h_0) = \{2,4\}$ and $\chi(h_1) = \{5,6\}$. Moreover, we have $h_0 = \begin{pmatrix} 0 & w^2 & 0 & w & 0 & 0 & 0 & 0 \end{pmatrix}$ and $h_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & w^2 & 1 & 0 & 0 \end{pmatrix}$. We now create the lists $\tilde{L}_{h_j}$, for $j = 0,1$:

$$\tilde{L}_{h_j} = \left\{ \left(X_j, (\bar{\varphi}(h_j) \otimes I_{n'}) \cdot (X_j + 0) \right) \mid X_j \in \sum_{i \in \chi(h_j)} L_i \right\}$$

both sorted again on their second components. More specifically, we create the lists of the form:

$$\tilde{L}_{h_0} = \left\{ (x_2, x_4), \left((x_2^2 \oplus x_4^1 \oplus x_4^2) || (x_2^1 \oplus x_2^2 \oplus x_4^1 \right) \mid (x_2, x_4) \in L_2 \times L_4 \right\} ,$$

$$\tilde{L}_{h_1} = \left\{ (x_5, x_6), \left((x_5^2 \oplus x_6^1) || (x_5^1 \oplus x_5^2 \oplus x_6^2) \right) \mid (x_5, x_6) \in L_5 \times L_6 \right\} .$$

Because $|L_i| \approx 2^{3n/5}$, creating $\tilde{L}_{h_j}$ takes roughly $\mathcal{O}(n2^{6n/5})$ time and $\mathcal{O}(2^{6n/5})$ memory. The lists $\tilde{L}_{h_j}$ contribute roughly $2^{12n/5}$ elements in total for the First Join Phase, which collide on $2n$ bits (in the second component of the lists $\tilde{L}_{h_j}$), of which $2n/5$ bits are set to zero; hence

$$|L_h| \approx \frac{2^{12n/5}}{2^{(2-2/5)n}} = 2^{4n/5} .$$

Collision-finding can be performed in conjunction with the First Merge Phase in $\mathcal{O}(n2^{6n/5})$ time and $\mathcal{O}(2^{6n/5})$ memory. Now, by using the dual codeword

$$h' = \begin{pmatrix} w^2 & w & w^2 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \in \mathbb{F}_{2^2}^8 ,$$

we perform another Merge-Join Phase (as we cannot generate a unique preimage W using the elements contained in L_h and using h' is more efficient in terms of time-complexity). In order to further optimize the time-complexity, we take $\chi(h'_0) = \{1,2,4\}$ and $\chi(h'_1) = \{3,7\}$ to create the lists $\tilde{L}_{h_j'}$, for $j = 0,1$ (note that this is the optimal solution, considering the asymmetry between the list sizes). Projected to the L_2 and L_4 components, we obtain only $2^{4n/5}$ possibilities in L_h corresponding to L_2 and L_4. Each of he remaining lists L_1, L_3 and L_7, however, contains roughly $2^{3n/5}$ elements. Therefore,

$$|\tilde{L}_{h'_0}| \approx 2^{7n/5} \quad \text{and} \quad |\tilde{L}_{h'_1}| \approx 2^{6n/5} .$$

As both lists contribute roughly $2^{13n/5}$ elements that need to collide on $2n$ bits, of which $2n/5$ bits are set to zero, we have

$$|L_{h'}| = \frac{2^{13n/5}}{2^{(2-2/5)n}} = 2^n .$$

Collision-finding can be performed in conjunction with the First Merge-Join Phase in $\mathcal{O}(n2^{7n/5})$ time and $\mathcal{O}(2^{6n/5})$ memory.

We have 2^n preimage candidates W' satisfying $C_6^{\mathsf{pre}}(W') = x_6$ and $C_7^{\mathsf{pre}}(W') = x_7$. So, for the Finalization, there is a list of 2^n partial preimages on all but one output. So, we expect one of these partial preimages to be part of list L_8 and to be a full collision; hence this partial preimage creates a preimage. This step can be performed by simply doing another membership-check with the list L_8. The time-complexity of this step (which is $\mathcal{O}(2^n)$) is dominated by the previous stages of the algorithm. Therefore, summing up the stated complexities results in a time-complexity of $\mathcal{O}(2^{8n/5})$ (due to Query Phase) with a memory requirement of $\mathcal{O}(2^{6n/5})$ $2n$-bits. □

6.3.4 Proof of Theorems 6.3.5 and 6.3.7

We proceed step by step to prove Theorems 6.3.5 and 6.3.7. The first three steps are common for the MDS and non-MDS cases (where for MDS codes $d^\perp = k+1$). The remaining steps are treated separately. In the complexity estimations below, we concentrate on expected values and largely ignore (the effects of) factors that are polynomial in n; memory requirements are measured in multiples of cn-bit blocks.

QUERY PHASE. The time-complexity of this step is simply $2^{\alpha n}$ PuRF evaluations for $\alpha = r/k$ as $q = 2^{\alpha n}$. Per L_i we need $|L_i| \approx q/2^n = 2^{(\alpha-1)n}$ memory.

(FIRST) MERGE PHASE. The main computational requirement of this step is the generation of the lists $\tilde{L}_{h_j}$ for $j = 0, 1$. The time required for generating $\tilde{L}_{h_0}$ and $\tilde{L}_{h_1}$ is essentially equal to their respective sizes, namely $|L_i|^{|\chi(h_0)|}$ and $|L_i|^{|\chi(h_1)|}$. We left i unspecified, as all the L_i should be about the same size, namely $|L_i| \approx 2^{(\alpha-1)n}$. Because, by construction, $|\chi(h_0)| = \lfloor d^\perp/2 \rfloor$ and $|\chi(h_1)| = \lceil d^\perp/2 \rceil$, the respective cardinalities become $2^{(\alpha-1)\lfloor d^\perp/2 \rfloor n}$ and $2^{(\alpha-1)\lceil d^\perp/2 \rceil n}$. This is clearly dominated by the latter.

(FIRST) JOIN PHASE. This step constructs L_h by finding collisions between $\tilde{L}_{h_0}$ and $\tilde{L}_{h_1}$ in their second components. As $|\tilde{L}_{h_0}| \cdot |\tilde{L}_{h_1}| \approx 2^{d^\perp(\alpha-1)n}$ and we are interested in collisions on αn bits, the expected cardinality of L_h is $|L_h| \approx 2^{(d^\perp(\alpha-1)-\alpha)n}$ (substituting α and $d^\perp = k+1$ results in $2^{(r-k-1)n}$ for MDS codes). Note that each colliding element can be forwarded directly to the next step, thus eliminating the need to store L_h. Moreover, the collision search can be performed in conjunction with (FIRST) MERGE PHASE by storing $\tilde{L}_{h_0}$ and checking (and processing) collisions on-the-fly when generating $\tilde{L}_{h_1}$. This way the memory requirements are reduced to $|\tilde{L}_{h_0}| \approx 2^{(\alpha-1)\lfloor d^\perp/2 \rfloor n}$.

SECOND MERGE PHASE and SECOND JOIN PHASE (for non-MDS codes). For this scenario we restrict ourselves to the four codes suggested by Knudsen and Preneel. For the (chosen) systematic generator matrices we can always find (by inspection) $h, h' \in \mathscr{C}^\perp$ with the property that h has minimal weight, $\{1, \ldots, k\} \subset \chi(h) \cup \chi(h')$ (so we reach reach full rank and can FINALIZE afterwards) and the number of $i \in \chi(h')$ for which $i \notin \chi(h)$ equals $k - d^\perp + 2$. As a result, the relation defined by h' (and thus the second phase) involves $k - d^\perp + 2$ 'fresh' lists L_i (those for which $i \notin \chi(h)$ and $i \in \chi(h')$) with $|L_i| = 2^{(\alpha-1)n}$, as well as L_h, for which $|L_h| = 2^{(d^\perp(\alpha-1)-\alpha)n}$. Hence, regardless of the way of MERGEing and JOINing there are

$$2^{(d^\perp(\alpha-1)-\alpha)n} \cdot 2^{(k-d^\perp+2)(\alpha-1)n} = 2^{(\alpha+k(\alpha-1)-2)n}$$

elements in total to be checked for collisions. As in the previous step, collisions are searched for on αn bits. This leads to a list $L_{h'}$ of roughly $|L_{h'}| = 2^{(k(\alpha-1)-2)n} = 2^{(r-k-2)n}$ elements at the end of SECOND JOIN PHASE.

To minimize the complexity of the merging phase, we need to find the sets $\chi(h'_0)$ and $\chi(h'_1)$ such that $\chi(h'_0) \cap \chi(h'_1) = \emptyset$ and the full $2^{(\alpha+k(\alpha-1)-2)n}$ elements (involved in the merging) are distributed as evenly as possible *without* violating the constraints imposed by the asymmetric list sizes. The condition $\chi(h) \cap \chi(h'_1) = \emptyset$ implies that L_h is merged into $\tilde{L}_{h'_0}$; assume that j further fresh L_i are merged into $\tilde{L}_{h'_0}$. This automatically means that $|\chi(h'_1)| = (k - d^\perp + 2 - j)$ and furthermore that

$$|\tilde{L}_{h'_1}| = 2^{(k-d^\perp+2-j)(\alpha-1)n} \quad \text{and} \quad |\tilde{L}_{h'_0}| = 2^{j(\alpha-1)n+(d^\perp(\alpha-1)-\alpha)n}.$$

Given a particular j, the merging time is governed by the maximum of $|\tilde{L}_{h'_1}|$ and $|\tilde{L}_{h'_0}|$, whereas the

storage requirement is similarly the minimum of that pair. In order to optimize the overall time-complexity, we take the minimum (of the maximum just mentioned) over all j, denote the value by T_1 and, for the value j used, denote by T_2 the corresponding 'memory'-minimum. Note that $T_1 + T_2 = k(\alpha - 1) + \alpha - 2$. Collision-finding can then be performed in $2^{T_1 n}$ time with a memory requirement of roughly $2^{T_2 n}$.

FINALIZATION. For each element in L_h (respectively in $L_{h'}$ for non-MDS codes) we need to perform a simple check (we assume it costs unit time and constant memory). For MDS codes, after the FIRST JOIN PHASE, we obtain that L_h has size roughly $2^{(r-k-1)n}$. For the non-MDS case, we have shown that $|L_{h'}| = 2^{(r-k-2)n}$ (at least for the four non-MDS codes provided by Knudsen and Preneel).

Summing up the obtained complexities for the various steps concludes the proof. (See our results on the compression functions suggested by Knudsen and Preneel in Tables 6.3 and 6.4).

Remark 6.3.9. For the schemes for which our attack is not optimal, the time-complexity is higher than the query-complexity. Thus it is natural to consider increasing the query-complexity in order to bring the overall complexity down, as is so effectively demonstrated by Wagner [200] for the ordinary generalized birthday problem. However, our situation differs crucially from that ordinary one, as we show below.

Normally the lists are constructed as *outputs* of a compression function (or PuRF). Thus increasing the amount of queries simply leads to more elements (uniformly distributed) in the same set. In other words, the density of available elements increases with the number of queries. In our setting, the lists are constructed as *inputs* of a compression function, depending on whether they hit a specific target. Moreover, for our attack to work, we choose our inputs very carefully and exhaust all inputs of the prescribed format. To increase the number of queries, we need to relax the prescribed format. So although we get more answers, they will be chosen from a *larger* space (which will be felt throughout the algorithm). Thus, the density of elements remains the same (namely 2^{-n}), making it less likely for a Wagner-style attack to go through. (From a more technical perspective, the way Wagner achieves an increase in speed is by selecting those elements that have a certain zero-bit-pattern; because we already control the zero-bit pattern of the elements that end up in the list to some extent, intuitively there is less for us to gain by relaxing this control.)

6.3.5 A Space-Efficient Preimage Attack

In the preimage attacks [2] presented in Section 6.3, our main tool for reducing the time-complexity of the MERGE and JOIN PHASES is a general problem whose special instance is a well-known and studied k-list problem: Given k-lists $L_1, \ldots, L_k$ consisting of independent and uniform samples from $\{0,1\}^n$, for each $i \in 1, \ldots, k$, find $x_i \in L_i$ such that $x_1 \oplus \ldots \oplus x_k = 0$. Our attacks take a direct approach to solving the k-list problem, by merging pairs of lists and looking for collisions in corresponding entries. This approach provides an efficient attack in terms of time-complexity, but it is not always sufficiently *space-efficient* for us [3].

2. We note that the improvements presented in this section are also applicable to the relevant steps of the collision attacks given in Section 6.4; yet they do not affect the overall complexity due to the LOCAL COLLISION DETECTION step.

3. We are certainly not the first to face this issue: A series of papers including [41, 47, 179, 200] provide alternatives with different time/memory trade-offs. See Wagner [200] for a survey of cryptographic applications of the k-list problem ranging from symmetric-key to public-key cryptography.

To make things concrete, consider the case of $k = 4$. To find a single solution for the 4-list problem (with high probability), it is straightforward to see that the cardinalities of the lists should be around $2^{n/4}$. This also serves as the minimum memory-complexity [47, 179] among all the known solutions to the 4-list problem. Whereas, the known algorithms requiring $2^{n/4}$ memory are able to achieve only about $2^{n/2}$ time-complexity. Wagner's work [200] presents an excellent compromise, achieving time-complexity $2^{n/3}$ and also memory-complexity $2^{n/3}$. In terms of time-complexity to find a *single* solution to the 4-list problem, this is the best known result.

From One Solution to All Solutions Rather than looking for a single solution, our attacks are interested in finding *all* solutions of a k-list problem (or a related problem, depending on the relations imposed by the particular code $\mathscr{C}$). Indeed, recall the attacks from Section 6.3: We generate all solutions by splitting the codeword in two and looking for all collisions in the corresponding entries in the respective lists.

One simple idea for finding all solutions might involve iterating Wagner's algorithm. But for us, this approach is lacking. To illustrate, let us take a look at the relevant steps (i.e., the MERGE and JOIN PHASES) in our warm-up example from Section 6.3.1. If we apply Wagner's algorithm iteratively 2^n times (to find the same number of solutions) each with $|L_i| = 2^{n/3}$, we need to generate $2^{4n/3}$ partial preimages in total in the QUERY PHASE. This obviously increases the time-complexity of this step, hence the time-complexity of overall attack. Instead, our approach employs a method (primarily due to Chose et al. [47]) that can be seen as a special instance of Wagner's tree algorithm (in particular for $k = 4$), and we aim to hold the time-complexity in place while gaining efficiency in memory-complexity.

Consider again the $\mathsf{KP}^1([5,3,3]_4)$ compression function. We now describe our more space-efficient attack: The differences between this attack and the one given in Section 6.3.1 are in the MERGE and JOIN phases; the remaining steps of the algorithm work exactly as before.

Claim 6.3.10. *For the compression function* $\mathsf{h} = \mathsf{KP}^1([5,3,3]_4)$*, preimages in* h_n *can be found in* $\mathcal{O}(2^{5n/3})$ *time with a memory requirement of* $\mathcal{O}(2^{2n/3})$ n*-bit blocks.*

Proof. Refer to Figure 6.2 for an illustration of the attack. To avoid repetition, we refer to Section 6.3.1 for the details of the QUERY PHASE and FINALIZATION phases. We give the modifications to the MERGE and JOIN phases, for which we need to find all solutions of the 4-list problem over the lists $L_1, \ldots, L_4$. The list of tuples containing all the solutions of the 4-list problem is defined by

$$L_{\{1,2,3,4\}} = \{(x_1, x_2, x_3, x_4) \in L_1 \times L_2 \times L_3 \times L_4 \mid x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0\} \ .$$

We follow a step-by-step approach, using a Wagner-style tree-like algorithm given the lists L_i with $|L_i| \approx 2^{2n/3}$. Let l be a positive integer less than $2n$; its precise choice will be specified later. We denote by $(x_i)_l$ the concatenation of the least significant $l/2$ bits of the n-bit strings x_i^1 and x_i^2 (corresponding to the non-zero blocks in our queries). We first construct the lists (by looking for a correspondence on l bits) $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ defined by (for some l-bit constant $\tilde{c}$)

$$\tilde{L}_{\{1,2\}} = \{(x_1, x_2) \mid (x_1)_l \oplus (x_2)_l = \tilde{c} \text{ and } (x_1, x_2) \in L_1 \times L_2\} \ ,$$

$$\tilde{L}_{\{3,4\}} = \{(x_3, x_4) \mid (x_3)_l \oplus (x_4)_l = \tilde{c} \text{ and } (x_3, x_4) \in L_3 \times L_4\} \ .$$

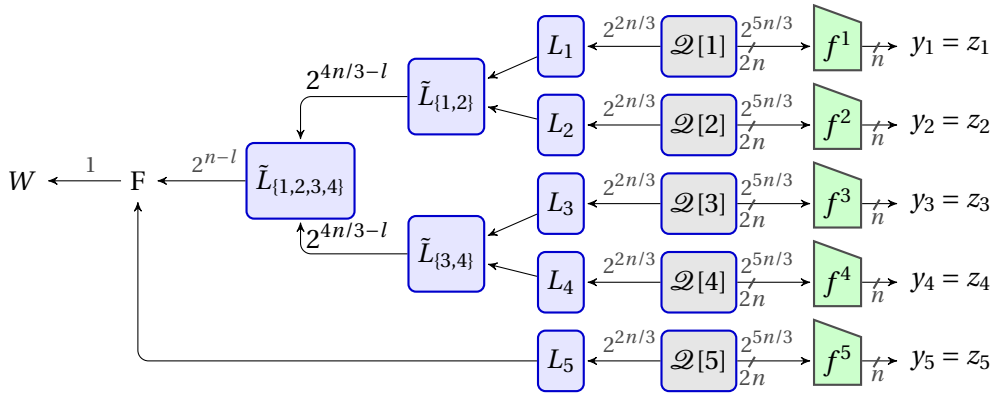

Figure 6.2 – Our space-efficient preimage attack on $\mathsf{h}_n = \mathsf{KP}^1([5,3,3]_{2^2})$ illustrated. The attack works for $l = 2^{2n/3}$.

Note that constructing each list takes $\mathcal{O}(n2^{2n/3})$ time (for sorting and scanning e.g., L_1 and L_3) and $\mathcal{O}(2^{2n/3})$ memory (for storing $L_1, \ldots, L_4$). We are interested in the cardinalities of the lists $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ as they are the main factors affecting the time- and memory-complexity of this phase. It is expected that both lists have roughly $2^{4n/3-l}$ elements, as we can generate $2^{4n/3}$ tuples in total (each from $L_1 - L_2$ and $L_3 - L_4$) and the probability of hitting one of the $\tilde{c} \in \{0,1\}^l$ is 2^{-l}.

Next we look for the correspondence on the remaining $5n/3 - l$ bits of the surviving elements in $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ to obtain all the solutions. As there are $2^{8n/3-2l}$ tuples in total ($2^{4n/3-l}$ from each of $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$) and we are interested in correspondence on $5n/3 - l$ bits, we expect to produce

$$2^{8n/3-2l} \cdot 2^{l-5n/3} = 2^{n-l}$$

solutions, i.e., $|L_{\{1,2,3,4\}}| = 2^{n-l}$. Constructing $L_{\{1,2,3,4\}}$ then takes (on average) $\mathcal{O}(n2^{4n/3-l})$ time and $\mathcal{O}(2^{4n/3-l})$ memory. So, choosing $l = 2n/3$ gives us the optimal memory-complexity for this particular case as the QUERY PHASE requires $\mathcal{O}(2^{2n/3})$ memory. Consequently, we can produce $2^{n/3}$ solutions of the 4-list problem in $\mathcal{O}(2^{2n/3})$ time using $\mathcal{O}(2^{2n/3})$ n-bit blocks of memory with a single $\tilde{c} \in \{0,1\}^{2n/3}$.

The idea for reducing the memory requirements is to iterate this phase for all $\tilde{c} \in \{0,1\}^{2n/3}$ to obtain 2^n solutions at the end (which is essentially what we require for the FINALIZATION phase). Note that the trick is to remove the internal lists $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ at the end of each iteration to keep the memory requirements small. All in all, MERGE and JOIN phases take $\mathcal{O}(n2^{4n/3})$ time ($2^{2n/3}$ iterations each requiring $\mathcal{O}(n2^{2n/3})$ time) and $\mathcal{O}(2^{2n/3})$ memory. Hence, the time-complexity of the overall attack is still dominated by the QUERY PHASE (asymptotically in n) and the memory requirement is $\mathcal{O}(2^{2n/3})$. So, the claim follows. □

Generalization to $k > 4$ The main idea of generalizing the algorithm for $k > 4$ is to reduce the k-list problem to solving a respective 4-list problem. In brief, the algorithm works as follows. Firstly, the k lists are divided into four groups of lists each containing $l_0, \ldots, l_3$ smaller lists. More specifically, we first determine the integers $l_0, \ldots, l_3$ with $l_0 \geq l_1$, $l_2 \geq l_3$ such that $l_0 + \ldots + l_3 = k$. Then, a merging phase is performed (similar to the one presented in our preimage attacks) in each of the four groups to generate four large lists (each containing l_i smaller lists). Finally, the algorithm to find all the solutions of the 4-list problem is mounted over the four large newly generated lists. In turn, this

Algorithm 6.3.11 (A Space-Efficient Algorithm for (FIRST) MERGE-JOIN PHASE)**.**

Input: *A dual codeword h with Hamming weight $d^\perp \geq 4$, partial preimage lists L_i, each of cardinality $N = 2^{(r-k)n/k}$, whose elements are in $\{0,1\}^{cn}$ and obtained from the* QUERY PHASE *as specified in Algorithm 6.3.4.*

Output: $L_h = \left\{X \in \prod_{i\in\chi(h)} L_i \mid (\bar{\varphi}(h) \otimes I_{n'}) \cdot (X+0) = 0\right\}$.

1. MERGE PHASE. *Let $h = h_0 + \ldots + h_3$ with $\chi(h_0) \cap \ldots \cap \chi(h_3) = \emptyset$ and of Hamming weights $l_0, \ldots, l_3$, respectively, with $l_0 \geq l_1$, $l_2 \geq l_3$ such that $l_0 + \ldots + l_3 = d^\perp$. Create for $j = 0, \ldots, 3$,*

$$\tilde{L}_{h_j} = \left\{ \left(X_j, (\bar{\varphi}(h_j) \otimes I_{n'}) \cdot (X_j + 0)\right) \mid X_j \in \prod_{i\in\chi(h_j)} L_i \right\}$$

   all sorted on their second component.

2. JOIN PHASE. *For all $\tilde{c} \in \{0,1\}^l$ (for an $l \leq rn/k$ to be determined, see the proof of Theorem 6.3.12),*

   (a) *Construct $\tilde{L}_{h_{0,1}}$ and $\tilde{L}_{h_{2,3}}$ (store and sort $\tilde{L}_{h_1}$ and $\tilde{L}_{h_3}$ then scan all the elements in $\tilde{L}_{h_0}$ and $\tilde{L}_{h_2}$) defined by*

$$\begin{aligned} \tilde{L}_{h_{0,1}} &= \{((X_0,Y_0),(X_1,Y_1)) \in \tilde{L}_{h_0} \times \tilde{L}_{h_1} \mid (Y_0)_l \oplus (Y_1)_l = \tilde{c}\}, \\ \tilde{L}_{h_{2,3}} &= \{((X_2,Y_2),(X_3,Y_3)) \in \tilde{L}_{h_2} \times \tilde{L}_{h_3} \mid (Y_2)_l \oplus (Y_3)_l = \tilde{c}\}. \end{aligned}$$

   (b) *Obtain (by checking the correspondence on the remaining $rn/k - l$ bits of the related entries of $\tilde{L}_{h_{0,1}}$ and $\tilde{L}_{h_{2,3}}$)*

$$L_h = \left\{ X \in \prod_{i\in\chi(h)} L_i \mid (\bar{\varphi}(h) \otimes I_{n'}) \cdot (X+0) = 0 \right\}.$$

   (See also the JOIN PHASE*(b) in the proof of Theorem 6.3.12 how to perform this step in practice.)*

3. FINALIZATION. *Output L_h.*

gives all the solutions to the corresponding k-list problem. Algorithm 6.3.11 provides the detailed explanation of our novel algorithm for the (FIRST) MERGE-JOIN PHASEs. The analysis is given in Theorem 6.3.12.

Theorem 6.3.12. *Let $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ be given and let $d^\perp \geq 4$ be the minimum distance of the dual code of $\mathscr{C}$. Consider the algorithm described in Algorithm 6.3.11 running as a subroutine for finding preimages for h_n. Then, the expected cardinality of L_h is as in Theorem 6.3.5, the time-complexity of Algorithm 6.3.11 is $\mathscr{O}(N^{\max(l_0+l_1, l_2+l_3)} \log N)$ for $N \approx 2^{(r-k)n/k}$ and $n \to \infty$, and its memory-complexity is $\mathscr{O}(N^{\max(l_1,l_3)})$ (expressed in the number of cn-bit blocks where $e = bc$). In particular (without loss of generality),*

$$l_0 = \lceil d^\perp/4 \rceil\,, \quad l_1 = \lceil d^\perp/2 \rceil - \lceil d^\perp/4 \rceil\,, \quad l_2 = \lfloor d^\perp/2 \rfloor - \lfloor d^\perp/4 \rfloor\,, \quad l_3 = \lfloor d^\perp/4 \rfloor$$

provides an optimal solution (over the partial preimage lists of equal cardinality).

Proof. We first determine the computational complexity of each step separately (neglecting the steps JOIN PHASE(a) and FINALIZATION). Then, summing up each step's time and memory complexities

gives the overall computational requirements of Algorithm 6.3.11. In the following, we assume that $N \approx 2^{(r-k)n/k}$ and the asymptotic analysis is performed for $n \to \infty$. We begin with the MERGE PHASE.

MERGE PHASE. This phase is performed independently over four groups each containing $l_0, \ldots, l_3$ smaller lists of roughly N elements. So, due to the constraint that $l_0 \geq l_1$ and $l_2 \geq l_3$, the expected time-complexity of this step is $\mathcal{O}(N^{\max(l_0,l_2)} \log N)$. Note that memory requirement of this step can be optimized by running MERGE and JOIN PHASE(a) in conjunction (see the analysis of the next phase). So it is enough to store only the merged lists $\tilde{L}_{h_1}$ and $\tilde{L}_{h_3}$; hence the memory requirement is $\mathcal{O}(N^{\max(l_1,l_3)})$ cn-bit blocks.

JOIN PHASE(a). This step constructs $\tilde{L}_{h_{0,1}}$ (and $\tilde{L}_{h_{2,3}}$) by finding collisions between $\tilde{L}_{h_0}$ and $\tilde{L}_{h_1}$ (respectively $\tilde{L}_{h_2}$ and $\tilde{L}_{h_3}$) in their second components. As $|\tilde{L}_{h_0}| \cdot |\tilde{L}_{h_1}| \approx N^{l_0+l_1}$ and $|\tilde{L}_{h_0}| \cdot |\tilde{L}_{h_1}| \approx N^{l_2+l_3}$ and we are interested in collisions on l bits (which will be determined soon), we expect to have $|\tilde{L}_{h_{0,1}}| \approx N^{l_0+l_1} \cdot 2^{-l}$ and $|\tilde{L}_{h_{2,3}}| \approx N^{l_2+l_3} \cdot 2^{-l}$. We can find $\tilde{L}_{h_{i,j}}$ (for $i < j$) by only storing $\tilde{L}_{h_j}$ and scanning through all elements in $\tilde{L}_{h_i}$. Therefore this step requires $\mathcal{O}(N^{\max(l_0,l_2)} \log N)$ time with a memory-complexity of $\mathcal{O}(N^{\max(l_1,l_3)})$ cn-bit blocks. Note that MERGE PHASE can be run as a sub-step of this phase; we can first generate and store the merged lists $\tilde{L}_{h_1}$ and $\tilde{L}_{h_3}$. Then, we can find the collisions by simply generating the lists $\tilde{L}_{h_0}$, $\tilde{L}_{h_2}$ and checking (on-the-fly) the required l-bit relations.

JOIN PHASE(b). In this step, we are interested in the collisions on the remaining $rn/k - l$ non-zero bits of the related entries of $\tilde{L}_{h_{0,1}}$ and $\tilde{L}_{h_{2,3}}$. In order to minimize the memory requirements, we store only the smaller of $\tilde{L}_{h_{0,1}}$ and $\tilde{L}_{h_{2,3}}$. Therefore, this step can be performed in $\mathcal{O}(N^{\max(l_0+l_1,l_2+l_3)} 2^{-l} \log N)$ time with a memory-complexity of $\mathcal{O}(N^{\min(l_0+l_1,l_2+l_3)} 2^{-l})$ cn-bit blocks. As

$$|\tilde{L}_{h_{0,1}}| \cdot |\tilde{L}_{h_{2,3}}| \approx N^{d^\perp} \cdot 2^{-2l}$$

(because $l_0 + \ldots + l_3 = d^\perp$) and we are looking for collisions on the remaining $rn/k - l$ bits, the expected cardinality of L_h is

$$|L_h| \approx N^{d^\perp} \cdot 2^{-2l} \cdot 2^{l - rn/k}.$$

Substituting $N = 2^{(r-k)n/k}$, we get

$$|L_h| \approx 2^{(d^\perp(r-k)-r)n/k - l}.$$

Because we iterate this step 2^l times, we eventually have

$$|L_h| \approx 2^{(d^\perp(r-k)-r)n/k}.$$

This is exactly what we obtained in Theorem 6.3.5. Note that regardless of the choice of l, this step requires $\mathcal{O}(N^{\max(l_0+l_1,l_2+l_3)} \log N)$ time. Whereas, for the memory-complexity, we can choose the optimal value of l to find the minimum required storage. That is, because we already need $\mathcal{O}(N^{\max(l_1,l_3)})$ cn-bit blocks of memory, choosing l such that

$$\min(N^{l_0+l_1}, N^{l_2+l_3}) \cdot 2^{-l} = N^{\max(l_1,l_3)}$$

does not increase the memory requirements. Hence, JOIN PHASE can be performed with $\mathcal{O}(N^{\max(l_1,l_3)})$ cn-bit of memory.

Note that a particular even division among $d^\perp$ lists can be done as follows. Firstly, we group the $d^\perp$ lists into two; one with $\lceil d^\perp/2 \rceil$ smaller lists and the other with $\lfloor d^\perp/2 \rfloor$. This is an even 2-division.

Code	$d^\perp$	Compression Function	Our Attack Complexity Query $2^{rn/k}$	Time Theorems 6.3.5, 6.3.7	Memory Theorems 6.3.12, 6.3.7
$[5,3,3]_4$	4	$(5+1)n \to 5n$	$2^{5n/3}$	$2^{5n/3}$	$2^{2n/3}$
$[8,5,3]_4$	4	$(8+2)n \to 8n$	$2^{8n/5}$	$2^{8n/5}$	$2^{3n/5}$
$[12,9,3]_4$	7	$(12+6)n \to 12n$	$2^{4n/3}$	$2^{4n/3}$	$2^{2n/3}$
$[9,5,4]_4$	4	$(9+1)n \to 9n$	$2^{9n/5}$	$2^{11n/5}$	$2^{4n/5}$
$[16,12,4]_4$	11	$(16+8)n \to 16n$	$2^{4n/3}$	$2^{7n/3}$	2^n
$[6,4,3]_{16}$	5	$(6+2)n \to 6n$	$2^{3n/2}$	$2^{3n/2}$	$2^{n/2}$
$[8,6,3]_{16}$	7	$(8+4)n \to 8n$	$2^{4n/3}$	$2^{4n/3}$	$2^{2n/3}$
$[12,10,3]_{16}$	11	$(12+8)n \to 12n$	$2^{6n/5}$	$2^{6n/5}$	$2^{3n/5}$
$[9,6,4]_{16}$	7	$(9+3)n \to 9n$	$2^{3n/2}$	2^{2n}	2^n
$[16,13,4]_{16}$	14	$(16+10)n \to 16n$	$2^{16n/13}$	2^{2n}	$2^{9n/13}$
$[4,2,3]_8$	3	$(4+2)n \to 4n$	2^{2n}	2^{2n}	2^n
$[6,4,3]_8$	5	$(6+6)n \to 6n$	$2^{3n/2}$	$2^{3n/2}$	$2^{n/2}$
$[9,7,3]_8$	8	$(9+12)n \to 9n$	$2^{9n/7}$	$2^{9n/7}$	$2^{4n/7}$
$[5,2,4]_8$	3	$(5+1)n \to 5n$	$2^{5n/2}$	2^{3n}	$2^{3n/2}$
$[7,4,4]_8$	5	$(7+5)n \to 7n$	$2^{7n/4}$	$2^{9n/4}$	$2^{3n/4}$
$[10,7,4]_8$	8	$(10+11)n \to 10n$	$2^{10n/7}$	2^{2n}	$2^{6n/7}$

Table 6.5 – Space-efficient results on Knudsen–Preneel Compression Functions based on $[r,k,d]_{2^e}$ codes. Non-MDS parameters in italic.

Then, we proceed similarly with the newly generated smaller groups of lists. Therefore, the choice

$$l_0 = \lceil d^\perp/4 \rceil\,, \quad l_1 = \lceil d^\perp/2 \rceil - \lceil d^\perp/4 \rceil\,, \quad l_2 = \lfloor d^\perp/2 \rfloor - \lfloor d^\perp/4 \rfloor\,, \quad l_3 = \lfloor d^\perp/4 \rfloor$$

provides an optimal 4-division. Hence, the claim follows. □

Remark 6.3.13. Because $d^\perp < 4$ for $\mathsf{KP}^1([4,2,3]_8)$ and $\mathsf{KP}^1([5,2,4]_8)$, we cannot apply Algorithm 6.3.11 for these compression functions, instead we perform the naive method presented in Algorithm 6.3.4. These are the only exceptions among the parameters suggested by Knudsen and Preneel. We could also apply (a variant of) Algorithm 6.3.11 to have a more space-efficient solution for the SECOND MERGE-JOIN PHASE of the non-MDS case. However, due to the asymmetry between the list sizes, we need a similar optimization trick to determine the l_i values. Once this is done, the rest of the algorithm works exactly in the same way. The ramifications of the space-efficient preimage-finding algorithm to all compression functions suggested by Knudsen and Preneel are given in Table 6.5.

6.3.6 Information-Theoretic Security Proof

The following result provides a security proof for preimage resistance of the Knudsen–Preneel compression functions in the information-theoretic model. That is, we give a lower bound on the query-complexity (for a computationally unbounded adversary) of any preimage-finding attack. This bound shows that the query-complexity of our new attack is optimal, up to a small factor. Hence, the time-complexity of our preimage attack becomes *optimal* whenever the time-complexity of our attack matches its query-complexity.

Theorem 6.3.14. *Let* $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *(with systematic generator matrix) and, for* b *dividing* n*, consider* h_n *based on underlying PuRFs* $f^i \in \mathrm{Func}(cn,n)$ *for* $i = 1,\ldots,r$ *with* $c = e/b$*. Then for* $q \le 2^{cn}$ *queries to each of the oracles and* $\delta \ge 0$ *an arbitrary real number:*

$$\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q) \le \binom{r}{k}\frac{kq^{1+\delta}}{2^{(r-k+1)n}} + rp\,,$$

where $p = \Pr\left[B\left[q;2^{-n}\right] > q^{\delta/(k-1)}\right]$ *and* $B\left[q;2^{-n}\right]$ *denotes a random variable counting the number of successes in* q *independent Bernoulli trials, each with success probability* 2^{-n}*.*

Proof. Let $Z = z_1 \| \cdots \| z_r$ be the range point to be inverted. Recall that $f^1,\ldots,f^k$ are the functions corresponding to the systematic part of the $[r,k,d]_{2^e}$ code. Without loss of generality, we restrict our attention to an adversary $\mathscr{A}$ asking exactly q queries to each of its oracles. Consider the transcript of the oracle queries and responses. To complete a preimage, in this transcript there must necessarily exist at least one tuple $(x_1,\ldots,x_k)$ of queries to $f^1,\ldots,f^k$ such that for all $i = 1,\ldots,k$ we have $f^i(x_i) = z_i$. Notice that because $f^1,\ldots,f^k$ correspond to the systematic portion of the code, any tuple $(x_1,\ldots,x_k)$ of queries to these k PuRFs *uniquely* defines a tuple of queries $(x_{k+1},\ldots,x_r)$ to the remaining $r-k$ PuRFs. Thus the number of tuples $(x_1,\ldots,x_k)$ in the transcript such that $f^i(x_i) = z_i$ for all $i = 1,\ldots,k$ determines the number of tuples $(x_{k+1},\ldots,x_r)$ that could possibly be a (simultaneous) preimage for $z_{k+1} \| \cdots \| z_r$. Intuitively, if this number is bounded to be sufficiently small, then $\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(\mathscr{A})$ will also be small. Let us make this formal.

For all $i = 1,\ldots,k$, let L_i denote the list of partial preimages (at the end of the game), that is the set of all x_i that $\mathscr{A}$ queried to f^i for which $f^i(x_i) = z_i$ and let $N_i = |L_i|$. Then N_i is the random variable denoting the number of partial preimages $\mathscr{A}$ found. Let $\mathsf{bad}(\mathscr{Q})$ be the event that there exists an $i \in \{1,\ldots,r\}$ such that $N_i \ge q^{\delta/(k-1)}$ where $\delta \ge 0$ is an arbitrary real number. As all the N_i are independent binomial random variables with success probability 2^{-n} and q trials each, we have that

$$\Pr[\mathsf{bad}(\mathscr{Q})] \le r\Pr\left[B\left[q;2^{-n}\right] > q^{\delta/(k-1)}\right] = rp\,.$$

Let $\mathscr{Q}_j$ be the transcript of the oracle queries and responses after the j'th query. Then, we have (Proposition 3.3.7)

$$\Pr\left[\mathsf{epre}_Z(\mathscr{Q})\right] \le rp + \binom{r}{k}\sum_{j=1}^{kq}\Pr\left[\mathsf{epre}_Z(\mathscr{Q}_j) \mid \neg\mathsf{epre}_Z(\mathscr{Q}_{j-1}) \wedge \neg\mathsf{bad}(\mathscr{Q}_{j-1})\right]\,,$$

where the term $\binom{r}{k}$ is the maximum number of possible choices for the k linearly independent PuRF inputs. Given that $\neg\mathsf{bad}(\mathscr{Q}_{j-1})$ holds, we know that $N_i < q^{\delta/(k-1)}$ and hence $\prod_{i\in\mathscr{I}} N_i < q^{\delta}$ for $\mathscr{I} \subset \{1,\ldots,k\}$ with $|\mathscr{I}| = k-1$. Thus, no matter what fresh query is asked to $f^1,\ldots,f^k$ (and regardless of order), we know that there will be at most q^{δ} query tuples generated by the fresh query that can be preimages of $z_1 \| \cdots \| z_k$. By our argument above, then, there will be at most this many tuples $(x_{k+1},\ldots,x_r)$ that can contribute to the occurrence of the event $\mathsf{epre}_Z(\mathscr{Q}_j)$.

Consider the j'th query; without loss of generality, assume that it is asked to f^1. As argued above, there exist at most q^{δ} query tuples generated by this fresh query that can be preimages of $z_1 \| \cdots \| z_k$. Now let us study the following experiment: Adversary $\mathscr{A}'$ is allowed to make at most q^{δ} queries to an

oracle that, on input an $(r-k+1)$-tuple $(x_1, x_{k+1}, \ldots, x_r)$, returns

$$\left(f^1(x_1), f^{k+1}(x_{k+1}), \ldots, f^r(x_r)\right) .$$

$\mathscr{A}'$ wins if the oracle ever returns $(z_1, z_{k+1}, \ldots, z_r)$. As the PuRFs $f^1, f^{k+1}, \ldots, f^r$ are independent, it is easy to see that the probability that $\mathscr{A}'$ wins is at most $q^\delta / 2^{(r-k+1)n}$. Thus in the epre experiment, the probability that $\mathscr{A}$ manages to produce a tuple $(x_1, x_{k+1}, \ldots, x_r)$ that yields $z_1, z_{k+1}, \ldots, z_r$, given that at most q^δ such tuples can be considered, is at most $q^\delta / 2^{(r-k+1)n}$. So then

$$\Pr\left[\mathsf{epre}_Z(\mathscr{Q}_j) \mid \neg\mathsf{epre}_Z(\mathscr{Q}_{j-1}) \wedge \neg\mathsf{bad}(\mathscr{Q}_{j-1})\right] \leq \frac{q^\delta}{2^{(r-k+1)n}} .$$

We use the union bound to conclude the proof. □

Corollary 6.3.15. *Let* $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *(with systematic generator matrix) and, for b dividing n, consider* h_n *based on underlying PuRFs* $f^i \in \mathrm{Func}(cn, n)$ *for* $i = 1, \ldots, r$ *with* $c = e/b$*. Then asymptotically for n (with b* $n' = n$*) and*

$$q \leq g(n)\left(\frac{2^n}{e}\right)^{r/k}$$

with $g(n) = o(1)$ *it holds that*

$$\mathsf{Adv}^{\mathsf{epre}}_{\mathsf{h}}(q) = o(1) .$$

Proof. Set $\delta = (r-k)(k-1)/r$ and substitute $q = g(n)\,(2^n/e)^{r/k}$ in the statement of Theorem 6.3.14. We will show that asymptotically both terms vanish, starting with the first. After substitution, using

$$1 + \delta = (rk - k(k-1))/r ,$$

we obtain

$$\frac{kq^{1+\delta}}{2^{(r-k+1)n}} = \frac{k\left[g(n)\left(\frac{2^n}{e}\right)^{r/k}\right]^{(rk-k(k-1))/r}}{2^{(r-k+1)n}} = \frac{kg(n)^{(rk-k(k-1))/r}}{e^{r-k+1}} ,$$

which clearly vanishes whenever g does. For the second term in the bound of Theorem 6.3.14, we need to bound the tail probability of a binomial distribution, for which we use Chernoff's bound applied to $\kappa = q^{\delta/(k-1)} = q^{(r-k)/r}$, so

$$p = \Pr\left[B\left[q; 2^{-n}\right] > \kappa\right] < \left(eq/(\kappa 2^n)\right)^\kappa = \left(\frac{e}{2^n} q^{k/r}\right)^\kappa ,$$

where we have left κ in the exponent for brevity. Substituting $q = g(n)\left(\frac{2^n}{e}\right)^{r/k}$ in the rightmost side of the previous equation then leads to

$$p < \left(\frac{e}{2^n}\left(g(n)\left(\frac{2^n}{e}\right)^{r/k}\right)^{k/r}\right)^\kappa = \left(g(n)\right)^{k\kappa/r} ,$$

where $k\kappa/r \geq (k^2)/r$, implying that p vanishes as well for $g(n) = o(1)$ as $n \to \infty$. □

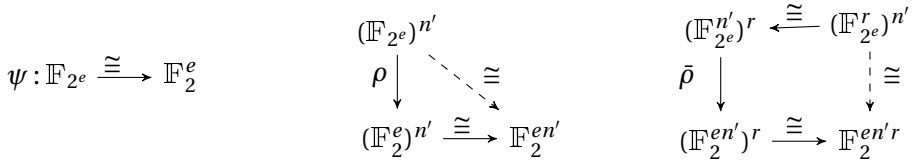

Figure 6.3 – Auxiliary maps used in Section 6.4.1. The rightmost diagram illustrates the isomorphism $\bigoplus_{j=1}^{n'} U_j \to \{0,1\}^{ern'}$ for $U_j = \mathbb{F}_{2^e}^r$.

6.4 Another Look at Collision Resistance

6.4.1 Decoding the Knudsen–Preneel Preprocessing

In this section, we develop the tools that are used in the optimized collision-finding algorithms presented in the subsequent sections. For the preimage resistance, such an analysis was not required simply because of the simplicity of the algorithms; for collision resistance, we definitely need to give more mathematical details (we use the same notation as in the previous sections in this chapter; if a new notion is needed, we introduce related notation and definitions here).

Recall that (see Section 3.1) we are given an injective ring homomorphism $\varphi : \mathbb{F}_{2^e} \to \mathbb{F}_2^{e\times e}$ and a group isomorphism $\psi : \mathbb{F}_{2^e} \to \mathbb{F}_2^e$ that satisfy $\varphi(g)\psi(h) = \psi(gh)$ for all $g, h \in \mathbb{F}_{2^e}$. In the following, as usual, we let $[r,k,d]_{2^e}$ be a linear code with generator matrix $G \in \mathbb{F}_{2^e}^{k\times r}$, let b be a positive divisor of e such that $ek > rb$ and finally let $n = bn'$. Then the preprocessing $C^{\mathsf{pre}} : \{0,1\}^{ekn'} \to \{0,1\}^{ern'}$ of the Knudsen–Preneel compression function is defined by $C^{\mathsf{pre}}(W) = (\bar{\varphi}(G^T) \otimes I_{n'}) \cdot W$ (and note that $ern' = rcn$). Throughout this chapter, we denote $\Im(C^{\mathsf{pre}})$ to be the image of C^{pre}.

Characterization of $\Im(C^{\mathsf{pre}})$ as a Sum We have previously written the co-domain of C^{pre} as a direct sum of *PuRF inputs* by identifying $\{0,1\}^{ren'}$ with $\bigoplus_{i=1}^r V_i$ for $V_i = \mathbb{F}_2^{en'}$. Here we use a second interpretation that emphasizes the *code*. We consider $\bigoplus_{j=1}^{n'} U_j$ for $U_j = \mathbb{F}_{2^e}^r$. As $\mathbb{F}_{2^e}^r$, and by extension $\bigoplus_{j=1}^{n'} U_j$, is a vector space over $\mathbb{F}_{2^e}$, whereas $\{0,1\}^{ern'}$ is a stand-in for the vector space $\mathbb{F}_2^{ern'}$ over $\mathbb{F}_2$, we cannot find a vector space isomorphism. Nonetheless, we can find a suitable group isomorphism

$$\bigoplus_{j=1}^{n'} U_j \to \{0,1\}^{ern'}.$$

To define the group isomorphism we exploit that, luckily, the underlying $\mathbb{F}_{2^e}$ arithmetic is essentially preserved by $C^{\mathsf{pre}} : \{0,1\}^{ekn'} \to \{0,1\}^{ern'}$, even though the '$\otimes I_{n'}$' in $C^{\mathsf{pre}}(W) = (\bar{\varphi}(G^T) \otimes I_{n'}) \cdot W$ garbles things up. To formalize this, let $\rho : \mathbb{F}_{2^e}^{n'} \to \mathbb{F}_2^{en'}$ be the group isomorphism such that

$$\rho(g\delta) = (\varphi(g) \otimes I_{n'}) \cdot \rho(\delta)$$

for all $\delta \in \mathbb{F}_{2^e}^{n'}$ and $g \in \mathbb{F}_{2^e}$. As usual, we extend ρ to e.g., r-tuples of elements in $\mathbb{F}_{2^e}^{n'}$ (and hence to vectors in $\left(\mathbb{F}_{2^e}^{n'}\right)^r$) by component-wise application, i.e., $\bar{\rho} : \left(\mathbb{F}_{2^e}^{n'}\right)^r \to \mathbb{F}_2^{en'r}$. This suffices for a group isomorphism

$$\bigoplus_{j=1}^{n'} U_j \to \{0,1\}^{ern'}$$

as well [4] (see Figure 6.3). The following lemma makes the above discussion more concrete in the context of Knudsen–Preneel preprocessing.

Lemma 6.4.1. *Let $I_0 \subset \{1,\dots,r\}$, $\mathscr{C}'$ be the (quasi) shortening of $\mathscr{C}$ on I_0 (see Section 3.1 for the definition) and let $\mathscr{C}'_j = \mathscr{C}' \subseteq U_j = \mathbb{F}_{2^e}^r$ for $j = 1,\dots,n'$. Then*

$$X = (x_1,\dots,x_r) \in \Im(C^{\mathsf{pre}})$$

with $x_i = 0$ for all $i \in I_0$ if and only if for all $j = 1,\dots,n'$ there exists a unique

$$g_j = (g_{j1},\dots,g_{jr}) \in \mathscr{C}'_j$$

such that

$$x_i = \rho\left(g_{1i},\dots,g_{n'i}\right)$$

for all $i = 1,\dots,r$.

Proof. Let $X \in \Im(C^{\mathsf{pre}})$. Then, by construction of a Knudsen–Preneel compression function, there is a unique $W \in \mathbb{F}_2^{ken'}$ for which $X = (\bar{\varphi}(G^T) \otimes I_{n'}) \cdot W$. Because the generator matrix (of the corresponding code $\mathscr{C}$) G is contained in $\mathbb{F}_{2^e}^{k\times r}$, we can write, as usual, $X = (\bar{\varphi}(G^T) \otimes I_{n'}) \cdot W$ as for all $i \in \{1,\dots,r\}$ and $j' \in \{1,\dots,k\}$

$$x_i = \sum_{j'=1}^{k} \left(\varphi(G_{j'i}) \otimes I_{n'}\right) \cdot w_{j'} \quad \text{for} \quad W = (w_1,\dots,w_k) \quad \text{and} \quad w_{j'} \in \mathbb{F}_2^{en'} .$$

Here $G_{j'i} \in \mathbb{F}_{2^e}$ denotes the entry in the (j')'th row and the i'th column of G. By applying ρ^{-1} on both sides (and after some algebraic manipulation together with using $\rho(g\delta) = (\varphi(g) \otimes I_{n'}) \cdot \rho(\delta)$), this leads, for all $i \in \{1,\dots,r\}$, to

$$\rho^{-1}(x_i) = \rho^{-1}\left(\sum_{j'=1}^{k} \left(\varphi(G_{j'i}) \otimes I_{n'}\right) \cdot w_{j'}\right) = \sum_{j'=1}^{k} \rho^{-1}\left(\left(\varphi(G_{j'i}) \otimes I_{n'}\right) \cdot w_{j'}\right) = \sum_{j'=1}^{k} G_{j'i} \cdot \rho^{-1}(w_{j'}) .$$

Observing that both $\rho^{-1}(x_i)$ and $\rho^{-1}(w_{j'})$ are vectors in $\mathbb{F}_{2^e}^{n'}$ yet $G_{j'i}$ is simply a scalar, the above expression for $\rho^{-1}(x_i)$ is equivalent to (for all $j \in \{1,\dots,n'\}$ and $i \in \{1,\dots,r\}$)

$$g_{ji} := \left(\rho^{-1}(x_i)\right)_j = \sum_{j'=1}^{k} G_{j'i} \cdot \left(\rho^{-1}(w_{j'})\right)_j ,$$

where we write for $\left(\rho^{-1}(x_i)\right)_j \in \mathbb{F}_{2^e}$ and $j \in \{1,\dots,n'\}$

$$\rho^{-1}(x_i) = \left(\left(\rho^{-1}(x_i)\right)_1,\dots,\left(\rho^{-1}(x_i)\right)_{n'}\right) .$$

Similarly, for $\left(\rho^{-1}(w_{j'})\right)_j \in \mathbb{F}_{2^e}$, we have

4. Note that if Knudsen and Preneel had defined their compression functions slightly differently (we believe the choice was not made on purpose) namely by $C^{\mathsf{pre}}(W) = \bar{\varphi}(G^T \otimes I_{n'}) \cdot W$, then we simply could have used $\rho = \bar{\varphi}$ without any need for ungarbling; for collision and preimage resistance of the compression function this change is irrelevant in the PuRF model, as the garbling can effectively be absorbed by the PuRFs.

$$\rho^{-1}(w_{j'}) = \left(\left(\rho^{-1}(w_{j'})\right)_1, \ldots, \left(\rho^{-1}(w_{j'})\right)_{n'}\right) .$$

Now, for $j = 1, \ldots, n'$, we have

$$g_j := (g_{j1}, \ldots, g_{jr}) = \left(\sum_{j'=1}^{k} G_{j'1} \cdot \left(\rho^{-1}(w_{j'})\right)_j, \ldots, \sum_{j'=1}^{k} G_{j'r} \cdot \left(\rho^{-1}(w_{j'})\right)_j\right) \in U_j = \mathbb{F}_{2^e}^r .$$

Note that $g_j \in \mathscr{C}$; this follows from the fact that g_j is a codeword obtained from codewords appearing in the rows of the generator matrix. Indeed, it is obtained by multiplying the codewords in each row by the constant $\left(\rho^{-1}(w_{j'})\right)_j$ and summing them up, which clearly gives a valid codeword in $\mathscr{C}$. Moreover, we have

$$x_i = \rho\left(g_{1i}, \ldots, g_{n'i}\right) ,$$

which is easy to verify from the definition of g_{ji}. Finally, for all $i \in I_0$ we are given that $x_i = 0$, which implies $\rho^{-1}(x_i) = 0$ and therefore $g_{ji} = 0$ for all $j = 1, \ldots, n'$. Thus, indeed $g_j \in \mathscr{C}'$ for all $j = 1, \ldots, n'$. Running this argument in reverse completes the proof. □

Pure Tensors in $\Im(C^{\mathsf{pre}})$ As $\mathbb{F}_{2^e}^{n'r}$ is isomorphic (as vector space over $\mathbb{F}_{2^e}$) to the tensor product $\mathbb{F}_{2^e}^r \otimes \mathbb{F}_{2^e}^{n'}$ this leads in a natural way to a function from $\mathbb{F}_{2^e}^r \times \mathbb{F}_{2^e}^{n'}$ to $\{0,1\}^{ren'}$ by considering pure tensors $g \otimes \delta$ with $g \in \mathbb{F}_{2^e}^r$ and $\delta \in \mathbb{F}_{2^e}^{n'}$. Note that we do not discriminate between different representatives, that is for some non-zero $\beta \in \mathbb{F}_{2^e}$ we have that $g \otimes \delta = (\beta g) \otimes (\beta^{-1}\delta)$. The following lemma provides an alternative formulation to Lemma 6.4.1.

Lemma 6.4.2. *If $g \in \mathbb{F}_{2^e}^r$ and $\delta \in \mathbb{F}_{2^e}^{n'}$ then*

$$\bar{\rho}(g \otimes \delta) \in \Im(C^{\mathsf{pre}}) \;\Leftrightarrow\; g \in \mathscr{C} \quad \text{or} \quad \delta = 0 .$$

Proof. Let $g \in \mathbb{F}_{2^e}^r$ such that $g = (g_1, \ldots, g_r)$ and $\delta \in \mathbb{F}_{2^e}^{n'}$ with $\delta = (\delta_1, \ldots, \delta_{n'})$. Consider n' copies of g in the various subspaces that add up to $\mathbb{F}_{2^e}^{n'r}$. Then

$$g \otimes \delta = \left(\left(g_1\delta_1, \ldots, g_1\delta_{n'}\right), \ldots, \left(g_r\delta_1, \ldots, g_r\delta_{n'}\right)\right) ,$$

which implies

$$\bar{\rho}(g \otimes \delta) = \left(\rho\left(g_1\delta_1, \ldots, g_1\delta_{n'}\right), \ldots, \rho\left(g_r\delta_1, \ldots, g_r\delta_{n'}\right)\right) .$$

Writing $g_{ji} = g_i\delta_j$, for $i = 1, \ldots, r$ and $j = 1, \ldots, n'$, together with Lemma 6.4.1 with empty I_0 then implies that $\bar{\rho}(g \otimes \delta) \in \Im(C^{\mathsf{pre}})$ if and only if $g'_j := \left(g_{j1}, \ldots, g_{jr}\right) \in \mathscr{C}$ for all $j = 1, \ldots, n'$. The latter holds if and only if $g \in \mathscr{C}$ or $\delta = 0$. □

Completion Property For a code and any index set $I \subseteq \{1, \ldots, r\}$ (see Section 3.1), we want to define $\tilde{I} \subset \{1, \ldots, r\}$ such that $G_{\tilde{I}}$ is invertible (thus in particular $|\tilde{I}| = k$) and $\tilde{I} \subseteq I$ or $I \subseteq \tilde{I}$. The following lemma states that invertibility of $G_{\tilde{I}}$ suffices to invert C^{pre}.

Lemma 6.4.3. *Let G be a generator matrix for an $[r, k, d]_{2^e}$ code. Let $\tilde{I} \subset \{1, \ldots, r\}$ be such that $G_{\tilde{I}}$ is invertible, with transposed inverse $G_{\tilde{I}}^{-T}$. Let n' be an integer and, for $i = 1, \ldots, r$, let $V_i = \mathbb{F}_2^{en'}$. If given*

$x_i \in V_i$ for $i \in \tilde{I}$, or equivalently $\tilde{X} = (x_{i_1}, \ldots, x_{i_{|\tilde{I}|}})$, for $\{i_1, \ldots, i_{|\tilde{I}|}\} = \tilde{I}$ then

$$W = \left(\bar{\varphi}(G_{\tilde{I}}^{-T}) \otimes I_{n'}\right) \cdot \tilde{X}$$

is the unique element for which $X' = C^{\mathsf{pre}}(W)$ satisfies $x'_i = x_i$ for $i \in \tilde{I}$.

Proof. Simply note that $\left(\bar{\varphi}\left(G_{\tilde{I}}^{T}\right) \otimes I_{n'}\right) \cdot \left(\bar{\varphi}\left(G_{\tilde{I}}^{-T}\right) \otimes I_{n'}\right) = I_{ern'}$. □

6.4.2 Watanabe's Collision-Finding Attack Revisited

Knudsen and Preneel leave a considerable gap between the actual complexity of their attacks and claimed lower bounds in the case of collision resistance. Watanabe [205] points out a collision attack (Algorithm 6.4.4) that runs in (asymptotically in n) time $\mathcal{O}(2^n)$. Thus, for many of the parameter sets, it is better than the one given by Knudsen and Preneel. More interestingly, his attack serves as proof that the time-complexity *lower* bound (Proposition 6.1.4) given by Knudsen and Preneel is *incorrect* for a large class of parameters: whenever $r - k < k$ (to ensure that the attack works) and $d > 3$ (so $n < (d-1)n/2$). This affects 6 out of 16 parameter sets.

The idea of the attack is to find non-trivial collisions for the PuRFs corresponding to the systematic portion of the generator matrix while imposing trivial collisions for the rest. Assume that the code's generator matrix is systematic, that is $G = (I_k|P)$ with $P \in \mathbb{F}_{2^e}^{k\times(r-k)}$. Then the goal is to generate, for each $i \in \{1, \ldots, k\}$, a colliding pair of inputs $x_i \neq x'_i$ (and $f^i(x_i) = f^i(x'_i)$) in such a way that their completion to full 'codewords' satisfies $x_i = x'_i$ for $i \in \{k+1, \ldots, r\}$. This is done by ensuring that $x_i \oplus x'_i = \Delta_i$ where

$$\Delta = (\Delta_1, \ldots, \Delta_k) \in \mathbb{F}_2^{ken'} \setminus \{0\}$$

is in the kernel of $\bar{\varphi}(P^T) \otimes I_{n'}$ (as $r - k < k$ the kernel is guaranteed to contain a non-trivial element). Mutual independence of the inputs to the PuRFs corresponding to the code's systematic part allow the initial collision searches to be mounted independently. Because the collisions need to be rather special (due to fixed Δ_i values), however, the birthday paradox does not apply and a collision search costs about 2^n queries (per PuRF). On the plus side, the attack is trivially memoryless and can be parallelized.

Proposition 6.4.5 (Original Watanabe attack [205])**.** *Let $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ be given with $r - k < k$. Consider h_n (with $b|n$ for b n' $=n$). Then collisions for h_n can be found in time $\mathcal{O}(2^n)$, asymptotically in n, using as many PuRF evaluations.*

Revising Watanabe's Attack

Watanabe's attack has complexity $k2^n$, requires $k > r - k$ and essentially finds a single collision. Below we give a revised and improved version of his algorithm (Algorithm 6.4.6). It has a complexity of only $d2^n$, requires $k \geq d$ and it potentially results in many collisions. More precisely, if $k > d$, then after the initial effort (of $d2^n$) we can find a new collision in *constant* time, for up to an impressive $2^{(k-d)cn}$ number of collisions (recall that $e = bc$).

In his note, Watanabe describes his attack as a differential attack where originally Δ was computed as some non-trivial kernel element. We compute Δ based on a codeword $g \in \mathscr{C}$ of sufficiently low

Algorithm 6.4.4 (Watanabe's Original Collision Attack)**.**

Input: $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *satisfying* $r-k<k$*, a systematic generator matrix* $G=(I_k|P)$*, and a block-size* n *with* $b|n$ *(*$b\ n'=n$*).*

Output: *A colliding pair* $\left(W,W'\right) \in \left(\{0,1\}^{ekn'}\right)^2$ *such that* $\mathsf{h}_n(W)=\mathsf{h}_n(W')$ *and* $W \oplus W' = \Delta \neq 0$ *with* $(\bar{\varphi}(P^T)\otimes I_{n'})\cdot\Delta = 0$.

1. INITIALIZATION. *Compute* $0 \neq \Delta = (\Delta_1,\ldots,\Delta_k)$ *in the kernel of* $\bar{\varphi}(P^T)\otimes I_{n'}$.
2. QUERY PHASE. *For* $i=1,\ldots,k$ *do*
   a. *Generate a random* $x_i \in \mathbb{F}_2^{en'}$;
   b. *Set* $x_i' \leftarrow x_i \oplus \Delta_i$;
   c. *Query* $y_i \leftarrow f^i(x_i)$ *and* $y_i' \leftarrow f^i(x_i')$;
   d. *If* $y_i = y_i'$ *keep* (x_i,x_i') *and proceed to the next* i*, else return to step a.*
3. FINALIZATION. *For* $i=1,\ldots,k$ *set* $w_i \leftarrow x_i$ *and* $w_i' \leftarrow x_i'$*. Output* (W,W').

weight and an arbitrary (non-zero) 'block multiplier' δ. In particular, we set $\Delta = \bar{\rho}(g\otimes\delta)$. By using a minimal weight codeword, the attack performs best.

For the revised attack to work, we need one further ingredient. Watanabe assumes a systematic code and exploits that, when $k<r-k$, there exists a non-zero codeword $g\in\mathscr{C}$ for which $\chi(g)\subseteq\{1,\ldots,k\}$. This allows for an easy completion of a partial collision to a full collision. Our revised version allows for an arbitrary (non-zero) codeword g of weight at most k (existence of which requires $d\leq k$). Thus $\chi(g)$ might no longer map to the systematic part of the code. Luckily, Lemma 6.4.3 provides completion to a full collision, provided $I=\chi(g)$ is *admissible* (see Section 3.1). For MDS codes, all codewords are admissible; for the four non-MDS codes proposed by Knudsen and Preneel, it can be verified that the minimum distance codewords are admissible.

Theorem 6.4.7 (Revised Watanabe attack)**.** *Let* $\mathsf{h}=\mathsf{KP}^b([r,k,d]_{2^e})$ *be given with* $d\leq k$*. Consider* h_n *(with* $b|n$ *and* $b\ n'=n$*). Then Algorithm 6.4.6 using a minimum-weight codeword* g *(and an arbitrary non-zero* δ*) finds collisions for* h_n *in expected time* $d2^n$ *(using as many PuRF evaluations and ignoring a small additive constant).*

Proof. Before proving the correctness of the algorithm, we quickly verify the complexity claim. For this we observe that all steps, apart from the QUERY PHASE, are simple linear algebra so we ignore the related costs. For the QUERY PHASE, remark that for a minimal codeword it holds that $|I|=d$. Similar to Algorithm 6.4.4, for each $i\in I$ the expected work effort is 2^n, as the probability that a pair (x_i,x_i') with a predetermined XOR-difference collides under f^i is equal to 2^{-n} (and note that $|V_i|>2^n$; hence the collisions are present under f^i).

Now all that remains is to show the correctness. For this we first suppose that the algorithm finds x_i and x_i' for $i\in\tilde{I}$ according to the steps QUERY PHASE and DEGREES OF FREEDOM and computes W and W' as in FINALIZATION. By virtue of Lemma 6.4.3 we are guaranteed that

$$\left(C^{\mathsf{pre}}(W)\right)_i = x_i \quad \text{and} \quad \left(C^{\mathsf{pre}}(W')\right)_i = x_i'$$

Algorithm 6.4.6 (Revised Watanabe Collision Attack)**.**

Input: $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *satisfying* $d \le k$*, a non-zero* $g \in \mathscr{C} \subseteq \mathbb{F}_{2^e}^r$ *with* $|\chi(g)| \le k$*, a block-size* $n = bn'$*, and an arbitrary non-zero* $\delta \in \mathbb{F}_{2^e}^{n'}$*.*

Output: *A colliding pair* $(W, W') \in \left(\{0,1\}^{ekn'}\right)^2$ *such that* $\mathsf{h}_n(W) = \mathsf{h}_n(W')$*,* $W \neq W'$ *and* $C^{\mathsf{pre}}(W) \oplus C^{\mathsf{pre}}(W') = \bar{\rho}(g \otimes \delta)$*.*

1. INITIALIZATION. *Compute* $\Delta = (\Delta_1, \ldots, \Delta_r) \leftarrow \bar{\rho}(g \otimes \delta)$*, set* $I \leftarrow \chi(g)$ *and determine* $\tilde{I} \supseteq I$ *for which* $G_{\tilde{I}}$ *is invertible.*

2. QUERY PHASE. *For* $i \in I$ *do*
   a. *Generate a random* $x_i \xleftarrow{\$} V_i (= \mathbb{F}_2^{en'})$ *and set* $x_i' \leftarrow x_i \oplus \Delta_i$*;*
   b. *Query* $y_i \leftarrow f^i(x_i)$ *and* $y_i' \leftarrow f^i(x_i')$*;*
   c. *If* $y_i = y_i'$ *then keep* (x_i, x_i') *and proceed to the next* i*, else return to step a.*

3. DEGREES OF FREEDOM. *For* $i \in \tilde{I} \backslash I$ *pick randomly* $x_i \xleftarrow{\$} V_i$ *and set* $x_i' \leftarrow x_i$*.*

4. FINALIZATION. *For* $X = (x_{i_1}, \ldots, x_{i_{|\tilde{I}|}})$*,* $X' = (x'_{i_1}, \ldots, x'_{i_{|\tilde{I}|}})$ *and* $\{i_1, \ldots, i_{|\tilde{I}|}\} = \tilde{I}$*, set*

$$W \leftarrow \left(\bar{\varphi}(G_{\tilde{I}}^{-T}) \otimes I_{n'}\right) \cdot X \qquad \text{and} \qquad W' \leftarrow \left(\bar{\varphi}(G_{\tilde{I}}^{-T}) \otimes I_{n'}\right) \cdot X' \,.$$

   Output (W, W')*.*

for $i \in \tilde{I}$. By extension, it follows that

$$\left(C^{\mathsf{pre}}(W)\right)_i \oplus \left(C^{\mathsf{pre}}(W')\right)_i = \Delta_i$$

for $i \in \tilde{I}$ and $\Delta = \bar{\rho}(g \otimes \delta)$. Finally, we need to argue that

$$\left(C^{\mathsf{pre}}(W)\right)_i \oplus \left(C^{\mathsf{pre}}(W')\right)_i = \Delta_i$$

for the remaining $i \in \{1, \ldots, r\} \backslash \tilde{I}$. Yet this follows from Lemma 6.4.3 as well: Given $(\Delta_{i_i}, \ldots, \Delta_{i_{|\tilde{I}|}})$ for $\tilde{I} = \{i_1, \ldots, i_{|\tilde{I}|}\}$ there is a *unique* element Δ^{-1} for which $(C^{\mathsf{pre}}(\Delta^{-1}))_i = \Delta_i$ for $i \in \tilde{I}$. However, because we already know that $\Delta \in \Im(C^{\mathsf{pre}})$, uniqueness would be violated if $\Delta_i \neq \left(C^{\mathsf{pre}}(\Delta^{-1})\right)_i$ for some $i = 1, \ldots, r$. Hence, the claim follows. □

Remark 6.4.8. We note that for MDS codes the two preconditions (i.e., $k \ge d$ and $2k > r$) in Watanabe's attack are equivalent. Indeed, due to the Singleton bound, as $r - k = d - 1$, we get $k \ge d$ if and only if $2k > r$. This does not hold for non-MDS codes in general. Nevertheless, for non-MDS parameters suggested by Knudsen and Preneel, $r - k = d$ is satisfied. Hence, by using a similar argument above, we can obtain the equivalence of the two preconditions.

A New Symbiotic Attack

Our revised version of Watanabe's attack clearly shows that an attacker potentially has much freedom. Below, we transform some of this freedom into a faster attack. More to the point, as in the revised Watanabe attack, we still look for a collision with $\Delta = \bar{\rho}(g \otimes \delta)$ and fix the codeword $g \in \mathscr{C}$, but we do *not* fix the multiplier δ up front. Instead, we determine it based on the outcomes of the queries we make. To increase our success probability, we restrict ourselves to the same kind of

Algorithm 6.4.9 (New Symbiotic Collision Attack)**.**

Input: $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *satisfying* $d \le k$, $g \in \mathscr{C} \subseteq \mathbb{F}_{2^e}^r$ *with* $|\chi(g)| = d$, *and a block-size* $n = bn'$.

Output: *A colliding pair* $(W,W') \in \left(\{0,1\}^{ekn'}\right)^2$ *such that* $\mathsf{h}_n(W) = \mathsf{h}_n(W')$, $W \neq W'$ *and* $C^{\mathsf{pre}}(W) \oplus C^{\mathsf{pre}}(W') = \bar{\rho}(g \otimes \delta)$ *for some non-zero* $\delta \in \mathbb{F}_{2^e}^{n'}$ *to be determined.*

1. Initialization. *Set* $\alpha = d/(d+1)$, $I = \chi(g)$ *and determine* $\tilde{I}$. *Let* $g = (g_1,\ldots,g_r)$ *with* $g_i \in \mathbb{F}_{2^e}$ *for* $i = 1,\ldots,r$.
2. Query Phase. *Define*
$$\mathscr{X} = (\{0\}^{\frac{n}{b}-\frac{\alpha n}{e}} \times \{0,1\}^{\frac{\alpha n}{e}})^e$$
and, for $i \in I$ *let* $\mathscr{Q}[i] = \mathscr{X} \subset V_i$. *Query* f^i *for all* $x_i \in \mathscr{Q}[i]$ *and store the results.*
3. Local Collision Detection. *For* $i \in I$ *create a list* L_i *of all tuples*
$$\left(g_i^{-1} \cdot \rho^{-1}(x_i \oplus x_i'), x_i, x_i'\right)$$
satisfying $x_i, x_i' \in \mathscr{Q}[i]$, $x_i \neq x_i'$ *and* $f^i(x_i) = f^i(x_i')$.
4. Global Collision Detection. *Find a set of* $|\chi(g)|$*-tuples in the respective* L_i *that all share the same first element. That is, for some* $\delta \in \mathbb{F}_{2^e}^{n'}$ *and* $(x_i, x_i')_{i\in I}$ *it holds for all* $i \in I$ *that* $(\delta, x_i, x_i') \in L_i$.
5. Degrees of Freedom. *For* $i \in \tilde{I} \backslash I$ *pick* $x_i \xleftarrow{\$} V_i$ *and set* $x_i' \leftarrow x_i$.
6. Finalization. *For* $X = (x_{i_1},\ldots,x_{i_{|\tilde{I}|}})$, $X' = (x'_{i_1},\ldots,x'_{i_{|\tilde{I}|}})$ *and* $\{i_1,\ldots,i_{|\tilde{I}|}\} = \tilde{I}$, *set*
$$W \leftarrow \left(\bar{\varphi}(G_{\tilde{I}}^{-T}) \otimes I_{n'}\right) \cdot X \qquad \text{and} \qquad W' \leftarrow \left(\bar{\varphi}(G_{\tilde{I}}^{-T}) \otimes I_{n'}\right) \cdot X' .$$
Output (W,W').

queries as in Theorem 3.2.3 (or as in our preimage attacks; hence the name 'symbiotic' to refer to the combination of Watanabe's attack and our preimage attack). The result is an attack (Algorithm 6.4.9) that (ignoring small factors) runs in time $2^{dn/(d+1)}$ provided that $d \le k$ (so the two cases $\mathsf{KP}^1([4,2,3]_8)$ and $\mathsf{KP}^1([5,2,4]_8)$ are excluded as before).

Theorem 6.4.10 (Symbiotic attack)**.** *Let* $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *be given with* $k \ge d$. *Consider* h_n *(with* $b|n$ *and* $b\ n' = n$*). Then Algorithm 6.4.9 finds collisions for* h_n *in* $2^{dn/(d+1)}$ *time (ignoring small factors) using as many PuRF evaluations and memory (expressed in* n*-bit blocks).*

Proof. The correctness of the statement follows from the proof of Theorem 6.4.7; here we only prove that a collision is expected and that the query and time-complexities are as claimed. Because $|\mathscr{X}| = 2^{\alpha n}$ by construction, the attack has the stated query-complexity (per PuRF) for $\alpha = d/(d+1)$ as all queries are made during the Query Phase. Using a naive approach, the Local Collision Detection step can be performed in roughly $2^{dn/(d+1)}$ comparisons, resulting in partial collision lists of expected cardinality $|L_i| \approx 2^{(2\alpha-1)n}$ for $i \in I$.

For Global Collision Detection, we simply enumerate one partial collision list and check for membership against the others. Assuming constant time memory access, the time complexity of this step is at most $(d-1)\max_{i\in I}\{|L_i|\}$. As $\alpha < 1$ it follows that $2\alpha - 1 < \alpha$ making the Query Phase

dominant with its time-complexity of $2^{\alpha n}$. Because we have d active PuRFs in total, the probability of finding a common element among d such lists is then

$$\frac{\prod_{i\in I}|L_i|}{|\mathscr{X}|^{d-1}} = 2^{((2\alpha-1)d-\alpha(d-1))n}\,.$$

To ensure an expected number of collisions of one, we need the second exponent to be at least zero, and indeed, solving for zero gives the desired $\alpha = d/(d+1)$. □

6.4.3 A Parametrized Collision-Finding Attack

The symbiotic attack and the information-theoretic attack given in Section 6.2 have different query complexities and which one is the best seems very parameter dependent. However, it turns out that both attacks are the extreme cases of a more general parametrized attack, as given by Algorithm 6.4.12 (which contains a family of collision-finding attacks based on the number of *active* PuRFs).

Recall that we call a PuRF f^i active, if for any $W \neq W'$ (that satisfies $\mathsf{h}_n(W) = \mathsf{h}_n(W')$), we have $x_i \neq x'_i$. Algorithm 6.4.12 is analyzed in Theorem 6.4.11, where we explain how we determine the query-complexity of the parametrized collision attacks, given the number of *active* PuRFs. More specifically, we investigate the query-complexity separately for the cases where the number of active PuRFs is more than or at most k. The search for the optimal query-complexity—hence the corresponding number of active PuRFs—is carried out in Corollary 6.4.13. We show that the optimality, which results in a query-complexity of $2^{kn/(3k-r)}$ for h_n, is achieved in the shared boundary; specifically when the active number of PuRFs is equal to k (and $d \leq k$).

Theorem 6.4.11. *Let* $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *be given. Consider* h_n *(with* $b|n$ *and* $b\ n' = n$*). Then collisions for* h_n *can be found with Algorithm 6.4.12 by using* $2^{\alpha n}$ *queries (per PuRF) where*

$$\alpha = \begin{cases} (r-\theta)/(2k-\theta) & \text{for} \quad 0 \leq \theta \leq \min(r-d, r-k)\,; \\ (r-\theta)/(r+k-2\theta) & \text{for} \quad r-k \leq \theta \leq r-d\,. \end{cases}$$

Here $r-\theta$ *is the number of active PuRFs.*

Proof. That the attack has the stated query-complexity follows readily from the usual observation that $|\mathscr{X}| = 2^{\alpha n}$, combined with the computation of α exactly matching the theorem statement. What remains to be shown is that collisions are expected to be produced with a good probability.

For correctness, assume (W, W') is output by the algorithm and consider $X = C^{\mathsf{pre}}(W)$ and $X' = C^{\mathsf{pre}}(W')$. First, Lemma 6.4.3 implies that projecting $(X \oplus X', X, X')$ onto $\bigoplus_{i\in\tilde{I}} V_i$ is in $L_{\tilde{I}}$. If $I \subset \tilde{I}$, then DEGREES OF FREEDOM ensures that further projection to $\bigoplus_{i\in I} V_i$ gives $(\tilde{\Delta}, \tilde{X}, \tilde{X}') \in L_I$. If $\tilde{I} \subset I$, however, the FILTERING guarantees that the aforementioned projection of $(X \oplus X', X, X')$ can be extended to $(\tilde{\Delta}, \tilde{X}, \tilde{X}') \in L_I$. Furthermore (by Lemma 6.4.3) we know that in fact $(\tilde{\Delta}, \tilde{X}, \tilde{X}')$ is a projection of $(X \oplus X', X, X')$. As $L_I \subseteq \tilde{L}_I$ it follows that $(x_i, x'_i) \in L_i$ for $i \in I$ and hence by construction (LOCAL COLLISION DETECTION) that $f^i(x_i) = f^i(x'_i)$ for those i. Finally, observe from the COLLISION PRUNING step that $\tilde{\Delta} + 0 \in \Im(C^{\mathsf{pre}})$ and due to DEGREES OF FREEDOM the projections of $\tilde{\Delta} + 0$ and $X \oplus X'$ onto $\bigoplus_{i\in\tilde{I}} V_i$ are equal. The uniqueness is again guaranteed by Lemma 6.4.3 (and $\tilde{\Delta} + 0 = X \oplus X'$), implying $x_i = x'_i$ for all $i \in I_0$.

Algorithm 6.4.12 (Parameterized Collision Attack)**.**

Input: $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$*, an index set* $I_0 \subset \{1,\ldots,r\}$ *with* $\theta = |I_0|$ *and* $0 \le \theta \le r-d$*, and a block-size* $n = bn'$*.*

Output: *A colliding pair* $(W,W') \in \left(\{0,1\}^{ekn'}\right)^2$ *such that* $\mathsf{h}_n(W) = \mathsf{h}_n(W'), W \neq W'$*, and if* $X = C^{\mathsf{pre}}(W)$ *and* $X' = C^{\mathsf{pre}}(W')$ *then for all* $i \in I_0$ *it holds that* $x_i = x'_i$*.*

1. INITIALIZATION. *Set* $I \leftarrow \{1,\ldots,r\}\backslash I_0$*, determine* $\tilde{I}$ *(see Lemma 6.4.3), and set*

$$\alpha \leftarrow \begin{cases} (r-\theta)/(2k-\theta) & \text{for} \quad 0 \le \theta \le \min(r-k, r-d)\ ; \\ (r-\theta)/(r+k-2\theta) & \text{for} \quad r-k \le \theta \le r-d\ . \end{cases}$$

2. QUERY PHASE. *Define*

$$\mathscr{X} = (\{0\}^{\frac{n}{b}-\frac{\alpha n}{e}} \times \{0,1\}^{\frac{\alpha n}{e}})^e$$

and, for $i \in I$ *let* $\mathscr{Q}[i] = \mathscr{X} \subset V_i$*. Query* f^i *on all* $x_i \in \mathscr{Q}[i]$ *and store the results.*

3. LOCAL COLLISION DETECTION. *For* $i \in I$ *create a list* L_i *of all tuples* $(\Delta_i = x_i \oplus x'_i, x_i, x'_i)$ *satisfying* $x_i, x'_i \in \mathscr{Q}[i], x_i \neq x'_i$ *and* $f^i(x_i) = f^i(x'_i)$*.*

4. MERGE PHASE. *Create* $\tilde{L}_I = \prod_{i\in I} L_i$ *or more to the point*

$$\tilde{L}_I = \left\{ (\Delta, X, X') \,|\, (\Delta, X, X') \in \prod_{i\in I} L_i \right\}\ .$$

5. COLLISION PRUNING. *Create* L_I *consisting precisely of those elements of* $\tilde{L}_I$ *whose first vector (when mapped to the full space) is in* $\Im(C^{\mathsf{pre}})$*. Formally*

$$L_I = \left\{ (\tilde{\Delta}, \tilde{X}, \tilde{X}') | (\tilde{\Delta}, \tilde{X}, \tilde{X}') \in \tilde{L}_I \wedge \tilde{\Delta} + 0 \in \Im(C^{\mathsf{pre}}) \right\}\ .$$

6. FILTERING. *If* $\tilde{I} \subset I$ *then only select* $(\tilde{\Delta}, \tilde{X}, \tilde{X}') \in L_I$ *for which* $\tilde{X}$ *is in the projection of* $\Im(C^{\mathsf{pre}})$ *onto* $\bigoplus_{i\in I} V_i$*. Create* $L_{\tilde{I}}$ *by projecting the selected elements of* L_I *to the subspace* $\bigoplus_{i\in\tilde{I}} V_i$*.*

7. DEGREES OF FREEDOM. *If* $I \subset \tilde{I}$ *then, for* $i \in \tilde{I}\backslash I$ *pick randomly* $x_i \xleftarrow{\$} V_i$ *and set* $x'_i \leftarrow x_i$*. Create* $L_{\tilde{I}}$ *by adding*

$$\left((0, x_{i_1}, x'_{i_1}), \ldots, (0, x_{i_{|\tilde{I}\cap I_0|}}, x'_{i_{|\tilde{I}\cap I_0|}})\right)$$

for $\tilde{I} \cap I_0 = \{i_1, \ldots, i_{|\tilde{I}\cap I_0|}\}$ *to all elements in the* L_I *list.*

8. SKIP. *If* $\tilde{I} = I$ *set* $L_{\tilde{I}} \leftarrow L_I$*.*

9. FINALIZATION. *For some element in* $(\tilde{\Delta}, \tilde{X}, \tilde{X}') \in L_{\tilde{I}}$ *create*

$$W \leftarrow (\bar{\varphi}(G_{\tilde{I}}^{-T}) \otimes I_{n'}) \cdot \tilde{X} \qquad \text{and} \qquad W' \leftarrow (\bar{\varphi}(G_{\tilde{I}}^{-T}) \otimes I_{n'}) \cdot \tilde{X}'$$

and output (W, W')*.*

Let us move on to the number of expected collisions output. As $|\mathscr{X}| = 2^{\alpha n}$, the expected number of local collisions found per active PuRF for $i \in I$ is

$$|L_i| \approx |\mathscr{X}|^2/2^n = 2^{(2\alpha-1)n}\ .$$

Using that $|I| = r - \theta$ we arrive at a total number of potential collisions of

$$|\tilde{L}_I| \approx 2^{(2\alpha-1)(r-\theta)n} .$$

For a true collision to occur, we need to find a pair $(x_i, x'_i)_{i\in\tilde{I}}$ such that their completion to $\{0,1\}^{ern'}$ is a codeword such that $x_i = x'_i$ for $i \in I_0$.

If the eventual collision consists of (X, X'), then $\Delta = X \oplus X'$ is a codeword as well, and the above implies that it satisfies $\Delta_i = 0$ for $i \in I_0$. Lemma 6.4.1 applies and $\tilde{\Delta} = (\Delta_{i_1}, \ldots, \Delta_{i_{|I|}})$ for $I = \{i_1, \ldots, i_{|I|}\}$ is somehow 'spanned' by the shortened code. The restriction $\theta \le r - d$ ensures non-triviality of the shortened code. We note that shortening any further would lead to a shortened code that consists of the zero codeword only; it would result in $W = W'$ (so no collision). In case of MDS codes, the shortened code has parameters $[r-\theta, k-\theta, d']_{2^e}$, in particular it has dimension $k-\theta$. (For non-MDS codes it is possible that a higher dimension is achieved.) As a result, a fraction $2^{(k-r)\alpha n}$ of the $\tilde{\Delta}$ values are satisfactory, leading to an expected cardinality of

$$|L_I| \approx 2^{((2\alpha-1)(r-\theta)-\alpha(r-k))n} .$$

If $I \subseteq \tilde{I}$ or equivalently $r - \theta \le k$, we are done whenever $|L_I| \ge 1$. Because $r - \theta \le k$ can be rewritten as $\theta \ge r - k$, we are in the second case, with $\alpha = (r-\theta)/(r+k-2\theta)$. Writing $F = (\log|L_I|)/n$ and substitution lead to

$$F \approx (2\alpha-1)(r-\theta) - \alpha(r-k) = (r-2\theta+k)(r-\theta)/(r+k-2\theta) - (r-\theta) = 0$$

or $|L_I| \approx 1$ as desired. If $\tilde{I} \subset I$, further filtering is needed. In particular, given a potential 'half' of a collision X, we need to check if it can correspond to a codeword. Because $\tilde{I} \subset I$, we can uniquely complete X to a codeword, given k of its elements (all within I). The remaining $|I| - k$ coordinates need to be in sync. Per remaining element, this occurs with probability $2^{-\alpha n}$, leading to $|\tilde{L}_{\tilde{I}}| \approx |L_I| \cdot 2^{-\alpha n(r-\theta-k)}$. We are in the first case as $0 \le \theta \le r - k$. Writing $F = (\log \tilde{L}_{\tilde{I}})/n$, we obtain

$$F \approx ((2\alpha-1)(r-\theta) - \alpha(r-k)) - \alpha(r-\theta-k) = \alpha(2k-\theta) - (r-\theta).$$

As we aim for $F = 0$, it follows that $\alpha = (r-\theta)/(2k-\theta)$ as desired. Hence the claim follows. □

Corollary 6.4.13. *Assuming $d \le k$, substitution of $\theta = r - k$ in Theorem 6.4.11 gives $\alpha = k/(3k-r)$. This is optimal (for Algorithm 6.4.12) whenever $r \le 2k$.*

Proof. That the substitution does what it says can be readily verified, so we restrict ourselves to proving the optimality here. Let $f_1(\theta) = (r-\theta)/(2k-\theta)$ and $f_2(\theta) = (r-\theta)/(r+k-2\theta)$ be two real valued functions defined over closed intervals $0 \le \theta \le r-k$ and $r-k \le \theta \le r-d$, respectively. Note that both $f_1(\theta)$ and $f_2(\theta)$ are continuous in their respective domains (because their respective poles fall outside the domains). So both $f_1(\theta)$ and $f_2(\theta)$ attain their maximum and minimum in the closed intervals $[0, r-k]$ and $[r-k, r-d]$, respectively. As $f_1'(\theta) = (r-2k)/(2k-\theta)^2 \le 0$ (for $r \le 2k$) and $f_2'(\theta) = (r-k)/(r+k-2\theta)^2 \ge 0$, we can conclude that $f_1(\theta)$ decreases and $f_2(\theta)$ increases. Therefore, they both attain their minimum at their shared boundary $\theta = r - k$. □

Remark 6.4.14. The only two parameter sets proposed by Knudsen and Preneel, which do not satisfy the conditions of the corollary above, are $[4,2,3]_8$ and $[5,2,4]_8$. In both cases $d > k$ and only $f_1(\theta)$ is

applicable. For $[5,2,4]_8$ we can check that $2k < r$ and $f_1'(\theta) \geq 0$. Hence, the minimum α is attained at $\theta = 0$. For $[4,2,3]_8$ it holds that $2k = r$, so that $f_1(\theta)$ is in fact a constant function and both $\theta = 0$ and $\theta = 1$ lead to the same α. At one extreme, the substitution of $\theta = 0$ in Theorem 6.4.11 gives $\alpha = r/(2k)$ and the resulting query-complexity coincides with that reported in Proposition 6.2.1. At the other extreme, substitution of $\theta = r - k$ gives $\alpha = d/(2d - r + k)$ (assuming $d \leq k$). For MDS codes, this simplifies to $\alpha = d/(d+1)$, this time duly coinciding with our symbiotic attack. For non-MDS codes, there seems to be a slight mismatch. The reason is that if a non-MDS code is maximally shortened (by $\theta = r - d$), the shortened code has dimension one, whereas in the derivation of Theorem 6.4.11 we pessimistically assumed $k - \theta = 0$ (at least for the KP non-MDS codes that satisfy $r - d = k$). Correcting for this looseness would result in a match with the symbiotic attack.

6.4.4 Practical Collision Attack Against $\mathsf{KP}^1([5,3,3]_4)$ in $\mathcal{O}(2^{3n/4})$ Time

When using Algorithm 6.4.12, finding a collision for $\mathsf{h} = \mathsf{KP}^1([5,3,3]_4)$ with block-size n and stated (optimal) query-complexity (for $\alpha = 3/4$) takes (asymptotically in n) $\mathcal{O}(2^{3n/2})$ time (due to MERGE PHASE and COLLISION PRUNING). In this section, we follow in the footsteps of Algorithm 6.4.12 (with fixed $\alpha = k/(3k - r)$ obtained in Corollary 6.4.13) and try to reduce its time requirements. Our goal is to present a collision-finding algorithm with time-complexity almost coinciding with the targeted query-complexity. In an ideal scenario, this would provide an almost optimal attack (with specified α) for many of the parameter sets suggested by Knudsen and Preneel.

Note that, for this specific compression function, our ideal time-complexity is the same as the one achieved by the symbiotic attack presented in Algorithm 6.4.9. Although the attack presented in this section results in better time-complexity for most of the KP-suggested parameters, this is one of the few exceptions where we get the same time-complexity. We choose this particular compression function to make our attack illustration easier to understand.

Claim 6.4.15. *For the Knudsen–Preneel compression function* $\mathsf{h} = \mathsf{KP}^1([5,3,3]_4)$*, collisions for* h_n *can be found in* $\mathcal{O}(2^{3n/4})$ *time and memory (n-bit blocks) asymptotically in n.*

Proof. In this attack, we generate two different inputs W and W' for h such that $\mathsf{h}_n(W) = \mathsf{h}_n(W')$. It follows from $\theta = 2$ and $r = 5$ that we have three active PuRFs. Without loss of generality, let the first three PuRFs be active (corresponding to the systematic portion), namely $I = \{1,2,3\}$ (so $\tilde{I} = I$ and $I_0 = \{4,5\}$). To complete a collision, we need to generate the PuRF inputs $x_i = (x_i^1 || x_i^2)$, $x_i' = (x_i'^1 || x_i'^2) \in \{0,1\}^{2n}$ such that $f^i(x_i) = f^i(x_i')$ for all $i = 1,\ldots,5$ where $x_i \neq x_i'$ for $i = 1,2,3$ and $x_i = x_i'$ for $i = 4,5$ (hence $\theta = 2$). An outline of the attack is given in Figure 6.4, we proceed with the details.

As previously argued, finding such input pairs is equivalent to finding two codewords X and X' in $\Im(C^{\mathsf{pre}})$ for which the partial components collide under smaller PuRFs. Due to linearity, we know that $\Delta = X \oplus X'$ is also a non-zero codeword satisfying $\Delta_4 = \Delta_5 = 0$ for $\Delta = (\Delta_1,\ldots,\Delta_5)$. Observe that once we find such a difference Δ of this form (i.e., $\Delta \in \Im(C^{\mathsf{pre}})$ such that $\Delta_4 = \Delta_5 = 0$, or more formally $\Delta = \Delta' + 0$ for $\Delta' = (\Delta_1, \Delta_2, \Delta_3)$) together with the corresponding PuRF input pairs (x_i, x_i') for $i \in \tilde{I}$, we can easily finalize the global collision regardless of the values (x_i, x_i') for $i = 4,5$; it follows from the fact that both X and X' will be the encoded strings of the inputs W and W', respectively, (hence they are guaranteed to be true codewords) while automatically having $\Delta_4 = \Delta_5 = 0$. Note that the

uniqueness of such a difference follows from Lemma 6.4.3. So, our task is to find such a valid Δ.

We begin our attack with determining the tuples (Δ_i, x_i, x_i') subject to the condition that $x_i \oplus x_i' = \Delta_i \neq 0$ and $f^i(x_i) = f^i(x_i')$ (for all $i = 1,2,3$). In the QUERY PHASE, we query $f^i(x_i)$ for all

$$x_i^1, x_i^2 \in 0^{5n/8} \times \{0,1\}^{3n/8}$$

and then keep a list L_i of pairs (Δ_i, x_i, x_i') (where $x_i \oplus x_i' = \Delta_i \neq 0$) that collide under f_i for each $i = 1,2,3$. The latter phase is the LOCAL COLLISION DETECTION step (see Algorithm 6.4.12). As a result, a total of $2^{3n/4}$ queries are asked per PuRF, which results in partial collision lists of cardinality

$$|L_i| \approx (2^{3n/4})^2/2^n = 2^{n/2}$$

for each $i = 1,2,3$. Note that the steps QUERY PHASE and LOCAL COLLISION DETECTION can be performed (asymptotically in n) in conjunction in $\mathcal{O}(2^{3n/4})$ time (as many PuRF evaluations) and memory (in n-bit blocks).

To complete the collision, we need to determine whether each newly generated Δ is contained in $\Im(C^{\mathsf{pre}})$ or not. Thanks to the special form of Δ (namely $\Delta_4 = \Delta_5 = 0$), rather than searching for membership in $\Im(C^{\mathsf{pre}})$ (specified by the code $\mathscr{C}$), we can work on a smaller space $\Im(C'^{\mathsf{pre}})$ identified by $\mathscr{C}'$ (that is the shortened code obtained by dropping the zeros of the codewords from all the positions appearing in I_0). Indeed, for any non-zero $\Delta \in \Im(C^{\mathsf{pre}})$ of the form $\Delta = \Delta' + 0$ for $\Delta' = (\Delta_1, \Delta_2, \Delta_3)$, we have that $\Delta' \in \Im(C'^{\mathsf{pre}})$. Hence, we can now make an easier membership-check in the smaller space $\Im(C'^{\mathsf{pre}})$. Here, $\mathscr{C}'$ is the $[3,1,3]_4$ MDS code with a generator matrix

$$G = \begin{pmatrix} 1 & w & 1 \end{pmatrix}, \text{where we use the mapping} \quad \varphi(1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ \varphi(w) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

to define its shortened image $\Im(C'^{\mathsf{pre}})$. We limit ourselves to the novel shortened string $\Delta' = (\Delta_1, \Delta_2, \Delta_3)$ and check whether it is in $\Im(C'^{\mathsf{pre}})$, or not. Let us define (for $\Delta_i = (\Delta_i^1 || \Delta_i^2)$ where $\Delta_i^1, \Delta_i^2 \in \{0,1\}^n$)

$$L_{\{1,2\}} = \left\{ \left((\Delta_1, x_1, x_1'), (\Delta_2, x_2, x_2') \right) \in L_1 \times L_2 \mid \Delta_1^1 \oplus \Delta_1^2 = \Delta_2^1 \text{ and } \Delta_1^1 = \Delta_2^2 \right\} .$$

Note that $L_{\{1,2\}}$ contains the elements that partially satisfy the linear constraints; that is,

$$L_{\{1,2\}} = \Im(C'^{\mathsf{pre}}) \cap \prod_{i=1}^{2} L_i \ .$$

We can construct $L_{\{1,2\}}$ efficiently as done in our preimage attacks. It starts with the MERGE PHASE (see Algorithm 6.4.16), where we create [5] the lists $\tilde{L}_{\{1\}}$ and $\tilde{L}_{\{2\}}$ defined by

$$\tilde{L}_{\{1\}} = \left\{ \left(\Delta_1, x_1, x_1', \left((\Delta_1^1 \oplus \Delta_1^2) || \Delta_1^1 \right) \right) \mid (\Delta_1, x_1, x_1') \in L_1 \right\}$$

$$\tilde{L}_{\{2\}} = \left\{ \left(\Delta_2, x_2, x_2', (\Delta_2^1 || \Delta_2^2) \right) \mid (\Delta_2, x_2, x_2') \in L_2 \right\}$$

both sorted on their fourth component. In the JOIN PHASE we look for the collisions in their fourth components to determine possible Δ' candidates. As $|L_i| \approx 2^{n/2}$, creating $\tilde{L}_{\{1\}}$ and $\tilde{L}_{\{2\}}$ takes about

5. Normally MERGE PHASE contains merging of several lists which is not the case for our example due to the relatively small minimum distance of the dual code of $\mathscr{C}'$, see discussion in Section 6.3.2.

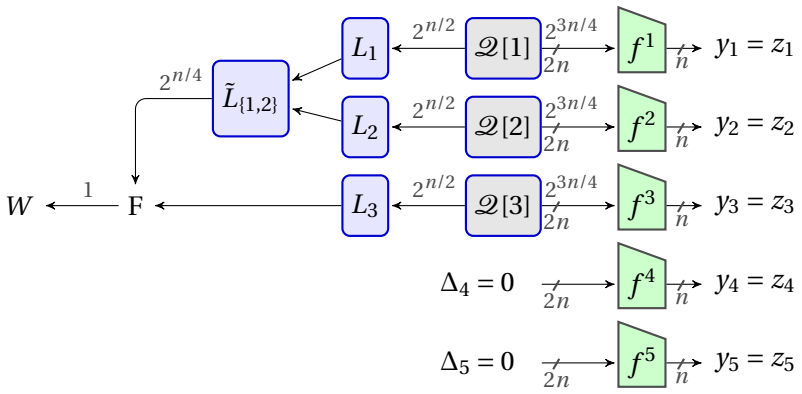

Figure 6.4 – Our collision attack on $\mathsf{h}_n = \mathsf{KP}^1([5,3,3]_4)$ illustrated. The (unlabeled) inputs to $f^1,\ldots,f^5$ correspond to $(x_1^1,x_1^2),\ldots,(x_5^1,x_5^2)$. Here, F denotes the final filtering.

$\mathcal{O}(n2^{n/2})$ time and memory, asymptotically in n. (In general, the smaller of the two is sorted and stored and the other is used for collision check.) As $\tilde{L}_{\{1\}}$ and $\tilde{L}_{\{2\}}$ both have roughly $2^{n/2}$ elements and they need to collide on $2n$ bits, of which $5n/4$ bits are zero, the expected number of collisions is

$$(2^{n/2})^2/2^{(2-5/4)n} = 2^{n/4} = |L_{\{1,2\}}|\,.$$

Now, we have identified roughly $2^{n/4}$ candidate Δ' values that are possibly in $\Im(C'^{\mathsf{pre}})$. All that needs to be done now is to check, for each candidate, whether the corresponding Δ_3 is contained in L_3. If this is the case, then Δ' is in $\Im(C'^{\mathsf{pre}})$; so the corresponding $\Delta \in \Im(C^{\mathsf{pre}})$ as well. This membership-check is performed in the Collision Pruning phase. It is clear that it cannot take much longer than it took to create $L_{\{1,2\}}$. Moreover, the expected number of remaining candidates is one; note that $|L_{\{1,2\}}| \approx 2^{n/4}$ and $|L_3| \approx 2^{n/2}$. Again, we need to check the correspondence on $2n$ bits, of which $5n/4$ are set to zero. Hence, we do expect to find $2^{(1/4+1/2)n}/2^{(2-5/4)n} = 1$ difference Δ. Finally, as we are given a valid Δ and the corresponding input pairs (x_i, x_i') for $i \in \tilde{I}$ we can set the collision pair (W,W') uniquely (as done in Algorithm 6.4.16). Summing up the stepwise time and memory complexities gives the desired result. □

6.4.5 Generic Collision Attack Against MDS Constructions

If we want to run Algorithm 6.4.12 (with fixed $\theta = r-k$ and $\alpha = k/(3k-r)$ as obtained in Corollary 6.4.13) we ideally want a time-complexity almost coinciding with the targeted query-complexity. For $\theta = r-k$, it holds that $I = \tilde{I}$, obviating the need for the steps Filtering and Degrees of Freedom. We have seen previously that Local Collision Detection increases the runtime by a small factor that is logarithmic in n, which leaves only the Merge Phase and Collision Pruning to worry about. Together, these two steps are designed to produce L_I. A naive approach would enumerate all elements in the much larger $\tilde{L}_I$, which is wasteful. Our task is therefore, given the lists of partial collisions L_i for $i \in I$, to create L_I more efficiently.

Here, we will follow the footsteps of Section 6.3.2 where the dual code is used in a similar problem related to the preimage-finding attack. An important innovation for the collision-finding attack stems from the realization that Δ can be regarded as belonging to the (quasi) shortened code. This

allows for the use of the dual code of the shortened code to speed up the search. As the minimum distance of the dual code is an important parameter in determining the overall time-complexity and shortening a code reduces the minimum distance of its dual accordingly, we make a significant efficiency gain this way.

As $I = \tilde{I}$ and $\theta = r - k$, we know from Algorithm 6.4.12 that it is enough to find a non-zero $\Delta \in \Im(C^{\mathsf{pre}})$ of the form $\Delta = \Delta' + 0$ for $\Delta' = (\Delta_{i_1}, \ldots, \Delta_{i_{|\tilde{I}|}})$ and $\tilde{I} = \{i_1, \ldots, i_{|\tilde{I}|}\}$ to complete the collision (see also Section 6.4.4). Notice that Δ' lies in a smaller space $\Im(C'^{\mathsf{pre}})$ identified by $\mathscr{C}'$ that is the $[r-\theta, k-\theta, d']$ shortened code obtained from $\mathscr{C}$ (by dropping the zeros of the codewords from all the positions appearing in I_0). This observation allows us to guarantee that $\Delta \in \Im(C^{\mathsf{pre}})$ once we determine that a candidate Δ' is in $\Im(C'^{\mathsf{pre}})$. Hence, it is enough for our purposes to limit ourselves to $\Im(C'^{\mathsf{pre}})$ rather than looking for membership in the larger space $\Im(C^{\mathsf{pre}})$.

To this end, we first identify an index set $\mathscr{I}_{h'} \subseteq \{1, \ldots, r\}$ (the role of h' will be explained momentarily) defining a subspace $\bigoplus_{i \in \mathscr{I}_{h'}} V_i$ for which $\Im(C'^{\mathsf{pre}})$ when restricted to this subspace, is not surjective. As a consequence, we will be able to prune significantly the total collection of candidate Δ' values keeping only those that are possibly in $\Im(C'^{\mathsf{pre}})$ (restricted to $\bigoplus_{i \in \mathscr{I}_{h'}} V_i$). Next, we will show how to *efficiently* find an index set $\mathscr{I}_{h'}$, and how to *efficiently* prune.

An important parameter determining the runtime of our collision attack is $d'^{\perp}$, the minimum distance of the dual code of the shortened code. Let χ be the function that maps $h' \in \mathbb{F}_{2^e}^{r-\theta}$ to the set of indices of non-zero entries in h'. Thus, $\chi(h') \subseteq \{1, \ldots, r\}$ and $|\chi(h')|$ equals the Hamming weight of the codeword h'. An easy adaptation of Proposition 6.3.3 shows that if we are given a codeword $h' \in \mathscr{C}'^{\perp}$ and an element $\Delta' \in \mathbb{F}_2^{(r-\theta)en'}$, then Δ' can only be in $\Im(C'^{\mathsf{pre}})$ if $(\bar{\varphi}(h'^T) \otimes I_{n'}) \cdot \Delta' = 0$, where the only parts of Δ' relevant for this check are those lining up with the non-zero entries of h'. Indeed, an element $\Delta'_{h'} \in \prod_{i \in \chi(h')} L_i$ can be completed to an element in the range of derived mapping C'^{pre} if and only if $(\bar{\varphi}(h'^T) \otimes I_{n'}) \cdot (\Delta'_{h'} + 0) = 0$. An efficient creation of

$$L_{h'} = \left\{ (\Delta'_{h'}, X, X') \in \prod_{i \in \chi(h')} L_i \mid (\bar{\varphi}(h'^T) \otimes I_{n'}) \cdot (\Delta'_{h'} + 0) = 0 \right\}$$

can be done as in Algorithm 6.3.4 by splitting the codeword in two and looking for all collisions in respective entries. That is, assume that $h' = h'_0 + h'_1$ with $\chi(h'_0) \cap \chi(h'_1) = \emptyset$, and define, for $j = 0, 1$

$$\tilde{L}_{h'_j} = \left\{ \left(\Delta'_{h'_j}, X_j, X'_j, (\bar{\varphi}(h'^T_j) \otimes I_{n'}) \cdot (\Delta'_{h'_j} + 0) \right) \mid (\Delta'_{h'_j}, X_j, X'_j) \in \prod_{i \in \chi(h'_j)} L_i \right\}.$$

Then $L_{h'}$ consists of those elements $\Delta'_{h'_0} + \Delta'_{h'_1}$ for which $(\Delta'_{h'_0}, X_0, X'_0, Y_0) \in \tilde{L}_{h'_0}$, $(\Delta'_{h'_1}, X_1, X'_1, Y_1) \in \tilde{L}_{h'_1}$ and $Y_0 = Y_1$. By the same method as in our preimage attacks, the time-complexity of creating $L_{h'}$ is then roughly the maximum cardinality of the two sets $\tilde{L}_{h'_0}$ and $\tilde{L}_{h'_1}$. Hence, again the main trick to reduce the time-complexity is to minimize the Hamming weights of h'_0 and h'_1, which is done by choosing a codeword $h' \in \mathscr{C}'^{\perp}$ of minimum distance $d'^{\perp}$ and splitting h' (almost) evenly (into h'_0 and h'_1). The complete collision-finding algorithm is given in Algorithm 6.4.16. We summarize our analysis in Theorem 6.4.17 whose proof is given in Section 6.4.7.

Theorem 6.4.17. *Let* $\mathsf{h} = \mathsf{KP}^b([r, k, d]_{2^e})$ *be given and* $\mathscr{C}'$ *be a shortened* $[r-\theta, k-\theta, d]_{2^e}$ *code derived from* $\mathscr{C}$ *for* $\theta = r - k$. *Let* $d'^{\perp}$ *be the minimum distance of the dual code of* $\mathscr{C}'$. *Suppose* $\mathscr{C}$ *is MDS (so is*

Algorithm 6.4.16 (Collision Attack against MDS-based schemes)**.**

Input: $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$*, an index set* $I_0 \subset \{1,\ldots,r\}$ *with* $\theta = |I_0| = r-k$ *and a block-size* $n = bn'$*.*

Output: *A colliding pair* $(W,W') \in \left(\{0,1\}^{ekn'}\right)^2$ *such that* $\mathsf{h}_n(W) = \mathsf{h}_n(W'), W \neq W'$*, and if* $X = C^{\mathsf{pre}}(W)$ *and* $X' = C^{\mathsf{pre}}(W')$ *then for all* $i \in I_0$ *it holds that* $x_i = x'_i$*.*

1. INITIALIZATION. *Set* $I \leftarrow \{1,\ldots,r\}\backslash I_0$ *(with* $|I| = k$*),* $I = \tilde{I}$*, and set* $\alpha \leftarrow k/(3k-r)$*. Obtain* $\mathscr{C}'$ *consisting of codewords* $g' \in \mathscr{C}'$ *that are constructed from* $g \in \mathscr{C}$ *by dropping zeros of* g *from all the positions appearing in* I_0*.*
2. QUERY PHASE. *As in Algorithm 6.4.12.*
3. LOCAL COLLISION DETECTION. *As in Algorithm 6.4.12.*
4. (FIRST) MERGE PHASE. *Find a non-zero codeword* $h' \in \mathscr{C}'^{\perp}$ *of minimum Hamming weight* $d'^{\perp} = 2k-r+1$*. Let* $h' = h'_0 + h'_1$ *with* $\chi(h'_0) \cap \chi(h'_1) = \emptyset$ *and of Hamming weights* $\lceil d'^{\perp}/2 \rceil$ *and* $\lfloor d'^{\perp}/2 \rfloor$*, respectively. Create for* $j = 0,1$*,*

$$\tilde{L}_{h'_j} = \left\{ \left(\Delta'_{h'_j} = X_j \oplus X'_j, X_j, X'_j, (\bar{\varphi}(h'_j) \otimes I_{n'}) \cdot (\Delta'_{h'_j} + 0) \right) \mid (\Delta'_{h'_j}, X_j, X'_j) \in \prod_{i \in \chi(h'_j)} L_i \right\}$$

   both sorted on their fourth component.
5. (FIRST) JOIN PHASE. *Create* $L_{h'}$ *consisting exactly of those elements* $\Delta'_{h'_0} + \Delta'_{h'_1}$ *for which* $(\Delta'_{h'_0}, X_0, X'_0, Y_0) \in \tilde{L}_{h'_0}, (\Delta'_{h'_1}, X_1, X'_1, Y_1) \in \tilde{L}_{h'_1}$ *and* $Y_0 = Y_1$*.*
6. COLLISION PRUNING. *For all* $(\Delta'_{h'}, X, X') \in L_{h'}$ *create the unique* Δ' *corresponding to it and check whether it results in* $\Delta_i \in L_i$ *for all* $i \in I(=\tilde{I})$*. If so, keep* $\Delta' = (\Delta_{i_1}, \ldots, \Delta_{i_{|\tilde{I}|}})$ *(for* $\tilde{I} = \{i_1,\ldots,i_{|\tilde{I}|}\}$*) in* L_I*. Formally*

$$L_I = \left\{ (\Delta', \tilde{X}, \tilde{X}') = (\Delta'_{h'}, X, X') \in L_{h'} + \prod_{i \in \tilde{I}\backslash\chi(h')} L_i \mid \Delta' \in \Im(C'^{\mathsf{pre}}) \right\}.$$

7. SKIP. *As in Algorithm 6.4.12.*
8. FINALIZATION. *As in Algorithm 6.4.12.*

$\mathscr{C}'$ with $d'^{\perp} = 2k-r+1$) and consider the collision attack described in Algorithm 6.4.16 run against h_n *by using* $q = 2^{\alpha n}$ *queries for* $\alpha = k/(3k-r)$*. Then the expected number of collision outputs is equal to one and the expectations for the internal list sizes are (for* $i \in I$*):*

$$|L_i| = 2^{(2\alpha-1)n}, \quad |L_{h'}| = 2^{((2\alpha-1)d'^{\perp}-\alpha)n}, \quad |\tilde{L}_{h'_0}| = 2^{(2\alpha-1)\lceil \frac{d'^{\perp}}{2} \rceil n}, \quad |\tilde{L}_{h'_1}| = 2^{(2\alpha-1)\lfloor \frac{d'^{\perp}}{2} \rfloor n}.$$

The average case time-complexity of the algorithm is a small constant multiple of $\max\left(q, |\tilde{L}_{h'_0}|, |L_{h'}|\right)$ *with a memory requirement of* $\max\left(q, |\tilde{L}_{h'_1}|\right)$ *(expressed in* en/b*-bit blocks).*

Table 6.6 contains an overview for the various cardinalities and complexities for all the compression functions based on MDS codes that were suggested by Knudsen and Preneel [93].

Code		Query Complexity	Cardinalities related to our attack				Complexity Time	Memory
$[r,k,d]_{2^e}$	$d'^{\perp}$	$\lvert\mathscr{Q}[i]\rvert$	$\lvert L_i\rvert$	$\lvert\tilde{L}_{h'_0}\rvert$	$\lvert\tilde{L}_{h'_1}\rvert$	$\lvert L_{h'}\rvert$	Theorem 6.4.17	
$[5,3,3]_4$	2	$2^{3n/4}$	$2^{n/2}$	$2^{n/2}$	$2^{n/2}$	$2^{n/4}$	$2^{3n/4}$	$2^{3n/4}$
$[6,4,3]_{16}$	3	$2^{2n/3}$	$2^{n/3}$	$2^{2n/3}$	$2^{n/3}$	$2^{n/3}$	$2^{2n/3}$	$2^{2n/3}$
$[8,6,3]_{16}$	4	$2^{3n/5}$	$2^{n/5}$	$2^{2n/5}$	$2^{2n/5}$	$2^{2n/5}$	$2^{3n/5}$	$2^{3n/5}$
$[12,10,3]_{16}$	9	$2^{5n/9}$	$2^{n/9}$	$2^{5n/9}$	$2^{4n/9}$	$2^{4n/9}$	$2^{5n/9}$	$2^{5n/9}$
$[9,6,4]_{16}$	4	$2^{2n/3}$	$2^{n/3}$	$2^{2n/3}$	$2^{2n/3}$	$2^{2n/3}$	$2^{2n/3}$	$2^{2n/3}$
$[16,13,4]_{16}$	11	$2^{13n/23}$	$2^{3n/23}$	$2^{18n/23}$	$2^{15n/23}$	$2^{20n/23}$	$2^{20n/23}$	$2^{15n/23}$
$[6,4,3]_8$	3	$2^{2n/3}$	$2^{n/3}$	$2^{2n/3}$	$2^{n/3}$	$2^{n/3}$	$2^{2n/3}$	$2^{2n/3}$
$[9,7,3]_8$	6	$2^{7n/12}$	$2^{n/6}$	$2^{n/2}$	$2^{n/2}$	$2^{5n/12}$	$2^{7n/12}$	$2^{7n/12}$
$[7,4,4]_8$	2	$2^{4n/5}$	$2^{3n/5}$	$2^{3n/5}$	$2^{3n/5}$	$2^{2n/5}$	$2^{4n/5}$	$2^{4n/5}$
$[10,7,4]_8$	5	$2^{7n/11}$	$2^{3n/11}$	$2^{9n/11}$	$2^{6n/11}$	$2^{8n/11}$	$2^{9n/11}$	$2^{7n/11}$

Table 6.6 – An overview of the list cardinalities and computational complexity of collision attacks on the Knudsen–Preneel compression functions based on on MDS codes.

6.4.6 Extending the Collision Attack Against Non-MDS Constructions

For non-MDS codes we can try to mount the collision attack given by Algorithm 6.4.16, but after the JOIN PHASE we may encounter a similar problem as our preimage attack: For some non-MDS codes, the map C'^{pre} restricted to $\bigoplus_{i\in\chi(h')} V_i$ is not injective and we can no longer reconstruct a unique (W,W') corresponding to some $(X,X') \in L_{h'}$. Here, we fix this issue as in Section 6.3.3 by performing a second stage of MERGEing and JOINing.

In Algorithm 6.4.18 we simply paste in extra MERGE and JOIN phases in order to maintain the low complexity. We only include one extra merge-join phase for non-MDS codes. For the parameters proposed by Knudsen and Preneel, this always suffices. For other parameters possibly extra merge-join phases are required before full rank is achieved, we did not investigate this. Note also that there is another issue regarding non-MDS parameters. This time we do not have enough freedom to arbitrarily choose the k active PuRFs in the INITIALIZATION step, unlike the MDS case. This is mainly because of the fact that k arbitrary columns of the generator matrix are not necessarily linearly independent. Nevertheless, it is still possible to find such an *admissible I* for the non-MDS parameters suggested by Knudsen and Preneel. For simplicity, we assign (without loss of generality) $I = \tilde{I}$; it is the set of positions corresponding to the systematic portion of the code. This assumption is always sufficient for our attacks to work for all given parameters.

Theorem 6.4.19. *Let* $[r,k,d] \in \{[8,5,3],[12,9,3],[9,5,4],[16,12,4]\}$ *be given along with a generator matrix G for* $[r,k,d]_4$ *(as given by Magma's BKLC routine); Let* $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$ *be given and* $\mathscr{C}'$ *be a shortened* $[r-\theta,k-\theta,d]_{2^e}$ *code derived from* $\mathscr{C}$ *for* $\theta = r-k$*. Let* $d'^{\perp}$ *be the minimum distance of the dual code of* $\mathscr{C}'$*. Suppose* $\mathscr{C}$ *is not an MDS code and consider the collision attack described in Algorithm 6.4.18 run against* h_n *using* $q = 2^{\alpha n}$ *queries for* $\alpha = k/(3k-r)$ *(i.e.,* $|\mathscr{X}| = 2^{kn/(3k-r)}$*). Then, the expected number of collision outputs is equal to one and the expectations for the internal list sizes for the first merge-join are as before (see Theorem 6.4.17) and for the second merge-join phase*

$$\max(|\tilde{L}_{h''_0}|,|\tilde{L}_{h''_1}|) \le 2^{T_1 n}\ ,\ \min(|\tilde{L}_{h''_0}|,|\tilde{L}_{h''_1}|) \le 2^{T_2 n}\ ,\ |L_{h''}| \le 2^{((2\alpha-1)(2k-r+2)-2\alpha)n}\ ,$$

Algorithm 6.4.18 (Collision attack against non-MDS-based schemes)**.**

Input: $\mathsf{h} = \mathsf{KP}^b([r,k,d]_{2^e})$*, an index set* $I_0 \subset \{1,\ldots,r\}$ *with* $\theta = |I_0| = r-k$ *and a block-size* $n = bn'$*.*

Output: *A colliding pair* $(W,W') \in \left(\{0,1\}^{ekn'}\right)^2$ *such that* $\mathsf{h}_n(W) = \mathsf{h}_n(W'), W \neq W'$*, and if* $X = C^{\mathsf{pre}}(W)$ *and* $X' = C^{\mathsf{pre}}(W')$ *then for all* $i \in I_0$ *it holds that* $x_i = x'_i$*.*

1. INITIALIZATION. *As in Algorithm 6.4.16.*
2. QUERY PHASE. *As in Algorithm 6.4.16.*
3. FIRST MERGE PHASE. *As in Algorithm 6.4.16.*
4. FIRST JOIN PHASE. *As in Algorithm 6.4.16.*

   If $d'^{\perp} > k'$*, go to step 7, else proceed with the next step.*
5. SECOND MERGE PHASE. *Find a codeword* $h'' \in C'^{\perp} \setminus \mathbb{F}_{2^e} h'$ *of minimum Hamming weight (possibly exceeding* $d'^{\perp}$*). Let* $h'' = h''_0 + h''_1$ *with* $\chi(h''_0) \cap \chi(h''_1) = \emptyset$*,* $\chi(h''_1) \cap \chi(h') = \emptyset$*, and of Hamming weights yet to be determined. Create,*

$$\tilde{L}_{h''_0} = \left\{ \left(\Delta'_{h''_0}, X_0, X'_0, (\bar{\varphi}(h''_0) \otimes I_{n'}) \cdot (\Delta'_{h''_0} + 0)\right) \mid (\Delta'_{h''_0}, X_0, X'_0) \in L_{h'} + \prod_{i \in \chi(h''_0) \setminus \chi(h')} L_i \right\}$$

$$\tilde{L}_{h''_1} = \left\{ \left(\Delta'_{h''_1}, X_1, X'_1, (\bar{\varphi}(h''_1) \otimes I_{n'}) \cdot (\Delta'_{h''_1} + 0)\right) \mid (\Delta'_{h''_1}, X_1, X'_1) \in \prod_{i \in \chi(h''_1)} L_i \right\}.$$

6. SECOND JOIN PHASE. *Create* $L_{h''}$ *consisting exactly of those elements* $\Delta'_{h''_0} + \Delta'_{h''_1}$ *for which* $(\Delta'_{h''_0}, X_0, X'_0, Y_0) \in \tilde{L}_{h''_0}, (\Delta'_{h''_1}, X_1, X'_1, Y_1) \in \tilde{L}_{h''_1}$ *and* $Y_0 = Y_1$*.*
7. COLLISION PRUNING. *For all* $(\Delta'_{h''}, X, X') \in L_{h''}$ *(or in* $L_{h'}$ *if* $d'^{\perp} > k'$*) create the unique* Δ' *corresponding to it and check whether it results in* $\Delta_i \in L_i$ *for all* $i \in I (= \tilde{I})$*. If so, keep* $\Delta' = \prod_{i \in \tilde{I}} \Delta_i$ *in* L_I*. Formally, (or as in Algorithm 6.4.16 if* $d'^{\perp} \geq k'$*)*

$$L_I = \left\{ (\Delta', \tilde{X}, \tilde{X}') = (\Delta'_{h''}, X, X') \in L_{h''} + \prod_{i \in \tilde{I} \setminus (\chi(h') \cup \chi(h''))} L_i | \Delta' \in \Im(C'^{\mathsf{pre}}) \right\}.$$

8. SKIP. *As in Algorithm 6.4.12.*
9. FINALIZATION. *As in Algorithm 6.4.12.*

where

$$T_1 = \min_{i \in \{0,\ldots,2k-r-d'^{\perp}+1\}} \left(\max\{(2k-r-d'^{\perp}+2-i)(2\alpha-1), ((2\alpha-1)d'^{\perp} - \alpha) + i(2\alpha-1)\} \right)$$

and $T_2 = ((2\alpha-1)(2k-r+2) - \alpha) - T_1$*. The expected time-complexity of the algorithm is a small constant multiple of*

$$\max\left(q, 2^{(2\alpha-1)\lceil \frac{d'^{\perp}}{2} \rceil} n, |L_{h'}|, 2^{T_1} n, |L_{h''}|\right)$$

requiring expected memory around $\max\left(q, |\tilde{L}_{h'_1}|, |\tilde{L}_{h''_1}|, 2^{T_2} n\right)$ *(expressed in* en/b*-bit blocks).*

Code		Cardinalities related to our attack						Complexity	
$[r,k,d]_{2^e}$	$d'^{\perp}$	$\lvert L_i\rvert$	$\lvert\tilde{L}_{h'_0}\rvert$	$\lvert\tilde{L}_{h'_1}\rvert$	$\lvert L_{h'}\rvert$	$\max\left(\lvert\tilde{L}_{h''_0}\rvert,\lvert\tilde{L}_{h''_1}\rvert\right)$	$\lvert L_{h''}\rvert$	**Time**	**Memory**
$[8,5,3]_4$	2	$2^{3n/7}$	$2^{3n/7}$	$2^{3n/7}$	$2^{n/7}$	$2^{4n/7}$	$2^{2n/7}$	$2^{5n/7}$	$2^{5n/7}$
$[12,9,3]_4$	4	$2^{n/5}$	$2^{2n/5}$	$2^{2n/5}$	$2^{n/5}$	$2^{3n/5}$	$2^{2n/5}$	$2^{3n/5}$	$2^{3n/5}$
$[9,5,4]_4$	2	$2^{2n/3}$	$2^{2n/3}$	$2^{2n/3}$	$2^{n/2}$	×	×	$2^{5n/6}$	$2^{5n/6}$
$[16,12,4]_4$	7	$2^{n/5}$	$2^{4n/5}$	$2^{3n/5}$	$2^{4n/5}$	$2^{4n/5}$	$2^{4n/5}$	$2^{4n/5}$	$2^{3n/5}$

Table 6.7 – An overview of the list cardinalities and computational complexity of collision attacks on the Knudsen–Preneel compression functions based on *non*-MDS codes.

Choice of Code We note, as in our preimage attacks, the obvious caveat that our algorithms against the four non-MDS codes are based on the generator matrices (hence the corresponding shortened and dual code of the shortened code) given by Magma's BKLC (Best Known Linear Codes) routine. Hence, it is conceivable that different, non-equivalent codes perform differently under our attack.

6.4.7 Proof of Theorems 6.4.17 and 6.4.19

As in Section 6.3.4, we proceed step by step to prove our claims and concentrate on the expected values and we largely ignore (the effects of) factors that are polynomial in n (e.g., due to memory access). In the following, memory is measured in multiples of cn-bit blocks. Note that the steps INITIALIZATION and FINALIZATION require negligible time; so we ignore their analysis.

QUERY PHASE. The time-complexity of this step is simply $2^{\alpha n}$ PuRF evaluations for $\alpha = k/(3k-r)$.

LOCAL COLLISION DETECTION. In this step, we collect all colliding pairs in partial collision lists L_i for all $i \in I$. Note that collision search can be done for each active PuRF independently. In order to reduce the time requirements, we suggest to run this phase with the previous step in conjunction. For each fresh query-response, we simply determine its position among the sorted list of previous responses. This takes logarithmic time in n using dichotomy search. Then, the task is simply to check the neighboring states whether they collide or not, which can be performed in constant time. All in all, this step takes $2^{\alpha n}$ time and memory (again ignoring the effects of the constants and the factors that are logarithmic in n). We also note that due to the special form of our queries we cannot directly exploit the well-known memory efficient collision-finding techniques (such as distinguished points).

(FIRST) MERGE PHASE. The main computational part of this step is the generation of the lists $\tilde{L}_{h'_j}$ whose analysis is very similar to the one performed in Section 6.3.4. The time required for generating $\tilde{L}_{h'_0}$ and $\tilde{L}_{h'_1}$ is equal to their respective sizes, namely $|L_i|^{|\chi(h'_0)|}$ and $|L_i|^{|\chi(h'_1)|}$. Moreover, we have that (for $i \in I$)

$$|L_i| \approx q^2/2^n = 2^{(2\alpha-1)n}\,.$$

As, by construction, $|\chi(h'_0)| = \lceil d'^{\perp}/2\rceil$ and $|\chi(h'_1)| = \lfloor d'^{\perp}/2\rfloor$, the relevant cardinalities become $2^{(2\alpha-1)\lfloor d'^{\perp}/2\rfloor n}$ and $2^{(2\alpha-1)\lceil d'^{\perp}/2\rceil n}$, respectively. This is clearly dominated by the latter.

(FIRST) JOIN PHASE. This step constructs the set of Δ' candidates by finding collisions in respective entries of the merged lists constructed in the previous phase. We assume that a collision search among two lists is performed naively by storing the smaller list and scanning through all elements

in the other. As $|\tilde{L}_{h'_0}| \cdot |\tilde{L}_{h'_1}| \approx 2^{d'^{\perp}(2\alpha-1)n}$ and we are interested in collisions on αn bits, we expect to have $|L_{h'}| \approx 2^{((2\alpha-1)d'^{\perp}-\alpha)n}$ as claimed. Note that each colliding element can be forwarded directly to the next step eliminating the need for storing $L_{h'}$. Moreover, the collision search can be performed in conjunction with the previous step storing only $\tilde{L}_{h'_1}$ and checking (and processing) on-the-fly collisions when generating $\tilde{L}_{h'_0}$. This way the memory requirements are reduced to $|\tilde{L}_{h'_1}|$.

SECOND MERGE PHASE and SECOND JOIN PHASE (for non-MDS codes). We limit ourselves to the four codes suggested by Knudsen and Preneel. For the (chosen) codes (and their shortened and dual code of the shortened codes) from the Magma's BKLC routine we can always find (by inspection) the codewords h', h'' in the dual code of $\mathscr{C}'$ with the property that h' has minimal weight, $\{1,\ldots,k\} \subset \chi(h') \cup \chi(h'')$ and the number of $i \in \chi(h'')$ for which $i \notin \chi(h')$ equals $k-\theta-d'^{\perp}+2$ (where substituting $\theta = r-k$ gives $2k-r-d'^{\perp}+2$).

As a result, the relation defined by h'' (and thus the second phase) will involve $2k-r-d'^{\perp}+2$ 'fresh' lists L_i (those for which $i \notin \chi(h')$ and $i \in \chi(h'')$) with $|L_i| = 2^{(2\alpha-1)n}$, as well as $L_{h'}$, for which we know from above that

$$|L_{h'}| = 2^{((2\alpha-1)d'^{\perp}-\alpha)n}\,.$$

Hence, regardless of the way of MERGEing and JOINing, there are

$$(2^{(2\alpha-1)n})^{(2k-r-d'^{\perp}+2)} \cdot 2^{((2\alpha-1)d'^{\perp}-\alpha)n} = 2^{((2\alpha-1)(2k-r+2)-\alpha)n}$$

elements in total (to be checked for collisions). By construction, collisions are searched for on αn bits. This leads to a list $L_{h''}$ of roughly

$$|L_{h''}| = 2^{((2\alpha-1)(2k-r+2)-2\alpha)n}$$

elements at the end of SECOND JOIN PHASE. To minimize the complexity of the merging phase, we need to find the sets $\chi(h''_0)$ and $\chi(h''_1)$ such that $\chi(h''_0) \cap \chi(h''_1) = \emptyset$ and the full $2^{((2\alpha-1)(2k-r+2)-\alpha)n}$ elements (involved in the merging) are distributed as evenly as possible *without* violating the constraints imposed by the asymmetric list sizes.

Here, we follow the same reasoning as in Section 6.3.4. The condition $\chi(h''_0) \cap \chi(h''_1) = \emptyset$ implies that $L_{h'}$ is assigned to h''_0; assume that j further fresh L_i are used for h''_0. This automatically means that $|\chi(h''_1)| = 2k-r-d'^{\perp}+2-j$ and furthermore that

$$|\tilde{L}_{h''_1}| = 2^{(2k-r-d'^{\perp}+2-j)(2\alpha-1)n} \quad \text{and} \quad |\tilde{L}_{h''_0}| = 2^{j(2\alpha-1)n+((2\alpha-1)d'^{\perp}-\alpha)n}\,.$$

Given a particular j, the merging time will be governed by the maximum of $|\tilde{L}_{h''_1}|$ and $|\tilde{L}_{h''_0}|$, whereas the storage requirement is the minimum of that pair. In order to optimize the overall time-complexity, we take the minimum (of the maximum just mentioned) over all j and denote the value by T_1 and, for the value j used, denote by T_2 the corresponding 'memory'-minimum. We also note the trivial fact: $T_1 + T_2 = (2\alpha-1)(2k-r+2)-\alpha$. Collision-finding can then be performed in $2^{T_1 n}$ time with a memory requirement of roughly $2^{T_2 n}$.

COLLISION PRUNING. For MDS codes, there are $|L_{h'}| \approx 2^{((2\alpha-1)d'^{\perp}-\alpha)n}$ difference candidates Δ'. In this step, we choose the true codeword among those by checking memberships for the remaining partial collision lists. Because the shortened code $\mathscr{C}'$ has parameters $[r', k', d'] = [k, 2k-r, d']$ and

we have previously reviewed $d'^{\perp}$ lists, we still need to perform $r' - d'^{\perp}$ list membership-checks to do this step. For each membership-check, the probability of hitting one of the elements in each remaining partial collision list is $2^{-\alpha n}$ and we have $2^{(2\alpha-1)n}$ targets to hit. All in all, after $r' - d'^{\perp}$ list membership-checks the number of remaining candidates becomes (for $r' = r - \theta = k$)

$$2^{((2\alpha-1)d'^{\perp}-\alpha)n} \cdot \left(2^{(2\alpha-1)n} \cdot 2^{-\alpha n}\right)^{k-d'^{\perp}} = 2^{((2\alpha-1)d'^{\perp}-\alpha)+(\alpha-1)(k-d'^{\perp})}.$$

Substituting $\alpha = k/(3k-r)$ and $d'^{\perp} = 2k - r + 1$ we expect to end up with a single Δ'. Note that the time-complexity of this step is equal to the cardinality of $L_{h'}$.

Regarding non-MDS codes, for each element in $L_{h''}$ we need to perform a simple check (that we assume costs unit time and constant memory). We have already shown that

$$|L_{h''}| = 2^{((2\alpha-1)(2k-r+2)-2\alpha)n}$$

(at least for the four non-MDS codes provided by Knudsen and Preneel). So, this step takes at most $|L_{h''}|$ time. As the shortened code $\mathscr{C}'$ has length k and we have already gone over $2k-r+2$ lists, we still need to perform $k-(2k-r+2)$ list membership-checks to finalize this step. For each membership-check, the probability of hitting one of the elements in each remaining partial collision list is $2^{-\alpha n}$ and there are $2^{(2\alpha-1)n}$ targets to hit. All in all, after $r-k-2$ checks the number of remaining candidates becomes

$$2^{((2\alpha-1)(2k-r+2)-2\alpha)n} \cdot \left(2^{(2\alpha-1)n} \cdot 2^{-\alpha n}\right)^{r-k-2} = 2^{((2\alpha-1)(2k-r+2)-2\alpha)+(\alpha-1)(r-k-2)}.$$

Substituting $\alpha = k/(3k-r)$, we expect to end up with a single difference Δ'. If $d'^{\perp} > k'$, however, several membership-checks need to be performed, which can be done in $\max(|L_i|, |L_{h'}|)$ time and $\min(|L_i|, |L_{h'}|)$ memory. Note also that this happens only for one parameter: $[9,5,4]_{2^4}$. Substituting the relevant values indeed result in one collision.

Summing up the obtained complexities for the various steps gives us the desired overall complexity as stated in Theorems 6.4.17 and 6.4.19.

7 Conclusions

In this thesis, we focus on one of the most active areas of cryptographic hash function research: analysis and design of multi-call multi-block-length primitive-based compression and hash functions. We investigate certain type of designs that aim to solve the problem of improving the security of a single-block-length primitive-based compression function, without changing the underlying primitive. We take blockciphers and PuRFs (Public Random Functions) as the underlying primitives and consider the constructions that can make parallel calls to these primitives. From a theoretical point of view, most of the designs considered in this thesis are supported by formal security proofs in the ideal primitive model. From a practical point of view, we concern ourselves with the constructions that are relevant in practice; indeed, all the schemes we consider make parallel calls to the underlying primitives thus minimizing the computational overhead. Furthermore, they can be instantiated with conventional cryptographic primitives, e.g., blockciphers such as AES; hence for resource constrained environments, we only need to implement one blockcipher to obtain simultaneously an encryption scheme and a hash function.

A Quick Glance at the Contributions and Open Problems Regarding the design of provable secure compression and hash functions, we provide several positive results in Chapters 4 and 5; positive in the sense that we show the existence of certain double-call DBL blockcipher- and PuRF-based compression and hash function constructions with security guarantees more than a single-block-length compression function can offer.

Specifically, in Chapter 4, we limit ourselves to the $3n \to 2n$ double-call DBL blockcipher-based compression functions with $2n$-bit key blockciphers and show that close-to-optimal collision and preimage resistance can be achieved. More to the point, we present sufficient conditions for a large class of designs that achieve high-level security; our framework provides a comprehensive and unified approach that also captures the analysis of many existing schemes.

One major open problem that remains regarding this class of compression functions is performance related; although the security of these schemes is well-understood, there do not exist many works (except [33, 35]) in the literature concerning performance. In particular, we believe that the designs in question might result in lightweight hash function constructions when instantiated with relatively efficient primitives. The design of lightweight blockciphers is an active area of research; nevertheless, not many efforts are put in designing lightweight functions that would mimic PuRFs.

Along the same lines, in Chapter 5, we study the open question of constructing a provably collision-resistant compression function from $3n$ to $2n$ bits that makes two parallel calls to an ideal primitive from $2n$ to n bits (either a PuRF or an ideal blockcipher with n-bit blocks and n-bit keys). Our major contribution is the design of a concrete compression function with a collision resistance bound well beyond $2^{n/2}$ queries; in this class this is the first construction with such a collision resistance bound.

As a side contribution motivated by the problems while analyzing our design, we build novel technical tools for the hash function security proofs, developed in Chapter 3, that can be used to analyze complex systems; a specific example is our construction given in Chapter 5. We leave the exploration of the efficiency of our construction as an open problem. Moreover, finding a more efficient construction in the same class as ours with strong security guarantees is also open. Finally, we note that the point-line incidences used in the preprocessing function can be extended to incidences in higher dimensions (e.g., point, line and planes) and similar compression functions with higher number of calls and digest sizes (with better security guarantees) might be achieved. Nevertheless, we believe it is of only theoretical interest, as the number of finite field multiplications is expected to increase leading to a less efficient scheme.

From an analysis point of view, we also show some negative results (in Chapter 6) for the well-known Knudsen–Preneel family of compression functions; by negative we mean that the KP schemes turn out to be not as secure as they were designed to be. Specifically, by presenting several collision- and preimage-finding algorithms against the KP compression functions, we falsify the collision resistance claim and preimage resistance conjecture initially given by Knudsen and Preneel. For preimage resistance, we provide both lower and upper bounds for any type of KP scheme. Regarding collision resistance, we can only show upper bounds; we leave proving collision resistance of the KP compression functions in the ideal primitive model open. In addition, the security analysis of KP framework in the Merkle–Damgård iteration is also open.

Another Look at Hash Function Research The recent research efforts on cryptographic hash functions, as well as the results presented in Chapters 4, 5 and 6, shed some light on the future directions of hash function research. First of all, we remark that the assumptions used in our security proofs are still far from being practical as we work in the ideal primitive model; hence, weakening these assumptions or working with some other primitives might be one of the ideas to be pursued in the future. Additionally, we mainly consider information-theoretic adversaries in this thesis, the adversaries that are computationally unbounded. To better reflect reality, we should also work in the complexity-theoretic setting; we leave the exploration of this as an open problem.

Secondly, from a theoretical point of view, there still remain many parameter choices for designing a multi-call multi-block-length primitive-based compression function with strong security guarantees (e.g., collision resistance well beyond $2^{n/2}$ queries when primitives with output length of n bits are used) assuming the correctness of Stam's conjecture (Conjecture 2.3.5). However, as shown in Chapter 5, such designs might result in less efficient and less attractive schemes to be used in practice; and indeed many simple choices turn out to be insufficient to obtain a secure construction. The KP construction is such an example: One of the lessons that we learn from the results of Chapter 6 is that we should be careful with the highly efficient compression functions that use primitives with domains smaller than that of the compression function and that utilize simple postprocessing functions. In addition, although making the calls parallel is an attractive choice for efficiency, it might

result in less secure schemes.

Thirdly, we note that focusing on the hash function, rather than imposing high-level security in the compression function (as done in [20]), is another research direction. Indeed, certain SHA-3 candidates (e.g., [19]), as well as new lightweight hash function constructions (e.g., [32, 71]) follow this idea. Unfortunately, current proof techniques for analyzing security in the domain extension are not as powerful as the ones for compression functions (without making strong assumptions on the compression function). Consequently, for the resulting hash functions that do not rely on the Merkle–Damgård paradigm (Theorem 2.2.5), the proofs are very complex. Finding more effective methods for analyzing the security in the iteration, as well as designing practical compression functions that provide improved security for the hash function, are two of the promising research ideas not only from a theoretical but also from a practical point of view. We believe, especially for hardware constrained environments, that this design choice is an efficient solution to achieve a lightweight hash function construction.

Finally, we note that the analysis of the currently available schemes from an efficiency point of view is another attractive research direction. A hardware benchmark of several multi-call multi-block-length blockcipher-based compression functions (using the lightweight blockcipher Present) is carried out in [33]; similarly, a software evaluation using AES and the AES instruction set is done in [35]. However, these works are still limited in the sense that they use certain primitives on specific platforms. Extending these benchmarks to a more general setting is still open. More importantly, designing concrete primitives (that themselves are not sufficient to provide a high-level security when used in single-block-length mode) to be used along with the multi-call multi-block-length primitive-based compression functions would be important. In addition, as noted in [32, 71], the currently available SHA-3 candidates (as well as SHA-2) are not very hardware-friendly: They require a footprint which is more than an RFID tag or a wireless sensor can afford for security applications. We believe that in the near future the research community will also be busy designing more lightweight hash function constructions.

Bibliography

[1] Deschall project: World's first des crack, http://home.earthlink.net/~rcv007/deschall.htm

[2] The eSTREAM project, http://www.ecrypt.eu.org/stream/

[3] Andreeva, E., Mennink, B., Preneel, B.: Security properties of domain extenders for cryptographic hash functions. JIPS 6(4), 453–480 (2010)

[4] Andreeva, E., Neven, G., Preneel, B., Shrimpton, T.: Three-property preserving iterations of keyless compression functions, presented at ECRYPT Hash Workshop, 2007

[5] Andreeva, E., Neven, G., Preneel, B., Shrimpton, T.: Seven-property-preserving iterated hashing: ROX. In: Kurosawa [99], pp. 130–146

[6] Andreeva, E., Preneel, B.: A three-property-secure hash function. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) Selected Areas in Cryptography. LNCS, vol. 5381, pp. 228–244. Springer, Heidelberg (2009)

[7] Andreeva, E., Stam, M.: The symbiosis between collision and preimage resistance. In: Chen, L. (ed.) IMA Int. Conf. LNCS, vol. 7089, pp. 152–171. Springer, Heidelberg (2011)

[8] Armknecht, F., Fleischmann, E., Krause, M., Lee, J., Stam, M., Steinberger, J.P.: The preimage security of double-block-length compression functions. In: Lee and Wang [103], pp. 233–251

[9] Aumasson, J.P., Çağdaş Çalık, Meier, W., Özen, O., Phan, R.C.W., Varıcı, K.: Improved cryptanalysis of Skein. In: Matsui [116], pp. 542–559

[10] Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 proposal BLAKE. Submission to NIST (Round 3) (2010), http://131002.net/blake/blake.pdf

[11] Bellare, M., Boldyreva, A., Palacio, A.: An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In: Cachin and Camenisch [40], pp. 171–188

[12] Bellare, M., Ristenpart, T.: Multi-property-preserving hash domain extension and the EMD transform. In: Lai and Chen [101], pp. 299–314

[13] Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proc. 1st ACM Conference on Computer and Communications Security. pp. 62–73 (1993)

[14] Bellare, M., Kohno, T.: Hash function balance and its impact on birthday attacks. In: Cachin and Camenisch [40], pp. 401–418

[15] Bellare, M., Rogaway, P.: Collision-resistant hashing: Towards making UOWHFs practical. In: Burt Kaliski and Burton [39], pp. 470–484

[16] Bernstein, D.J.: Cache-timing attacks on AES (2004), available at http://cr.yp.to/antiforgery/cachetiming-20050414.pdf

[17] Bernstein, D.J.: Cubehash specification (2.b.1). Submission to NIST (Round 2) (2009), http://cubehash.cr.yp.to/submission2/spec.pdf

[18] Bernstein, D.J., Lange, T.: eBACS: ECRYPT Benchmarking of Cryptographic Systems. http://bench.cr.yp.to (2010)

[19] Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak SHA-3 submission. Submission to NIST (Round 3) (2011), http://keccak.noekeon.org/Keccak-submission-3.pdf

[20] Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the indifferentiability of the sponge construction. In: Smart [189], pp. 181–197

[21] Biham, E., Dunkelman, O.: A framework for iterative hash functions - HAIFA. Cryptology ePrint Archive, Report 2007/278 (2007), http://eprint.iacr.org/

[22] Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A., Vanstone, S. (eds.) Advances in Cryptography—Crypto'90. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991)

[23] Biham, E., Shamir, A.: Differential cryptanalysis of the full 16-round DES. In: Brickell, E. (ed.) Advances in Cryptography—Crypto'92. LNCS, vol. 740. Springer, Heidelberg (1993)

[24] Biryukov, A., Khovratovich, D.: Related-key cryptanalysis of the full AES-192 and AES-256. In: Matsui [116], pp. 1–18

[25] Black, J., Cochran, M., Shrimpton, T.: On the impossibility of highly efficient blockcipher-based hash functions. In: Cramer [50], pp. 526–541

[26] Black, J., Rogaway, P., Shrimpton, T.: Black-box analysis of the block-cipher-based hash-function constructions from PGV. In: Yung [207], pp. 320–335

[27] Black, J., Rogaway, P., Shrimpton, T., Stam, M.: An analysis of the block-cipher-based hash functions from PGV. Journal of Cryptology 23(4), 519–545 (2010)

[28] Black, J.: The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In: Robshaw [170], pp. 328–340

[29] Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudo random bits. In: FOCS. pp. 112–117. IEEE Computer Society (1982)

[30] den Boer, B., Bosselaers, A.: Collisions for the compression function of MD5. In: Helleseth [74], pp. 293–304

[31] Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In: Lee and Wang [103]

[32] Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: Spongent: A lightweight hash function. In: Preneel and Takagi [161], pp. 312–325

[33] Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash functions and RFID tags: Mind the gap. In: Oswald, E., Rohatgi, P. (eds.) CHES'08. LNCS, vol. 5154, pp. 283–299. Springer, Heidelberg (2008)

[34] Bos, J.W., Özen, O., Hubaux, J.P.: Analysis and optimization of cryptographically generated addresses. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC. LNCS, vol. 5735, pp. 17–32. Springer, Heidelberg (2009)

[35] Bos, J.W., Özen, O., Stam, M.: Efficient hashing using the AES instruction set. In: Preneel and Takagi [161], pp. 507–522

[36] Boyd, C., Nieto, J.M.G. (eds.): Information Security and Privacy (ACISP'09), LNCS, vol. 5594. Springer, Heidelberg (2009)

[37] Brachtl, B., Coppersmith, D., Hyden, M., Matyas, S., Jr., Meyer, C., Oseas, J., Pilpel, S., Schilling, M.: Data authentication using modification detection codes based on a public one-way encryption function. U.S. Patent No 4,908,861 (March 1990)

[38] Brassard, G. (ed.): Advances in Cryptography—Crypto'89, LNCS, vol. 435. Springer, Heidelberg (1990)

[39] Burt Kaliski, J., Burton, S. (eds.): Advances in Cryptography—Crypto'97, LNCS, vol. 1294. Springer, Heidelberg (1997)

[40] Cachin, C., Camenisch, J. (eds.): Advances in Cryptography—Eurocrypt'04, LNCS, vol. 3027. Springer, Heidelberg (2004)

[41] Camion, P., Patarin, J.: The knapsack hash function proposed at Crypto'89 can be broken. In: Davies, D.W. (ed.) Advances in Cryptography—Eurocrypt'91. LNCS, vol. 547, pp. 39–53. Springer, Heidelberg (1991)

[42] Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. The Computing Research Repository (CoRR) cs.CR/0010019 (2000)

[43] Canetti, R., Goldreich, O., Halevi, S.: On the random-oracle methodology as applied to length-restricted signature schemes. In: Naor [131], pp. 40–57

[44] Cannière, C.D., Rechberger, C.: Finding SHA-1 characteristics: General results and applications. In: Lai and Chen [101]

[45] Canniere, C.D., Sato, H., Watanabe, D.: Hash function Luffa: Specification. Submission to NIST (Round 2) (2009), http://www.sdl.hitachi.co.jp/crypto/luffa/Luffa_v2_Specification_20091002.pdf

[46] Chabaud, F., Joux, A.: Differential collisions in SHA-0. In: Krawczyk, H. (ed.) Advances in Cryptography—Crypto'98. LNCS, vol. 1462, pp. 56–71. Springer, Heidelberg (1998)

[47] Chose, P., Joux, A., Mitton, M.: Fast correlation attacks: An algorithmic point of view. In: Knudsen [89], pp. 209–221

[48] Churchill, W.: "It was thanks to ultra that we won the war", http://www.history.co.uk/explore-history/ww2/code-breaking.html

[49] Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: How to construct a hash function. In: Shoup [185], pp. 430–448

[50] Cramer, R. (ed.): Advances in Cryptography—Eurocrypt'05, LNCS, vol. 3494. Springer, Heidelberg (2005)

[51] Daemen, J., Rijmen, V.: The Design of Rijndael. Springer-Verlag New York, Inc. Secaucus, NJ, USA (2002)

[52] Damgård, I.: A design principle for hash functions. In: Brassard [38], pp. 416–427

[53] Dean, R.D.: Formal Aspects of Mobile Code Security. Ph.D. thesis, Princeton University (1999)

[54] Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (1976)

[55] Dobbertin, H.: Cryptanalysis of MD4. Journal of Cryptology 11(4), 253–271 (1998)

[56] Dodis, Y., Ristenpart, T., Shrimpton, T.: Salvaging Merkle-Damgård for practical applications. In: Joux [83], pp. 371–388

[57] Dunkelman, O. (ed.): Fast Software Encryption (FSE'09), LNCS, vol. 5665. Springer, Heidelberg (2009)

[58] Duo, L., Li, C.: Improved collision and preimage resistance bounds on PGV schemes. Tech. Rep. 462, IACR's ePrint Archive (2006)

[59] ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (1985)

[60] Feistel, H.: Block cipher cryptographic system. U.S. Patent No 3,798,359 (Filed June 30, 1971 (IBM))

[61] Feller, W.: An Introduction to Probability Theory and its Applications, vol. 1. John Wiley and Sons, Inc. (1968)

[62] Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein hash function family. Submission to NIST (Round 3) (2010), http://www.skein-hash.info/sites/default/files/skein1.3.pdf

[63] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A. (ed.) Advances in Cryptography—Crypto'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)

[64] Fleischmann, E., Gorski, M., Lucks, S.: Security of cyclic double block length hash functions. In: Parker [153], pp. 153–175

[65] Fleischmann, E., Gorski, M., Lucks, S.: Security of cyclic double block length hash functions including Abreast-DM. Tech. Rep. 261, IACR's ePrint Archive (2009)

[66] Franklin, M.K. (ed.): Advances in Cryptography—Crypto'04, LNCS, vol. 3152. Springer, Heidelberg (2004)

[67] Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – A SHA-3 candidate. Submission to NIST (Round 3) (2011), http://www.groestl.info/Groestl.pdf

[68] Gazi, P., Maurer, U.: Free-start distinguishing: Combining two types of indistinguishability amplification. In: Kurosawa, K. (ed.) ICITS. LNCS, vol. 5973, pp. 28–44. Springer, Heidelberg (2010)

[69] Gilbert, H. (ed.): Advances in Cryptography—Eurocrypt'10, LNCS, vol. 6110. Springer, Heidelberg (2010)

[70] Gueron, S.: Intel's new AES instructions for enhanced performance and security. In: Dunkelman [57]

[71] Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: Rogaway [176], pp. 222–239

[72] Halevi, S., Krawczyk, H.: Strengthening digital signatures via randomized hashing. In: Dwork, C. (ed.) Advances in Cryptography—Crypto'06. LNCS, vol. 4117, pp. 41–59. Springer, Heidelberg (2006)

[73] Hattori, M., Hirose, S., Yoshida, S.: Analysis of double block length hash functions. In: Paterson, K. (ed.) CCC'03. LNCS, vol. 2898, pp. 290–302. Springer, Heidelberg (2003)

[74] Helleseth, T. (ed.): Advances in Cryptography—Eurocrypt'93, LNCS, vol. 765. Springer, Heidelberg (1993)

[75] Hirose, S.: Provably secure double-block-length hash functions in a black-box model. In: Park, C., Chee, S. (eds.) ICISC'04. LNCS, vol. 3506, pp. 330–342. Springer, Heidelberg (2005)

[76] Hirose, S.: Some plausible constructions of double-length hash functions. In: Robshaw [170], pp. 210–225

[77] Hirose, S., Park, J.H., Yun, A.: A simple variant of the Merkle-Damgård scheme with a permutation. In: Kurosawa [99], pp. 113–129

[78] Hohl, W., Lai, X., Meier, T., Waldvogel, C.: Security of iterated hash functions based on block ciphers. In: Stinson [196], pp. 379–390

[79] Hong, S., Iwata, T. (eds.): Fast Software Encryption (FSE'10), LNCS, vol. 6147. Springer, Heidelberg (2010)

[80] Indesteege, S.: The LANE hash function. Submission to NIST (2008), http://www.cosic.esat.kuleuven.be/publications/article-1181.pdf

[81] Jetchev, D., Özen, O., Stam, M.: Collisions Are Not Incidental: A Compression Function Exploiting Discrete Geometry. In: Cramer, R. (ed.) TCC. LNCS, vol. 7194, pp. 303–320. Springer, Heidelberg (2012)

[82] Joux, A.: Multicollisions in iterated hash functions. Application to cascaded constructions. In: Franklin [66], pp. 306–316

[83] Joux, A. (ed.): Advances in Cryptography—Eurocrypt'09, LNCS, vol. 5479. Springer, Heidelberg (2009)

[84] Joux, A., Peyrin, T.: Hash functions and the (amplified) boomerang attack. In: Franklin [66], pp. 244–263

[85] Kahn, D.: The Codebreakers. Scribner, New York (1996)

[86] Kaliski, B.: The MD4 message-digest algorithm, request for comments (RFC) 1320. Tech. rep., Internet Activities Board, Internet Privacy Task Force (1992)

[87] Kelsey, J., Kohno, T.: Herding hash functions and the nostradamus attack. In: Vaudenay, S. (ed.) Advances in Cryptography—Eurocrypt'06. LNCS, vol. 4004, pp. 183–200. Springer, Heidelberg (2006)

[88] Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2^n work. In: Cramer [50], pp. 474–490

[89] Knudsen, L. (ed.): Advances in Cryptography—Eurocrypt'02, LNCS, vol. 2332. Springer, Heidelberg (2002)

[90] Knudsen, L.R., Mendel, F., Rechberger, C., Thomsen, S.S.: Cryptanalysis of MDC-2. In: Joux [83], pp. 106–120

[91] Knudsen, L.R., Preneel, B.: Hash functions based on block ciphers and quaternary codes. In: Kim, K., Matsumoto, T. (eds.) Advances in Cryptography—Asiacrypt'96. LNCS, vol. 1163, pp. 77–90. Springer, Heidelberg (1996)

[92] Knudsen, L.R., Preneel, B.: Fast and secure hashing based on codes. In: Burt Kaliski and Burton [39], pp. 485–498

[93] Knudsen, L.R., Preneel, B.: Construction of secure and fast hash functions using nonbinary error-correcting codes. IEEE Transactions on Information Theory 48(9), 2524–2539 (2002)

[94] Knudsen, L., Lai, X., Preneel, B.: Attacks on fast double block length hash functions. Journal of Cryptology 11(1), 59–72 (1998)

[95] Knuth, D.E.: Fundamental Algorithms, The Art of Computer Programming, vol. 1. Addison Wesley, 3 edn. (1997)

[96] Knuth, D.E.: Seminumerical Algorithms, The Art of Computer Programming, vol. 2. Addison Wesley, 3 edn. (1997)

[97] Knuth, D.E.: Sorting and Searching, The Art of Computer Programming, vol. 3. Addison Wesley, 2 edn. (1998)

[98] Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation 48, 203–209 (1987)

[99] Kurosawa, K. (ed.): Advances in Cryptography—Asiacrypt'07, LNCS, vol. 4833. Springer, Heidelberg (2007)

[100] Özgül Küçük: The hash function Hamsi. Submission to NIST (updated) (2009), http://www.cosic.esat.kuleuven.be/publications/article-1203.pdf

[101] Lai, X., Chen, K. (eds.): Advances in Cryptography—Asiacrypt'06, LNCS, vol. 4284. Springer, Heidelberg (2006)

[102] Lai, X., Massey, J.L.: Hash function based on block ciphers. In: Rueppel, R.A. (ed.) Advances in Cryptography—Eurocrypt'92. LNCS, vol. 658, pp. 55–70. Springer, Heidelberg (1992)

[103] Lee, D.H., Wang, X. (eds.): Advances in Cryptography—Asiacrypt'11, LNCS, vol. 7073. Springer, Heidelberg (2011)

[104] Lee, J., Kwon, D.: The security of Abreast-DM in the ideal cipher model. Tech. Rep. 225, IACR's ePrint Archive (2009)

[105] Lee, J., Stam, M.: MJH: A faster alternative to MDC-2. In: Kiayias, A. (ed.) CT-RSA. LNCS, vol. 6558, pp. 213–236. Springer, Heidelberg (2011)

[106] Lee, J., Stam, M., Steinberger, J.: The collision security of Tandem-DM in the ideal cipher model. Tech. Rep. 409, IACR's ePrint Archive (2010), http://eprint.iacr.org/

[107] Lee, J., Stam, M., Steinberger, J.P.: The collision security of tandem-dm in the ideal cipher model. In: Rogaway [176], pp. 561–577

[108] Lee, J., Steinberger, J.P.: Multi-property-preserving domain extension using polynomial-based modes of operation. In: Gilbert [69], pp. 573–596

[109] Lenstra, A.K., Lenstra, H.W.: The Development of Number Field Sieve. Springer-Verlag Berlin Heidelberg (1993)

[110] Liskov, M.: Constructing an ideal hash function from weak ideal compression functions. In: Biham, E., Youssef, A.M. (eds.) Selected Areas in Cryptography. LNCS, vol. 4356, pp. 358–375. Springer, Heidelberg (2007)

[111] Lucks, S.: A failure-friendly design principle for hash functions. In: Roy, B.K. (ed.) Advances in Cryptography—Asiacrypt'05. LNCS, vol. 3788, pp. 474–494. Springer, Heidelberg (2005)

[112] Lucks, S.: A collision-resistant rate-1 double-block-length hash function. In: Biham, E., Handschuh, H., Lucks, S., Rijmen, V. (eds.) Symmetric Cryptography. No. 07021 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany (2007), http://drops.dagstuhl.de/opus/volltexte/2007/1017

[113] Manuel, S., Peyrin, T.: Collisions on SHA-0 in one hour. In: Nyberg, K. (ed.) FSE'08. LNCS, vol. 5086, pp. 16–35. Springer, Heidelberg (2008)

[114] Matsui, M.: Linear cryptoanalysis method for DES cipher. In: Helleseth [74], pp. 386–397

[115] Matsui, M.: The first experimental cryptanalysis of the Data Encryption Standard. In: Desmedt, Y. (ed.) Advances in Cryptography—Crypto'94. LNCS, vol. 839, pp. 1–11. Springer, Heidelberg (1994)

[116] Matsui, M. (ed.): Advances in Cryptography—Asiacrypt'09, LNCS, vol. 5912. Springer, Heidelberg (2009)

[117] Maurer, U., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Naor [131], pp. 21–39

[118] Maurer, U., Tessaro, S.: Domain extension of public random functions: Beyond the birthday barrier. In: Menezes [123], pp. 187–204

[119] Maurer, U.M.: Indistinguishability of random systems. In: Knudsen [89], pp. 110–132

[120] Maurer, U.M., Pietrzak, K.: Composition of random systems: When two weak make one strong. In: Naor [131], pp. 410–427

[121] Maurer, U.M., Pietrzak, K., Renner, R.: Indistinguishability amplification. In: Menezes [123], pp. 130–149

[122] Menezes, A., van Oorschot, P., Vanstone, S.: CRC-Handbook of Applied Cryptography. CRC Press (1996)

[123] Menezes, A. (ed.): Advances in Cryptography—Crypto'07, LNCS, vol. 4622. Springer, Heidelberg (2007)

[124] Merkle, R.C.: Secure communications over insecure channels. Communications of the ACM 21(4), 294–299 (1978)

[125] Merkle, R.C.: One way hash functions and DES. In: Brassard [38], pp. 428–446

[126] Miller, V.: Use of elliptic curves in cryptography. In: Williams, H. (ed.) Advances in Cryptography—Crypto'85. LNCS, vol. 218, pp. 417–425. Springer, Heidelberg (1986)

[127] von Mises, R.: İstanbul Üniversitesi fen fakültesi mecmuası 4, 145–163 (1939)

[128] Nandi, M.: Towards optimal double-length hash functions. In: Maitra, S., Madhavan, C.E.V., Venkatesan, R. (eds.) INDOCRYPT'05. LNCS, vol. 3797, pp. 77–89. Springer, Heidelberg (2005)

[129] Nandi, M., Lee, W., Sakurai, K., Lee, S.: Security analysis of a 2/3-rate double length compression function in black-box model. In: Gilbert, H., Handschuh, H. (eds.) FSE'05. LNCS, vol. 3557, pp. 243–254. Springer, Heidelberg (2005)

[130] Nandi, M.: Characterizing padding rules of md hash functions preserving collision security. In: Boyd and Nieto [36]

[131] Naor, M. (ed.): Theory of Cryptography Conference, LNCS, vol. 2951. Springer, Heidelberg (2004)

[132] Naor, M. (ed.): Advances in Cryptography—Eurocrypt'07, LNCS, vol. 4515. Springer, Heidelberg (2007)

[133] Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In: Johnson, D.S. (ed.) STOC. pp. 33–43. ACM (1989)

[134] National Institute of Standards and Technology: Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher (1993), NIST Special Publication 800-67 Version 1.1

[135] National Institute of Standards and Technology: Secure Hash Standard (1993), FIPS-180

[136] National Institute of Standards and Technology: Secure Hash Standard (1995), FIPS-180-1

[137] National Institute of Standards and Technology: Announcing Development of a Federal Information Processing Standard for Advanced Encryption Standard (1997), http://csrc.nist.gov/archive/aes/pre-round1/aes_9701.txt

[138] National Institute of Standards and Technology: Data Encryption Standard (DES) (1999), FIPS-46-3

[139] National Institute of Standards and Technology: Advanced Encryption Standard (2001), FIPS 197

[140] National Institute of Standards and Technology: The Keyed-Hash Message Authentication Code (HMAC) (2002), FIPS 198

[141] National Institute of Standards and Technology: Secure Hash Standard (2002), FIPS-180-3

[142] National Institute of Standards and Technology: Digital Signature Standard (2009), FIPS 186-3

[143] Needham, R.: "in 1966, we conceived the use of one-way functions to protect the password file, and this was an implemented feature from day one". The Internet Encyclopedia, Volume 3, by Hossein Bidgoli (2004)

[144] Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: Yung [207], pp. 111–126

[145] NIST: Cryptographic hash algorithm competition. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html (2008)

[146] Nivasch, G.: Cycle detection using a stack. Inf. Process. Lett. 90(3), 135–140 (2004)

[147] van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. Journal of Cryptology 12(1), 1–28 (1999)

[148] Osvik, D.A.: Efficient implementation of the Data Encryption Standard. Master's thesis, University of Bergen, Norway (2003)

[149] Özen, O., Shrimpton, T., Stam, M.: Attacking the Knudsen–Preneel compression functions. In: Hong and Iwata [79], pp. 94–115

[150] Özen, O., Stam, M.: Another glance at double-length hashing. In: Parker [153], pp. 176–201

[151] Özen, O., Stam, M.: Collision attacks against Knudsen–Preneel compression functions. In: Abe, M. (ed.) Advances in Cryptography—Asiacrypt'10. LNCS, vol. 6477, pp. 76–93. Springer, Heidelberg (2010)

[152] Özen, O., Varıcı, K., Tezcan, C., Çelebi Kocair: Lightweight block ciphers revisited: Cryptanalysis of reduced round PRESENT and HIGHT. In: Boyd and Nieto [36], pp. 90–107

[153] Parker, M.G. (ed.): Cryptography and Coding 2009, LNCS, vol. 5921. Springer, Heidelberg (2009)

[154] Peyrin, T., Gilbert, H., Muller, F., Robshaw, M.: Combining compression functions and block cipher-based hash functions. In: Lai and Chen [101], pp. 315–331

[155] Pietrzak, K.: Indistringuishability and Composition of Random Systems. Ph.D. thesis, ETH Zurich (2005)

[156] Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U. (ed.) Advances in Cryptography—Eurocrypt'96. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996)

[157] Pollard, J.: Monte Carlo methods for factorization 15, 331–334 (1975)

[158] Pollard, J.: Monte Carlo methods for index computation (mod p). Mathematics of Computation 32(143), 918–924 (1978)

[159] Preneel, B.: Analysis and design of cryptographic hash functions. Ph.D. thesis, Katholieke Universiteit Leuven (1993)

[160] Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: A synthetic approach. In: Stinson [196], pp. 368–378

[161] Preneel, B., Takagi, T. (eds.): Cryptographic Hardware and Embedded Systems (CHES'11), LNCS, vol. 6917. Springer, Heidelberg (2011)

[162] Rabin, M.: Digitalized signatures. Foundations of Secure Computation (Academic Press, New York) (1978)

[163] Ramanujan, S.: On question 294. J. Indian Math. Soc. 4, 151–152 (1912)

[164] Reyhanitabar, M.R., Susilo, W., Mu, Y.: Enhanced security notions for dedicated-key hash functions: Definitions and relationships. In: Hong and Iwata [79], pp. 192–211

[165] Rivest, R.: The MD2 message-digest algorithm, request for comments (RFC) 1319. Tech. rep., Internet Activities Board, Internet Privacy Task Force (1992)

[166] Rivest, R.: The MD5 message-digest algorithm, request for comments (RFC) 1320. Tech. rep., Internet Activities Board, Internet Privacy Task Force (1992)

[167] Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Comm. of the ACM 21(2), 120–126 (1978)

[168] Rivest, R.: Abelian square-free dithering for iterated hash functions (2005), presented at ECRYPT Workshop on Hash Functions, June 23-24 2005, Krakow, Poland

[169] Rivest, R.L.: The MD6 hash function – a proposal to nist for SHA-3. Submission to NIST (2008), http://groups.csail.mit.edu/cis/md6/submitted-2008-10-27/Supporting_Documentation/md6_report.pdf

[170] Robshaw, M.J. (ed.): Fast Software Encryption (FSE'06), LNCS, vol. 4047. Springer, Heidelberg (2006)

[171] Rogaway, P., Shrimpton, T.: Cryptographic hash-function basics: Definitions, implications and separations for preimage resistance, second-preimage resistance, and collision resistance. In: Roy, B.K., Meier, W. (eds.) FSE. LNCS, vol. 3017, pp. 371–388. Springer, Heidelberg (2004)

[172] Rogaway, P., Steinberger, J.: Security/efficiency tradeoffs for permutation-based hashing, full version of [174], available through authors' website.

[173] Rogaway, P., Steinberger, J.: Constructing cryptographic hash functions from fixed-key blockciphers. In: Wagner [201], pp. 433–450

[174] Rogaway, P., Steinberger, J.: Security/efficiency tradeoffs for permutation-based hashing. In: Smart [189], pp. 220–236

[175] Rogaway, P.: Formalizing human ignorance. In: Nguyen, P.Q. (ed.) VIETCRYPT'06. LNCS, vol. 4341, pp. 211–228. Springer, Heidelberg (2006)

[176] Rogaway, P. (ed.): Advances in Cryptography—Crypto'10, LNCS, vol. 6841. Springer, Heidelberg (2011)

[177] S. Matyas, C. Meyer, J.O.: Generating strong one-way functions with cryptographic algorithms. IBM Tech. Dis. Bull. 27 (10a) (1985)

[178] Satoh, T., Haga, M., Kurosawa, K.: Towards secure and fast hash functions. IEICE Transactions, Special Section on Cryptography and Information Security E82–A(1) (1999)

[179] Schroeppel, R., Shamir, A.: A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. SIAM Journal on Computing 10, 456–464 (1981)

[180] Seurin, Y., Peyrin, T.: Security analysis of constructions combining FIL random oracles. In: Biryukov, A. (ed.) FSE'07. LNCS, vol. 4593, pp. 119–136. Springer, Heidelberg (2007)

[181] Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal 27, 379–423; 623–656 (1948), also appears in [183].

[182] Shannon, C.E.: Communication theory of secrecy systems. Bell System Technical Journal 28, 656–715 (1949), also appears in [183]. The material originally appeared in a confidential report 'A Mathematical Theory of Cryptography', dated Sept. 1, 1945.

[183] Shannon, C.E.: Claude Elwood Shannon. IEEE Press, New York (1993), collected papers, Edited by N. J. A. Sloane and Aaron D. Wyner

[184] Shoup, V.: A composition theorem for universal one-way hash functions. In: Preneel, B. (ed.) Advances in Cryptography—Eurocrypt'00. LNCS, vol. 1807, pp. 445–452. Springer, Heidelberg (2000)

[185] Shoup, V. (ed.): Advances in Cryptography—Crypto'05, LNCS, vol. 3621. Springer, Heidelberg (2005)

[186] Shrimpton, T., Stam, M.: Building a collision-resistant compression function from non-compressing primitives. In: ICALP 2008, Part II. vol. 5126, pp. 643–654. Springer, Heidelberg (2008)

[187] Simon, D.R.: Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In: Nyberg, K. (ed.) Advances in Cryptography—Eurocrypt'98. LNCS, vol. 1403, pp. 334–345. Springer, Heidelberg (1998)

[188] Singh, S.: The Code Book. Fourth Estate, New York (1999)

[189] Smart, N.P. (ed.): Advances in Cryptography—Eurocrypt'08, LNCS, vol. 4965. Springer, Heidelberg (2008)

[190] Stam, M.: Beyond uniformity: Better security/efficiency tradeoffs for compression functions. In: Wagner [201], pp. 397–412

[191] Stam, M.: Blockcipher-based hashing revisited. In: Dunkelman [57], pp. 67–83

[192] Steinberger, J.: The collision intractability of MDC-2 in the ideal-cipher model. In: Naor [132], pp. 34–51

[193] Steinberger, J.P.: Stam's collision resistance conjecture. In: Gilbert [69], pp. 597–615

[194] Stevens, M., Lenstra, A.K., de Weger, B.: Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In: Naor [132], pp. 1–22

[195] Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A.K., Molnar, D., Osvik, D.A., de Weger, B.: Short chosen-prefix collisions for MD5 and the creation of a rogue ca certificate. In: Halevi, S. (ed.) Advances in Cryptography—Crypto'09. LNCS, vol. 5677, pp. 55–69. Springer, Heidelberg (2009)

[196] Stinson, D. (ed.): Advances in Cryptography—Crypto'93, LNCS, vol. 773. Springer, Heidelberg (1993)

[197] Szemerédi, E., Trotter, W.: Extremal problems in discrete geometry. Combinatorica 3(3)

[198] Tao, T.: The Szemerédi-Trotter theorem and the cell decomposition (2009), http://terrytao.wordpress.com/2009/06/12/the-szemeredi-trotter-theorem-and-the-cell-decomposition/

[199] Tromer, E., Osvik, D.A., Shamir, A.: Efficient cache attacks on AES, and countermeasures. Journal of Cryptology 23(1), 37–71 (2010)

[200] Wagner, D.: A generalized birthday problem. In: Yung [207], pp. 288–303

[201] Wagner, D. (ed.): Advances in Cryptography—Crypto'08, LNCS, vol. 5157. Springer, Heidelberg (2008)

[202] Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup [185], pp. 17–36

[203] Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer [50], pp. 19–35

[204] Wang, X., Yu, H., Yin, Y.L.: Efficient collision search attacks on SHA-0. In: Shoup [185], pp. 1–16

[205] Watanabe, D.: A note on the security proof of Knudsen-Preneel construction of a hash function (2006), unpublished manuscript, available at http://csrc.nist.gov/groups/ST/hash/documents/WATANABE_kp_attack.pdf

[206] Wu, H.: The hash function JH. Submission to NIST (round 3) (2011), http://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf

[207] Yung, M. (ed.): Advances in Cryptography—Crypto'02, LNCS, vol. 2442. Springer, Heidelberg (2002)

[208] Yuval, G.: How to swindle Rabin. Cryptologia 3, 187–189 (1979)

A The Birthday Paradox

The birthday paradox states that, if 23 or more people are present in a room, there is a high probability that two of them share the same birthday. The origin of birthday paradox is obscure as also noted by Knuth; yet there exist several early references listed in his book [97] pointing out some informal discussions of it among mathematicians (see e.g., [61,127]). Here we illustrate a more general problem for which the birthday paradox is a particular example. Suppose that there exists an urn of N balls, each of which is colored in one of N distinct colors. We draw a random ball from the urn, record its color and put it back into the urn; this process is repeated several times. By drawing randomly, we mean that each ball is equally likely to be picked, and the probabilities (for all the balls to be picked) are independent. We determine the expected number of draws needed to record the same color twice. Note that, for $N = 365$, we derive the well-known birthday problem.

Let X be the random variable denoting the minimal number of draws needed for a collision to occur. Then we have

$$\Pr[X \geq k] = 1 \times \left(1 - \frac{1}{N}\right) \times \ldots \times \left(1 - \frac{1}{N-k+1}\right) = \frac{N!}{(N-k+1)!N^{k-1}} .$$

Now note that the expected value $\mathbb{E}(X)$ of X can be written as

$$\mathbb{E}(X) = \sum_{k=1}^{\infty} \Pr[X \geq k] = \sum_{k=1}^{\infty} \frac{N!}{(N-k+1)!N^{k-1}} = \sum_{k=1}^{N+1} \frac{N!}{(N-k+1)!N^{k-1}} .$$

Changing the variable $t = k-1$, we obtain

$$\mathbb{E}(X) = \sum_{t=0}^{N} \frac{N!}{(N-t)!N^t} = 1 + \sum_{t=1}^{N} \frac{N!}{(N-t)!N^t} = 1 + Q(N) \approx 1 + \sqrt{\frac{\pi N}{2}} - \frac{1}{3} + \frac{1}{12}\sqrt{\frac{\pi}{2N}} - \frac{4}{135N} + \ldots .$$

The approximation for $Q(N)$ is due to Ramanujan [95, 163]. Hence, asymptotically in N, we obtain

$$\mathbb{E}(X) = \sqrt{\frac{\pi N}{2}} + O\left(\frac{1}{\sqrt{N}}\right) .$$

For $N = 365$ we obtain slightly more than what we initially claimed for the birthday paradox. The birthday paradox has various applications; the most evident of which being in the context of this thesis, i.e., collision-finding for hash functions. For more on the birthday paradox, we refer to [122].

B Compression Functions Based on Fixed-key Blockciphers

Conventional blockciphers require two auxiliary algorithms to encrypt a plaintext: (i) a key-scheduling algorithm for processing and expanding the secret-key to create the so-called sub-keys and (ii) a data-processing algorithm for processing the plaintext by using the sub-keys derived from the key-scheduling algorithm. For any fixed key (and hence sub-keys) the data-processing algorithm creates a fixed permutation over the domain of plaintexts (as previously discussed).

In spite of satisfactory security properties, hash functions created via blockcipher-based compression functions in an iterated manner face a major efficiency problem: For each invocation of the compression function, we need to perform both data-processing and key-scheduling (the latter usually requires as much time as data-processing). Avoiding the former is impossible as one needs the permutation anyway; so what about avoiding key-scheduling entirely by simply considering fixed-key blockciphers (i.e., permutations)?

The idea of creating compression functions based on fixed-key blockciphers stems from this efficiency issue. Defining permutation-based compression functions and related security properties is analogous and follows directly from Section 2.3. The obvious caveat is that this time our idealized primitives are permutations $\pi : \{0,1\}^n \rightarrow \{0,1\}^n$ drawn uniformly from the set $\mathrm{Perm}(n)$ of all permutations over $\{0,1\}^n$. Below we briefly look at the advances in permutation-based hashing; in Section B.1, we recapitulate the results on single-call single-block-length permutation-based compression functions. In Sections B.2 and B.3, we study the multi-call setting and some other extensions.

B.1 Single-Call Single-Block-Length Compression Functions

Single-call single-block-length permutation-based compression functions can be defined via Definition 2.3.6 with $\kappa = 0$ and π as the underlying permutation (see Figure B.1). For simplicity, let us also assume that $m = s = n$. These type of compression functions and corresponding iterated hash functions (via sMD) were studied by Black, Cochran and Shrimpton [25] who consider, in a more general setting, a small non-empty set of blockcipher keys and the single-call single-block-length blockcipher-based compression functions with blockcipher using only keys from the pre-selected set of keys.

They show (in the ideal cipher model), contrary to the results of Black, Cochran and Shrimpton

on the blockcipher-based setting, that any compression function constructed as just described cannot result in a provably collision-resistant hash function when iterated. More precisely, they present highly efficient information-theoretic collision attacks, efficient in the sense that they work with a very small number of queries. Therefore, the result of Black, Cochran and Shrimpton can be interpreted as an impossibility result against the provability of single-call single-block-length permutation-based compression functions (in the information-theoretic sense).

Note however that Black, Cochran and Shrimpton's work does not say anything in the computational setting. Indeed their attacks do not mean that there exist practical (with respect to computational complexity) attacks on these hash functions: Finding sub-exponential time attacks is still an open problem although the query-complexity is really low. Extending the work of Black, Cochran and Shrimpton was done in the subsequent years, by considering multi-call permutation-based compression functions where, contrary to the single-call setting, we can achieve several optimality results.

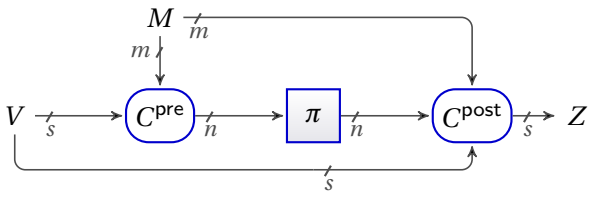

Figure B.1 – General form of single-call permutation-based compression functions. The case $m = s = n$ corresponds to the constructions studied by Black, Cochran and Shrimpton.

B.2 Multi-Call Compression Functions

As a follow-up work of Black, Cochran and Shrimpton, Rogaway and Steinberger [174] study the security-efficiency trade-offs for permutation-based hash functions by investigating how much security we would gain if the number of permutation calls is increased. As a result, they conclude (under an assumption) that any $2n \to n$ bits permutation-based compression function needs at least three permutation calls to have an optimal collision resistance, while a DBL construction has to make at least five calls (cf. Conjecture 2.3.5) to beat the bound $2^{n/2}$ (which is still suboptimal). In [172, 173], Rogaway and Steinberger also provide concrete proposals matching their stated (lower) bounds and number of permutation calls (see Figure B.2 for an example that employs three permutation calls).

The main ingredient of the Rogaway–Steinberger framework is the use of a matrix over $\mathbb{F}_{2^n}$ satisfying an independence criterion [173, 174]. This matrix is used to determine, for all i, the input x_i of the permutation π_i (as well as the digest) given the compression function input and the previous input-output pairs $(x_j, \pi_j(x_j))$ (for $j < i$). It should be noted, however, that the independence criterion is just a sufficient condition to get the stated lower bounds. Indeed, the three-call permutation-based compression function suggested by Shrimpton and Stam [186] (see Figure B.3) falls also under this general framework although their matrix does not satisfy the independence criterion.

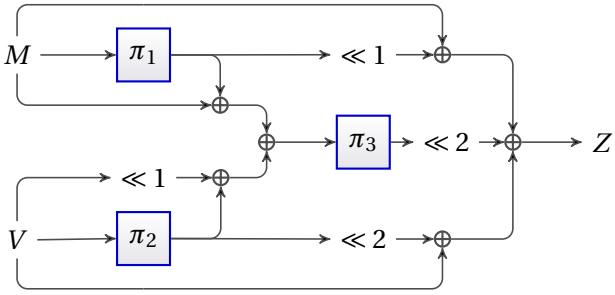

Figure B.2 – The Rogaway–Steinberger compression function is illustrated. Here $\ll 1$ and $\ll 2$ denote polynomial multiplication with x and x^2, respectively.

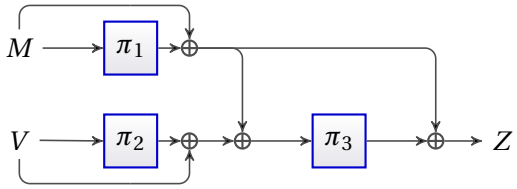

Figure B.3 – The Shrimpton–Stam compression function is illustrated.

B.3 Extensions

Several (current and old) SHA-3 candidates instantiate their compression functions using permutations. Some of the examples are CubeHash [17], Hamsi [100], JH [206], Keccak [19], Lane [80], Grøstl [67] and Luffa [45]. The common characteristic of these hash functions, besides being permutation-based, is that their compression functions do no behave ideally no matter how ideal the underlying permutations are. Hence, the only way to improve security is to properly iterate the compression function and use an output transformation, if necessary (some of these designs are constructed via close variants of sMD mentioned in Section 2.2.3). We refer to [20, 172, 190] for more on permutation-based hashing.

C Our Results on Extended KP-Parameters

The attacks presented in Sections 6.3 and 6.4 focus on the KP-suggested compression functions that can potentially be instantiated with single- or double-key blockciphers running in the Davies–Meyer mode. However, Knudsen and Preneel [93] also propose other parameters using $5n \to n$ PuRFs as the underlying primitives using MDS codes over $\mathbb{F}_{2^5}$. Although conventional blockciphers support at most double-key scenarios, some of the well-known compression functions, such as MD4 and MD5, can be regarded as $5n \to n$ blockcipher-based compression functions having n-bit block and $4n$-bit key. Fortunately, our techniques can be used to analyze this class of compression functions as well; we summarize the ramifications of our results in Table C.1.

Code	**Preimage Resistance Complexity**			**Collision Resistance Complexity**		
	Query	Time	Memory	Query	Time	Memory
$[r,k,d]_{2^e}$	$2^{rn/k}$	Thm. 6.3.5	Thm. 6.3.12	$2^{kn/(3k-r)}$	Thm. 6.4.17	Thm. 6.4.17
$[5,3,3]_{32}$	$2^{5n/3}$	$2^{5n/3}$	$2^{2n/3}$	$2^{3n/4}$	$2^{3n/4}$	$2^{3n/4}$
$[10,8,3]_{32}$	$2^{5n/4}$	$2^{5n/4}$	$2^{n/2}$	$2^{4n/7}$	$2^{4n/7}$	$2^{4n/7}$
$[20,18,3]_{32}$	$2^{10n/9}$	$2^{10n/9}$	$2^{5n/9}$	$2^{9n/17}$	$2^{9n/17}$	$2^{9n/17}$
$[5,2,4]_{32}$	$2^{5n/2}$	2^{3n}	$2^{3n/2}$	×	×	×
$[10,7,4]_{32}$	$2^{10n/7}$	2^{2n}	$2^{6n/7}$	$2^{7n/11}$	$2^{9n/11}$	$2^{7n/11}$
$[20,17,4]_{32}$	$2^{20n/17}$	2^{2n}	$2^{12n/17}$	$2^{17n/31}$	$2^{28n/31}$	$2^{21n/31}$

Table C.1 – Our results on $5n$-to-n bit primitive (PuRF or blockcipher) Knudsen–Preneel Compression Functions.

Curriculum Vitae – Onur Özen

Personal Information

Born on October 5th, 1983
Turkish Citizen

Contact

École Polytechnique Fédérale de Lausanne
Faculté Informatique et Communications
Laboratory for Cryptologic Algorithms
INJ 332, Station 14, CH-1015 Switzerland
onur.ozen@epfl.ch

Education

Ph.D. [February 2008 → Present]
École Polytechnique Fédérale de Lausanne
Faculty of Computer and Communication Sciences
Laboratory for Cryptologic Algorithms, Lausanne, Switzerland
Supervisor: Prof. Arjen K. Lenstra
Thesis Title: Design and Analysis of Multi-Block-Length Hash Functions
Expected Graduation: 2012

M.Sc. [September 2006 → January 2008]
Middle East Technical University
Institute of Applied Mathematics,
Department of Cryptography, Ankara, Turkey
Supervisor: Assoc. Prof. Dr. Ali Doğanaksoy
Thesis Title: On the Security of Tiger Hash Function
GPA: 4.00/4.00

B.Sc. [September 2001 → June 2006]
Middle East Technical University
Department of Mathematics, Ankara, Turkey
GPA: 3.47/4.00 (Honor's Graduate)

Refereed Papers

Dimitar Jetchev, Onur Özen, Martijn Stam: Foundations of Efficient Hashing: Collisions are not Incidental: To appear in the Proceedings of the 9th Theory of Cryptography Conference, TCC 2012.

Joppe W. Bos, Onur Özen, Martijn Stam: Efficient Hashing Using the AES Instruction Set. In: Preneel,B., Takagi,T. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2011, 13th International Workshop, Nara, Japan, September 28-October 1, 2011. Lecture Notes in Computer Science, vol. 6917, pp. 507-522. Springer, Heidelberg (2011).

Onur Özen, Martijn Stam: Collision Attacks against the Knudsen-Preneel Compression Functions: In: Abe, M. (ed.) Advances in Cryptology, ASIACRYPT 2010, 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Lecture Notes in Computer Science, vol. 6477, pp. 76-93. Springer, Heidelberg (2010).

Onur Özen, Thomas Shrimpton, Martijn Stam: Attacking the Knudsen-Preneel Compression Functions: In: Hong, S., Iwata, T. (eds.) Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010. Lecture Notes in Computer Science, vol. 6147, pp. 94-115. Springer, Heidelberg (2010).

Onur Özen, Martijn Stam: Another Glance at Double-Length Hashing: In: Parker, M.G. (ed.) Cryptography and Coding, 12th IMA International Conference, Cryptography and Coding 2009, Cirencester, UK, December 15-17, 2009. Lecture Notes in Computer Science, vol. 5921, pp. 176-201. Springer, Heidelberg (2009).

Jean-Philippe Aumasson, Çağdaş Çalık, Willi Meier, Onur Özen, Raphael C.-W. Phan, Kerem Varıcı: Improved Cryptanalysis of Skein. In: Matsui, M. (ed.) Advances in Cryptology, ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Lecture Notes in Computer Science, vol. 5912, pp. 542-559. Springer, Heidelberg (2009).

Joppe W. Bos, Onur Özen, Jean-Pierre Hubaux: Analysis and Optimization of Cryptographically Generated Addresses. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) Information Security, 12th International Conference, ISC 2009, Pisa, Italy, September 7-9, 2009. Lecture Notes in Computer Science, vol. 5735, pp. 17-32. Springer, Heidelberg (2009).

Onur Özen, Kerem Varıcı, Cihangir Tezcan, Çelebi Kocair: Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT. In: Boyd, C., Nieto, J.M.G. (eds.) Information Security and Privacy, 14th Australasian Conference, ACISP 2009, Brisbane, Australia, July 1-3, 2009, Lecture Notes in Computer Science, vol. 5594, pp. 90-107. Springer, Heidelberg (2009).

Onur Özen, Kerem Varıcı: On the Security of the Encryption Mode of Tiger. 3rd Information Security and Cryptology Conference (ISC Turkey), 2007. (Also Presented in Ecrypt Tools for Cryptanalysis Workshop, 2007)

Paper Reviews

2008: Asiacrypt'08, Indocrypt'08, SAC'08

2009: Asiacrypt'09, ISC'09, SHARCS'09

2010: Asiacrypt'10, Eurocrypt'10, Financial Cryptography'10, Latincrypt'10, SAC'10, SCN'10, Ecrypt Tools for Cryptanalysis'10

2011: Asiacrypt'11, Crypto'11, Eurocrypt'11, FSE'11, Financial Cryptography'11